# The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool*

W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin
Microsystems Design Lab
The Pennsylvania State University
University Park, PA 16802
(wye,vijay,kandemir,mji)@cse.psu.edu

## ABSTRACT

In this paper, we present the design and use of a comprehensive framework, *SimplePower*, for evaluating the effect of high-level algorithmic, architectural, and compilation tradeoffs on energy. An execution-driven, cycle-accurate RT level energy estimation tool that uses transition sensitive energy models forms the cornerstone of this framework. *SimplePower* also provides the energy consumed in the memory system and on-chip buses using analytical energy models.

We present the use of *SimplePower* to evaluate the impact of a new selective gated pipeline register optimization, a high-level data transformation and a power-conscious post compilation optimization (register relabeling) on the datapath, memory and on-chip bus energy, respectively. We find that these three optimizations reduce the energy by 18-36% in the datapath, 62% in the memory system and 12% in the instruction cache data bus, respectively.

## 1. INTRODUCTION

Power[1] consumption has become a critical issue in processor design, especially in embedded environments. When designing high-performance, low-power processors, designers need to experiment with software and architectural level tradeoffs and evaluate various power optimization techniques. Architectural level power estimation tools are becoming increasingly important with the growing complexity of current designs (Systems-on-a-Chip) to provide fast estimates of the energy consumption early in the design cycle. By the time the design of today's large and complex processors have been specified to the circuit or gate level, it may be too late or too expensive to go back and deal with excess power consumption problems. Further, software is becoming an important aspect of emerging embedded systems and the study of the integrated impact of software and hardware optimizations should be supported by such tools.

In this paper, we present the design of an execution-driven, cycle-accurate, RT level power estimation tool, *SimplePower*, and its use in evaluating algorithmic, architectural and compiler optimizations. Most research in architectural level power estimation is based on empirical methods that measure the power consumption of existing implementations and produce models from those measurements. This is in contrast to approaches that rely on information theoretic measures of activity to estimate power [12; 16]. Measurement based approaches for estimating the power consumption of datapath functional units can be divided into three sub-categories. The first technique, introduced by Powell and Chau [15], is a fixed-activity macro-modeling strategy called the Power Factor Approximation (PFA) method. The energy models are parameterized in terms of complexity parameters and a PFA proportionality constant. Similar schemes were also proposed by Kumar et. al. in [8] and Liu and Svensson in [11]. This approach assumes that the inputs do not affect the switching activity of a hardware block. To remedy this problem, activity-sensitive empirical energy models were developed. These schemes are based on predictable input signal statistics; an example is the method proposed by Landman and Rabaey [9]. Although the individual models built in this way are relatively accurate (a 10% - 15% error rate), overall accuracy may be sacrificed due to incorrect input statistics or the inability to model the interactions correctly. The third empirical technique, transition-sensitive energy models, is based on input transitions rather than input statistics. The method, proposed by Mehta, Irwin, and Owens [13], assumes an energy model is provided for each functional unit - a table containing the power consumed for each input transition. Closely related input transitions and energy patterns can be collapsed into clusters, thereby reducing the size of the table. Other researchers have also proposed similar macro-model based power estimation approaches [22; 1].

The *SimplePower* energy simulator was developed based on transition-sensitive energy models and is an execution-driven, cycle-accurate RT level tool. It simulates the integer subset of the instruction set of Simplescalar, which is a suite

*(This work was sponsored in part by grants from NSF (MIP-9705128), Sun Microsystems, and Intel)

[1] Throughout this paper, we often use the terms power and energy interchangeably. However, power savings and energy savings do not necessarily go hand-in-hand. The choice of metric used will depend on the application constraints.

of publicly available tools to simulate modern microprocessors [2]. Our simulation flow uses the Simplescalar compiler toolset to convert the C source benchmarks to *SimplePower* executables. *SimplePower* simulates these executables providing cycle-by-cycle energy estimates and the switch capacitance statistics for the processor datapath, memory and on-chip buses. Currently, *SimplePower* does not capture the energy consumed by the control unit of the processor and the clock generation and distribution network.

The rest of this paper consists of four sections. Section 2 presents the design of *SimplePower*. The effectiveness of a selectively gated pipeline register optimization on the processor datapath power is evaluated in Section 3. Section 4 discusses the effectiveness of compiler optimizations on the energy consumed by the system components. Finally, Section 5 draws the conclusions.

## 2. DESIGN OF SIMPLEPOWER

*SimplePower* is based on the architecture of a five-stage pipelined datapath and consists of the fetch stage IF, the instruction decode stage ID, the execution stage EXE, the memory access stage MEM, and the write-back stage WB. The instruction set architecture of simulated machine is a subset of the instruction set (the integer part) of Simplescalar, which is a suite of publicly available tools to simulate modern microprocessors [2]. The major components of *SimplePower* are: *SimplePower core*, *RTL power estimation interface*, *technology dependent switch capacitance tables*, *cache/bus simulator*, and *loader*.

At each clock cycle, the *SimplePower Core* simulates the execution of all active instructions and calls corresponding power estimation interfaces for all activated functional units. It continues the simulation until the program halt instruction is fetched. Once the simulator fetches the halt instruction, it continues executing all the instructions left in the pipeline and then presents the output. In order to keep the simulator technology independent, a *RTL power estimation interface* - a set of C routines - has been developed for all the architectural level functional units. The parameters of each interface module are the same as the inputs of the standard functional unit. If the architecture of a functional unit is changed, only the power table(s) or its interface implementation needs to be replaced while the simulator core does not need any modification. *Technology dependent switch capacitance tables* have been developed for the different functional units such as adders, ALU, multipliers, shifter, controllers, register file, pipeline registers, and multiplexors. The *cache simulator* simulates the status of the instruction cache and data cache. It is called by *SimplePower Core* when either cache (I-Cache or D-Cache) is accessed. The cache simulator was built by modifying the DineroIII cache simulator [4] and integrated with the memory energy model proposed in [18]. This analytical model has been validated to be accurate (within 2.4% error) for conventional cache systems [5; 18]. The *bus simulator* snoops the instruction cache address bus, the instruction cache data bus, the data cache address bus, and the data cache data bus. It records the total number of accesses and the number of transitions on those buses. Those statistics are combined with our interconnect power model [23] to compute the switch capacitances of the on-chip buses.

*SimplePower core* accesses a table (through the *interface*) containing the switch capacitance for each input transition of the functional unit exercised. Table 1 shows the structure of such a table for a $n$-input functional unit.

| Index: previous input vector | Index: current input vector | Switch Capacitance (pF) |
|---|---|---|
| $0_1 \ldots 0_n$ | $0_1 \ldots 0_n$ | $cap_0$ |
| $0_1 \ldots 0_n$ | $0_1 \ldots 1_n$ | $cap_1$ |
| $0_1 \ldots 0_n$ | $0_1 ..10_n$ | $cap_2$ |
| $0_1 \ldots 0_n$ | $0_1 ..11_n$ | $cap_3$ |
| ... | ... | ... |
| $1_1 \ldots 1_n$ | $1_1 ..10_n$ | $cap_{2^n-2}$ |
| $1_1 \ldots 1_n$ | $1_1 ..11_n$ | $cap_{2^n-1}$ |

**Table 1: Switch Capacitance Table**

The construction of these tables is based on the structure of the functional units. Each functional unit can be divided into one of the following classes: *bit-independent functional units* or *bit-dependent functional units*. In a bit-independent functional unit, the operation for each bit slice does not depend on the values of other bit slices. In this case, the only switch capacitance table we need is a small table for a one-bit slice. The total energy consumed by the module can be calculated by summing the energy consumed by each bit transition. Examples of bit-independent functional units include pipeline registers, the logic unit in the ALU, latches and buses.

In a bit-dependent functional unit, the operation in one bit slice depends on the values of other bit slices (for example, a 32-bit adder). Their energy characterization is based on a table lookup consisting of a full energy transition matrix where the row address is the previous input vector, the column address is the present input vector, and the matrix value is the switch capacitance (as shown in Table 1). Unfortunately, the size of this table grows exponentially with the size of the inputs. A clustering algorithm helps with this problem [13] while bounding the loss in accuracy. Although this algorithm can compress the table for a 16-bit ripple carry adder (with $2^{32}$ entries) to a table with only 97 entries, it is very difficult to compress the table for a 32-bit adder with acceptable accuracy. Lin et. al. proposed a power modeling and characterization method for functional units (called *macrocells* in their paper) using structure information [10]. However, their technique was more suitable for circuits with small input size. For instance, it took 29.3 hours to simulate a 4-bit fast adder (9 inputs) with a reduced number of entries (26,244). Two solutions have been used in our work to address this problem:

- If the structure of a module is too complicated (or large), analytical (transition independent) modeling can be used to estimate the power consumed in this module. In this case, the simulator only needs to record the necessary parameters. Currently, we use an analytical memory system power model.
- If a functional unit can be partitioned into smaller sub-modules, we can build the tables for the sub-modules first and then use these tables to estimate the total switched capacitance of the entire functional unit.

We have applied this technique to model adders, subtracters, shifters, multipliers, register files, decoders, and multiplexors.

Figure 1 shows the estimated power consumption using the functional partitioning approach (X-axis) versus the actual circuit level simulation results (Y-axis) from HSPICE for a 5-port register file. The accuracy of our technique is determined by comparing the HSPICE simulation results and the estimated power consumption from our simulator for 15 randomly chosen input transitions. The average error of our approach from the HSPICE simulations was found to be 13%. Figure 1 also shows that our method underestimated the power consumed by each input transition and that all the points are distributed linearly. A constant multiplicative factor based on the technology (between 1.1 and 1.2) can be used to improve the accuracy. For the 32×32 5-port register file, our power estimation approach took much less than 0.1 sec for each input transition as opposed to the 556.42 sec required for circuit level simulation using HSPICE. The machine running the HSPICE simulation and our simulator is a SUN ULTRA-10 with 640 MBytes memory.

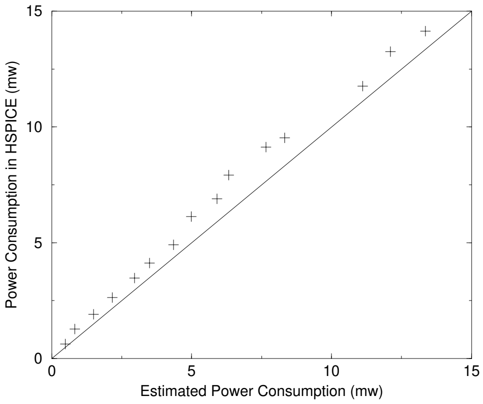


**Figure 1: 5-port register file: Estimated power consumption vs. HSPICE results. Ideally, all points should fall on the straight line.**

We have designed the shifter, adders, multipliers and divider energy models using an approach similar to that of the register file. The average error from HSPICE evaluations was found to be within 15% for all the units. In the following sections, we describe the use of *SimplePower* for analyzing the effectiveness of tradeoffs in algorithms, architectural design and compiler optimizations.

## 3. GATING PIPELINE REGISTERS

*SimplePower* was used to study different architectural optimizations using a set of benchmarks listed in Table 2. The *SimplePower* simulation flow used for estimating the energy of these applications is given below. First, the C source benchmark is compiled by the *SimpleScalar* version of `gcc`, which generates *SimpleScalar* assembly codes. The *SimpleScalar* assembler `gas` and loader (linker) `gld` produce *SimplePower* executables that can then be loaded into *SimplePower's* main memory and executed by *SimplePower* core. Users can specify options to configure the caches, to output the pipeline trace cycle-by-cycle and to dump the memory image. Besides the optional outputs, *SimplePower* provides the register file final status, the total number of cycles in execution, the number of transitions in the buses, switch capacitance statistics for each pipeline stage, switch capacitance statistics for different functional units, and the total switch capacitance.

| Name | Brief Description | # of Cycles |
|---|---|---|
| acker.c | calculate *Ackermann's Function* $A(3,6)$ | 5,339,731 |
| bubble.c | bubble sort 100 random numbers | 391,386 |
| heap.c | heap sort the same 100 numbers | 105,368 |
| quick.c | quick sort the same 100 numbers | 67,531 |
| bsrch.c | binary search the same 100 numbers | 373 |
| hanoi.c | *Tower of Hanoi* for 1 to 10 disks | 220,806 |
| fib.c | find *Fibonacci Number* $F_{30}$ | 47,426,259 |
| matm.c | 4x4 matrix multiplication | 3,994 |
| perm.c | permutation of 7 numbers | 751,777 |
| queens.c | find all the solution for *10-Queens Problem* | 468,446 |
| sieve.c | using an 8KB array to find some prime numbers by using *Sieve of Eratosthenes* | 689,284 |

**Table 2: Benchmark Set**

We obtained energy results using *SimplePower* and observed that most of the power in the processor core is spent in the gated pipeline registers (around 40% of datapath power using $0.35\mu$) for the unoptimized pipeline (nongated). Thus, the selective gated pipeline register optimization focuses on reducing the power consumption of the pipeline registers. In a pipelined datapath, the multiple stages are separated by different pipeline registers with large width. Those pipeline registers unconditionally latch their inputs to their outputs when the pipeline is active. As a result, those pipeline registers consume a large amount of energy. Thus, we propose a simple technique to reduce the pipeline register switching activity by using the control signals of the datapath for *selectively* gating subsets of the pipeline registers. To evaluate the effectiveness of this scheme, the simulator was modified to get the total pipeline register switch capacitance with and without selectively gated pipeline registers.

Pipeline registers latch their data inputs to their outputs unconditionally when the evaluating clock edge arrives. Usually, each pipeline register contains two types of inputs: *control* and *data*. The behavior of the functional units following a pipeline register is controlled by the corresponding control signals. If the control signals are not active for a functional unit, the latchings of the data inputs of that functional unit are not necessary since the data will not be used. *Thus, we can gate the data portion of the pipeline register by using the corresponding control signals.* The advantage is that no extra logic is needed to generate the signals that are used to gate the clock signal and only the clock gating logic needs to be added. At the same time, since all the data bits can share the same gated clock, the control overhead is small. For pipeline register MEM/WB shown in Figure 2, fields *MemData* (32 bits), *AluOut* (32 bits), *RtData* (32 bits) and *Writereg0* (5 bits) can be gated by the control signal *Regwrite0* (1 bit) contained in the control field *wb_cntl*. These

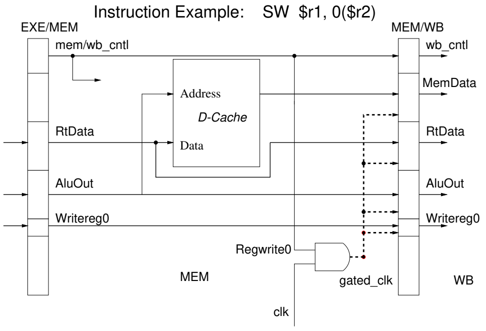


**Figure 2: Selectively Gated Clock for** *MemData*, *RtData*, *AluOut*, **and** *Writereg0*

data fields can be gated because *Regwrite0* is set based on whether the executed instruction writes back into the register file. Since *Regwrite0* is active high, the gated clock for these fields can be implemented as shown in Figure 2. Because the clock gating logic is shared by all the D flip-flops in those fields, its power overhead is found to be insignificant. Many similar cases of selective gating of data fields in the various pipeline registers occur.

It was observed that 23-36% (18-33%) energy reduction is possible in the processor datapath using this scheme for the various benchmarks. Also, we observed that the energy consumed by the pipeline registers is reduced by more than 50%. Further, the register file power decreases by up to 46% due to the elimination of spurious transitions that occur on the decoder inputs and write data drivers. However, the functional units do not show a significant improvement, because they already had latches present on their individual inputs to support data forwarding. Similar architectural alternatives can be investigated using *SimplePower* to make early energy-conscious design choices.

# 4. MEMORY/BUS OPTIMIZATIONS

In this section, the effect of traditional performance-oriented and new power-aware complier optimizations on energy consumption is evaluated using *SimplePower*. In particular, we study two optimizations that focus on the energy consumed by the memory system and the instruction cache (Icache) bus. An optimizing compiler framework to perform both high-level source-to-source transformations and low-level optimizations was built over the existing Simplescalar tools and interfaced with *SimplePower*. This is a powerful framework for analyzing the impact of a spectrum of compiler optimizations on each individual component of the system and the system as a whole. Among various high-level loop and data transformations that have been evaluated using *SimplePower* [7], we present the results of data transformations on the memory system, in particular, and the system (core+memory) as a whole. *SimplePower* can also evaluate the influence of low-level transformations such as instruction scheduling, operand re-ordering, register assignment and code motion on system energy. Here, we also present the application of a power-aware post compilation optimization that reduces the Icache data bus power. A new register relabeling algorithm is proposed for minimizing the transition on the Icache data bus and evaluated using *SimplePower*.

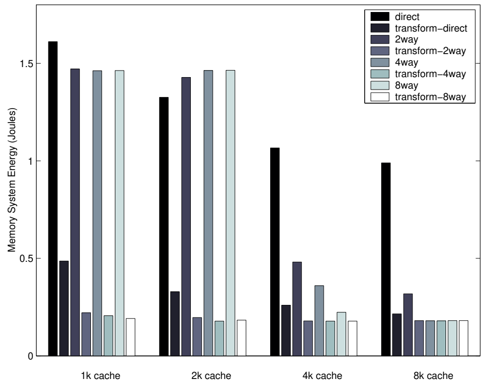


**Figure 4: Memory System Energy Optimization Using Data Transformation**

## 4.1 Memory System Power Optimization

Recently, data transformations have been proposed to improve spatial locality in situations where loop transformations are not effective [6]. These transformations, instead of changing the loop execution order, modify the underlying memory layouts of multidimensional arrays. Since these layout modifications, in a sense, correspond to variable-renaming operations, they are always legal provided that they are applied globally (i.e., program-wide). This is in contrast with loop transformations where data dependences must be maintained. As an example, we consider the matrix multiplication code shown in Figure 3(a). In this code, array `a` has spatial locality, array `c` has temporal locality, and `b` has no locality in the *innermost* loop. A data transformation framework converts the layout of the array `b` from row-major (the default layout used by C) to column-major to exploit spatial reuse (stride-one accesses) for that array. The resulting transformed code is shown in Figure 3(b).

The original and transformed codes were then simulated using *SimplePower* for different data cache configurations and a main memory size of 2Mbits. The other configuration parameters for all the simulations were a cache line size of 32 bytes and $0.8\mu$, 3.3V technology. Figure 4 shows the results of memory system energy consumption before and after applying the data transformation. The increased spatial locality provided by the high-level compiler transformation produces significant energy savings for smaller cache sizes (e.g., 1K and 2K) and associativities (direct, 2-way). However, the data cache locality is not as much of a concern when the data cache size is relatively large. The applied data transformation does not affect the processor core consumption (except for stall cycle power which is negligible) as it just manipulates the data layouts. The energy consumed by the core was 0.03 Joules for both these codes. While the data transformation presented for matrix multiplication is very effective in reducing the energy consumed in the memory system, we observed that some other data transformations generate very complex array subscript functions which, in turn, increases the core energy.

## 4.2 Bus Power Optimization

Many prior efforts have focused on encoding techniques for data buses such as gray code, bus invert code, sign magni-

```
for(i=0;i<N;i++)
 for(j=0;j<N;j++)
  {m= c[i][j];
   for(k=0;k<N;k++)
    m = m + (a[i][k] * b[k][j]);
   c[i][j] = m;
  }
```
**(a)**

```
for(i=0;i<N;i++)
 for(j=0;j<N;j++)
  {m= c[i][j];
   for(k=0;k<N;k++)
    m = m + (a[i][k] * b[j][k]);
   c[i][j] = m;
  }
```
**(b)**

**Figure 3: (a) Original Code (b) Code after applying data transformation on array** `b[][]`

tude and have reported 18 to 40% savings in energy [17]. However, these encodings are applied to address (instruction and data cache) buses and data cache data buses as opposed to the instruction encoding proposed in this work. There have also been various efforts at reducing the overall power dissipation between consecutive instructions using low-level compiler techniques. Towards this goal, Su and Despain [19] proposed a technique that combines Gray code addressing and instruction scheduling. Based on basic-block list scheduling, their approach uses a greedy algorithm to re-order the instructions to minimize power consumption. Tiwari, Malik, and Wolfe [20] proposed another scheduling technique which also tries to select instructions more judiciously to minimize power. Toburen et al. [21] presented a method for instruction scheduling which limits the number of instructions that can be scheduled in a given cycle depending on the power constraints.

In this section, we focus on reducing the switching activity on the Icache data bus (between the processor core and Icache) by relabeling the register fields of the compiler generated instructions. Sample traces are used to record the transition frequencies between register labels (encodings) in the instructions executed in consecutive cycles using *SimplePower*. This information is then used to obtain new encodings for the registers such that the switching activity (and consequently the energy consumption) in the Icache data bus is reduced. The technique is applicable to any system where switching activity imposed by register labels has an impact on overall energy consumption. The encoding of instructions using relabeling can be considered similar to data encoding techniques investigated for data.

To illustrate the idea, let us consider two consecutive `add` instructions in the *SimplePower* assembly language:

`add` $s_i, s_j, s_k$
`add` $s_l, s_m, s_n$

In this simple sequence, there are three switching activities between registers: (1) from $s_i$ to $s_l$ in the destination-register slot, (2) from $s_j$ to $s_m$ in the first source-register slot, and (3) from $s_k$ to $s_n$ in the second source-register slot. Depending on the encodings of the registers involved, the impact of these switching activities can be quite significant. These *register transitions* can also occur between different types of instructions.

It should be clear that for register fields that have frequent transitions, we need to use register numbers whose Hamming distance is minimum. The problem is that register assignments are done by the compiler using sophisticated algorithms and considering a number of other important issues such as minimizing register spills and maximizing register reuse [14]. Therefore, we cannot arbitrarily change the register numbers, just to minimize power consumption. Such modifications, among other things, can also violate data dependences across instructions, thereby changing the semantics of the program being optimized. On the other hand, other alternatives, namely, determining a near-optimal register assignment considering both power and performance is very difficult.

We developed a post-pass, polynomial-time algorithm that relabels the registers *after* global register allocation performed by the compiler back-end. Relabeling registers is always legal as long as it is performed throughout the code. For example, if we decide to relabel $s_i$ as $s_j$, *all* the occurrences of $s_i$ should be changed to $s_j$. Also, if we are to perform relabeling for multiple pairs, this should be done simultaneously for all pairs. Informally, our post-pass algorithm takes a compiler-generated register assignment (register allocation) as input and generates an alternative assignment that reduces the power, maintaining the same performance as the original assignment.

The register relabeling optimization was incorporated in the *SimplePower* compilation framework by modifying the Simplescalar toolset. Table 3 shows the energy reduction of the instruction cache (Icache) data bus for $0.35\mu m$ technology. For Icache data bus between the processor core and the Icache, the wire switch capacitance is estimated to be 0.7105pF per bit transition for $0.35\mu m$ using the interconnect model proposed in [23]. We observe a 12% reduction in the total energy reduction in the Icache data bus using the register relabeling optimization. With the growing number of VLIW designs, we expect the savings on the Icache data bus to be a significant portion of the system energy. In such designs, the instructions are fed to multiple processing unit on the same chip, resulting in a larger capacitive load on the Icache data bus. Further, these VLIW processors use a larger number of register fields in their instructions. Hence, a larger portion of the Icache data bus can benefit from the proposed optimization.

## 5. CONCLUSIONS

In this paper, the design and use of a comprehensive framework, *SimplePower*, for evaluating the effect of high level algorithmic, architectural, and compilation trade-offs on energy was presented. Design issues in the development of the transition sensitive energy macro-models used by *SimplePower* were presented and validated using the register

| Benchmark | Original (nF) | Relabeled (nF) | Reduction (nF) |
|---|---|---|---|
| acker.c | 17,742.8 | 15,784.5 | 11.04% |
| bsrch.c | 0.936 | 0.843 | 9.94% |
| bubble.c | 1,033.8 | 870.5 | 15.79% |
| fib.c | 166,117.1 | 146,017.5 | 12.10% |
| hanoi.c | 690.9 | 600.7 | 13.05% |
| heap.c | 338.6 | 302.7 | 10.61% |
| matm.c | 10.075 | 8.372 | 16.90% |
| perm.c | 2,459.6 | 2,284.5 | 7.12% |
| queens.c | 1,398.7 | 1,306.0 | 6.62% |
| quick.c | 194.8 | 168.6 | 13.47% |
| sieve.c | 1,902.2 | 1,659.5 | 12.76% |
| Average | - | - | 11.76% |
| Std.Deviation | - | - | 3.04% |

**Table 3: ICache Data Bus Switch Capacitance Reduction (0.35$\mu m$)**

file as an example. As its applications, we use *SimplePower* in evaluating the impact of an architectural modification, a data transformation applied on a simple benchmark and a back-end compiler (register relabeling) optimization on the datapath, memory and on-chip bus energy, respectively. We proposed a simple, selective gated pipeline optimization technique which can reduce the datapath switch capacitance up to 36% for the tested applications. A high-level data transformation is used to reduce the memory system power to a third. In contrast to other works, *SimplePower* is useful in optimizing the energy of the system as a whole and can capture the memory energy savings against that of the datapath. This capability is important when applying certain high-level transforms such as loop tiling that increase datapath energy. Finally, we propose a new energy-conscious register relabeling optimization and find that it reduces the energy consumed by the instruction cache data buses by 12%.

# 6. REFERENCES


[1] L. Benini, A. Bogoliolo, M. Favalli, and G. D. Micheli. Regression models for behavioral power estimates. In *Proceedings of International Workshop on Power, Timing, Modeling, Optimization and Simulation*, page 179, September 1996.

[2] D. Burger and T. Austin. The simplescalar tool set, version 2.0. Technical report, Computer Sciences Department, University of Wisconsin, June, 1997.

[3] M. Cierniak and W. Li. Unifying data and control transformations for distributed shared memory machines. In *Proceedings of SIGPLAN'95 Conference on Programming Language Design and Implementation*, June 1995.

[4] M. D. Hill, J. R. Larus, A. R. Lebeck, M. Talluri, and D. A. Wood. Wisconsin architectural research tool set (warts). *Computer Architecture News (CAN)*, August 1993.

[5] M. Kamble and K. Ghose. Analytical energy dissipation models for low power caches. In *Proceedings of International Symposium on Low Power Electronics and Design*, page 143, August 1997.

[6] M. Kandemir, A. Choudhary, J. Ramanujam, and P. Banerjee. Improving locality using loop and data transformations in an integrated framework. In *Proceedings of MICRO-31,* Dallas, TX, December, 1998.

[7] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of compiler optimization on system power. In *Proceedings of the 37th Design Automation Conference*, 2000.

[8] N. Kumar, S. Katkoori, L. Rader, and R. Vemuri. Profile-driven behavioral synthesis for low power vlsi systems. *IEEE Design and Test Magazine*, page 70, Fall 1995.

[9] P. Landman and J. Rabaey. Activity-sensitive architectural power analysis. *IEEE Transaction on CAD*, TCAD-15(6), page 571, June 1996.

[10] J. Lin, W. Shen, and J. Jou. A power modeling and characterization method for macrocells using structure information. In *International Conf. on Computer Aided Design*, page 502, November 1997.

[11] D. Liu and C. Svensson. Power consumption estimation in cmos vlsi chips. *IEEE Journal of Solid-State Circuits*, page 663, June 1994.

[12] D. Marculescu, R. Marculescu, and M. Pedram. Information theoretic measures of energy consumption at register transfer level. In *Proceedings of 1995 International Symposium on Low-Power Design*, page 81, April 1995.

[13] H. Mehta, R. M. . Owens, and M. J. Irwin. Energy characterization based on clustering. In *Proceedings of the 33rd Design Automation Conference*, page 702, June 1996.

[14] S. S. Muchnick. *Advanced Compiler Design Implementation.* Morgan Kaufmann Publishers, San Francisco, California, 1997.

[15] S. Powell and P. Chau. Estimating power dissipation of vlsi signal processing chips: the pfa technique. In *VLSI Signal Processing, IV*, page 250, 1990.

[16] J. M. Rabaey and M. Pedram. *Low Power Design Methodologies.* Kluwer Academic Publishers, Inc., 1996.

[17] J. Sacha and M. J. Irwin. Number representation for reducing data bus power dissipation. In *Proceedings of the 33rd Asilomar Conference on Signals, Systems, and Computers*, November 1998.

[18] W.-T. Shiue and C. Chakrabarti. Memory exploration for low power, embedded systems, clpe-tr-9-1999-20. Technical report, Arizona State University, 1999.

[19] C. Su and A. Despain. Cache design trade-offs for power and performance optimization: A case study. In *Proceedings of International Symposium on Low Power Electronics and Design*, page 63, 1995.

[20] V. Tiwari, S. Malik, A. Wolfe, and T. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing Systems*, Vol. 13, No. 2, August 1996.

[21] M. C. Toburen, T. M. Conte, and M. Reilly. Instruction scheduling for low power dissipation in high performance processors. In *Proceedings of the Power Driven Micro-architecture Workshop in conjunction with the ISCA'98*, June 1998.

[22] Q. Wu, Q. Qiu, M. Pedram, and C.-S. Ding. Cycle-accurate macro-models for rt-level power analysis. *IEEE Transactions on VLSI Systems*, 6(4), page 520, December 1998.

[23] Y. Zhang, R. Y. Chen, W. Ye, and M. J. Irwin. System level interconnect power modeling. In *Proceedings of the 11th International ASIC Conference*, page 289, September 1998.