Modeling Inter-Instruction Energy Effects in a Digital Signal Processor

Ben Klass, Donald E. Thomas, Herman Schmit, David F. Nagle
Department of ECE, Carnegie Mellon University
Pittsburgh, PA 15213
benk@ece.cmu.edu, thomas@ece.cmu.edu

Abstract

This paper explores techniques for creating accurate instruction-level energy models for digital signal processors (DSP). Our initial results confirm previous work showing that inter-instruction effects can become a significant component of power consumption for many programs. To overcome limitations of previous models, we develop a straightfoward method (the NOP model) that models transitions between any two instructions. Measurements show that our method accurately models inter-instruction effects without a quadratic increase in the size of energy tables. Complex instructions are handled by treating functional units within the processor separately.

1 Introduction

Instruction-level energy models can be an effective tool for high-level software-based optimizations [LEE97][TIWA94]. The basic technique constructs a table that records each instruction's average energy. High-level power estimators use this table to quickly determine each software instruction's energy consumption, avoiding costly circuit-level simulation (e.g., Spice). Because instructions are the atomic units used by code generators, instruction-level energy models can be integrated with power-optimizing compilers more easily than simulation-based estimators. Further, instruction-level energy models allow chip manufacturers to provide fine-grained power information without having to disclose confidential design layout and implementation details—allowing software designers to quickly and accurately estimate a program's power consumption without understanding the underlying implementation details.

Unfortunately, accurate instruction-level energy models require more than simple per-instruction power estimates. Inter-instruction effects can significantly alter the power consumed by a given instruction, making it difficult to derive a single power number for each architectural instruction [LEE97]. Power tables could be expanded to include every pair of instructions. Unfortunately, building such tables can be very time consuming and requires $O(N^2)$ space (where N is at least the size of the instruction set). Grouping instructions into common classes [Lee97] can reduce the table size, but does not scale well for DSP-type architectures with their rich addressing modes and parallel instruction issue capabilities.

To overcome the problems of classification, we have developed a straightfoward method that requires only O(N) space while accurately estimating program energy. Our results, simulated with an implementation of a subset of the Motorola DSP56000 (56K), produce instruction-level power tables that predict program power within 8% percent of simulation-based estimates. Further, by attributing each instruction's power consumption to the various functional units, we preserve accuracy while overcoming the difficulty associated with modeling the 56K's rich addressing modes and parallel functions.

Section 2 describes our subset of the 56K DSP and our design methodology. Section 3 presents our approach to generating instruction-level power tables and compares our results with previous techniques. Section 4 further evaluates these models and describes potential limitations. Finally, in Section 5 we present our conclusions and outline future work.

This work was supported by the Defense Advanced Research Projects Agency under Order No. A564 and the National Science Foundation under Grant No. MIP90408457.

Presented at the Power Driven Microarchitecture Workshop in conjunction with the 25th International Symposium on Computer Architecture, June 27-July 1, Barcelona, Spain.

2 Tools and Methodology

2.1 CMU 56000 DSP

To build accurate models and to compare our results with a real design, we designed and implemented a standard-cell based subset of the Motorola DSP56000 instruction set [MOTO90]. Synopsys and Cascade's Epoch synthesized our 56K Verilog model into a standard-cell layout (see Figure 1).

We choose the 56K because instruction-level power analysis is more complex than simple RISC cores and because the 56K's functionality is representative of many power-conscious architectures. The 56K is a 24-bit, fixed-point DSP that can encode and issue one arithmetic operation and up to two "parallel" data moves in one *packed* instruction. Our 56K core implements most of the arithmetic and basic data movement instructions and accounts for most of the logic that effects power consumption.

2.2 Mynoch Power Estimator

A variety of approaches have been used to characterize the power consumption of digital systems. For physical devices, direct-measurement of current gives the most accurate measurements [TIWA94]. However, the granularity of results is limited to device-level, multi-clock cycle measurements. Accurate, fine-grained results can be obtained with Spice-based simulators, such as Star-Sim [KRIS97], but long run-times severely limit the number of cycles/events that can be simulated. Gate-level power estimators improve simulation speed [KOJI95][PURS95][XANT97], by sacrificing accuracy to achieve faster run-times.

The initial analysis of our 56K's power consumption was done using CMU's gate level analysis tool, Mynoch [PURS95]. Mynoch estimates power by counting transitions from a Verilog simulation and calculating the dynamic energy consumed for each transition using:

$$E = \frac{1}{2} \bullet C \bullet V^2$$

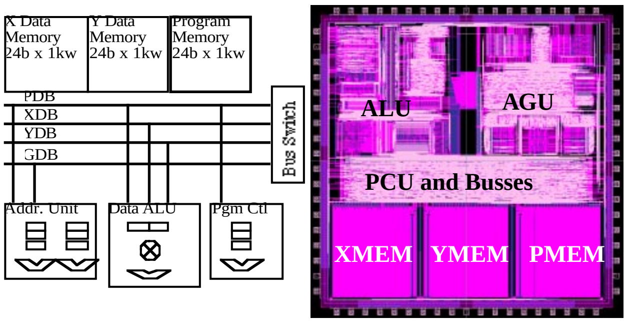

Figure 1: DSP56000 architecture and layout

Major components: 1) 3 KB of SRAM; 2) data ALU, which contains a 24x24-bit multiplier and 56-bit accumulator; 3) address generation unit (AGU), which contains three sets of eight 16-bit registers and two ALUs capable of arbitrary modulo and bit reversed arithmetic; and 4) program control unit (PCU).

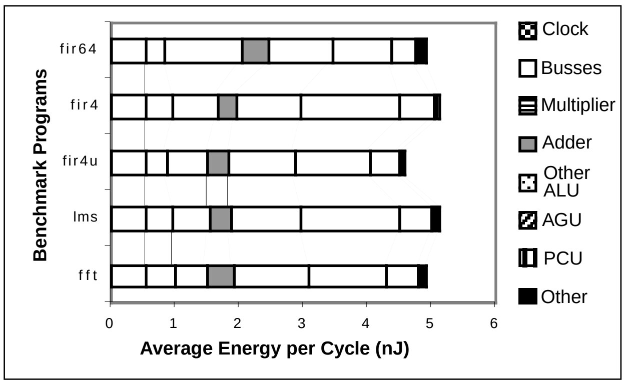

Figure 2: Energy consumption for workloads

This figure shows power consumption for the workloads. The AGU and ALU are the two largest consumers of power and almost all of the variation between programs is caused by variation in the multiplier and AGU power. The multiplier uses guard latches on the input and consumes less power with programs that have proportionally fewer multiply operations.

Mynoch runs 450 times faster than Spice simulation, allowing us to simulate thousands of cycles for each test program. Memory is not modeled with Mynoch since Epoch uses behavioral models for memory. Spice based simulations have shown memory to be approximately 20% of the total energy.

We verified the accuracy of Mynoch by comparing Mynoch's power estimates against Avanti's Star-Sim, which uses a modified Spice engine. Eighteen iterations of a 4-tap FIR filter were simulated with both Mynoch and Star-Sim. Initially, Mynoch's power estimates showed significant error in contrast to Star-Sim. To locate the source of Mynoch's error, we compared Star-Sim's per module power estimates with Mynoch's estimates. The analysis showed that Mynoch's inability to account for intra-gate capacitance within registers (i.e., D-flip flops) was the primary source of error. To correct for this error, we used Star-Sim's power estimates to build a simple linear-regression model that included the number of registers. The resulting model had a very high degree of accuracy (r^2 factor of 0.98). This model was further verified by comparing results of Mynoch augmented by the regression model vs. Star-Sim for 120 cycles of an FFT program. The error between the two methods was less than 1% for the processor, although error on functional units was higher. All of the Mynoch results reported in this study are augmented by the regression model.

2.3 Test Programs and Reference Energy

In contrast to general processors, a DSP is frequently used to compute fairly simple, data intensive programs. For the workloads of our power analysis we chose five program kernels that represent those found in typical signal processing applications (see Table 1). Three of the kernels are finite-impulse response (FIR) filters, one is a Fast Fourier Transform (FFT), and one is an adaptive filter (LMS). Gaussian white noise was used as input data.

Pgm	Description	Instr	%Instr arith	Sim cycles
fir4	4-tap FIR filter (direct form)	5	57%	1760
fir64	64-tap FIR filter (direct form)	5	96%	5,000
fir4u	4 tap FIR filter, unrolled once	12	66%	1,500
FFT	256 point FFT	24	79%	8,062
LMS	64-tap least mean squares adaptive filter	13	63%	30,000

Table 1: Description of workloads

The **Instr** column lists the number of unique instructions executed in each workload's main program loop. **%Instr arith** lists the percentage of instructions executed that are arithmetic instructions. **Sim cycles** lists the number of cycles simulated for power analysis.

Power consumption for the benchmark programs was measured using Mynoch (see Figure 2). Average energy consumed per cycle, given in nanoJoules, provides the basic unit of measure which will be used throughout the paper. This is obtained by dividing the total energy over the program execution by the number of cycles. Power consumed by the pads and memory is not included.

3 Instruction-level Models

Although circuit-level simulations provide insight into power consumption for a given program, an instruction-level model is more appropriate for code generators. Instruction-level models typically use an ***energy table*** that describes the energy cost for each instruction in an instruction-set architecture (ISA). The challenge in building such a model is balancing accuracy and energy table size. This section presents four models with different accuracy and table size trade-offs. The first model, ***base model***, produces the smallest table size, but yields poor energy-prediction accuracy across a program run. The second model, ***pair model***, has greater accuracy, but at the cost of much larger tables. The third model, ***NOP model***, provides nearly the accuracy of the pair model with much smaller tables. The fourth model, ***general model***, is similar to the NOP model but the energy table generated is independent of the program being evaluated. The general model provides reasonable accuracy and is appropriate for use with code generators.

3.1 Base Model and Estimation

Building the Base Model's Energy Table

Creating a complete and accurate energy table for the 56K DSP requires one to account for each instruction in the ISA, each instruction's different register and immediate values, and every possible packed instruction.[1] This can require a significant amount of time and space. To make the base model more tractable, we only computed the energy cost for every unique instruction[2] found in our five workloads—not for every possible tuple of {`instruction, reg/immed, instr pair`}.

1. Like many DSPs, the 56K allows for packed instructions, where an arithmetic instruction and a data-movement instruction are grouped together in one instruction.

DO #<50	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*;target instruction*
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*;target instruction*

Figure 3: Loop used to characterize MAC from FIR filters

This loop was used to calculate the base energy cost of the target instruction, in this case the MAC instruction as it occurs in the FIR filters. Two data values are read in from memory each cycle, so both multiplier inputs change. Two instances of the target instruction are needed to match the semantics of the DO instruction.

Each instruction's energy was estimated by constructing a tight-loop test program that included the target instruction and a zero-overhead branch instruction so that only the target instruction was executed in the core of the loop. Figure 3 shows the tight-loop test program used to characterize the `MAC` instruction. Each tight-loop test program was run through Mynoch to generate the base energy cost, B_{inst}. For measuring the `REP` instruction, we were forced to use a `NOP` inside of the loop because a loop of `REP` instructions is illegal.

Whenever possible, loops were made with the actual instructions used in the programs. Some instructions were modified slightly to ensure that different operands were used on each cycle. Data from the workload being characterized was used. Because the test programs are based on the actual instructions and data from our five workloads (Table 1), the results of this approach are optimistic in their accuracy. When generalizing this approach, parameters such as the destinations of parallel moves would be abstracted to reduce energy table's size.

Base Model Energy Estimates

The instruction measurements described above were used to construct a base model energy table that was then used to estimate the average energy per cycle for each of our workload programs. For example, the energy per cycle for the `fir4` program was calculated by:

CLR A X0,X:(r0)+ Y:(r4)+,Y0	*; A = 0;* *; X0 <- X(n-1); Y0 <- B(0)*
REP #<3	*; for j = 0 to 2*
MAC Y0,X0,A X:(r0)+,X0 Y:(r4)+,Y2	*; A+= X(n-j)*B(j);* *X0 <- X(n-j-1); Y0<- B(j+1)*
MACR Y0,X0,A (r0)-	*; A += X(n-3)*B(3);* *update pointer*
MOVE X:(r1)+,X0 A,Y:(r5)+	*; X0 <- X(n+1); Y:(out) <- A*

Figure 4: Main loop of `fir4`

This figure shows the code used in the fir4 main loop. The `fir64` is the same, except that the MAC operation is repeated 63 times by changing the repeat instruction to "`rep #<63`." The CLR, MAC, MACR, and MOVE instructions employ parallel moves to move data into registers immediately before the data is needed.

2. Instructions are considered different if there is any difference in their opcode, immediate values, registers, or pairings. For example, two versions of a MAC instruction, (`MAC x0,y0,a` vs. `MAC x1,y0,b`), are considered different and will have different entries in the base model's energy table.

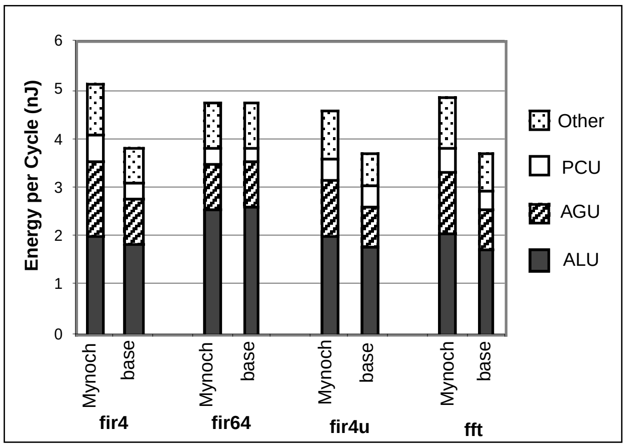

Figure 5: Mynoch vs. Base Model

Simulated energy from Mynoch and estimated energy using the base model (`base`) are given for major units. Both clock and bus power are contained in "Other." (Due to length and complexity, LMS has not been done.)

$$E_{fir4} = (B_{CLR} + B_{REP} + 3\ B_{MAC} + B_{MACR} + B_{MOVE})\ /\ 7$$

Figure 5 shows the power estimates gained from the instruction energy table. The results show a very accurate power estimate for the `fir64`. However, the estimated energy for other programs is underestimated by 17% to 25%. This error is 50% higher than the error reported in [LEE97].

This error can be understood by considering the `fir64` and `fir4` programs (see Figure 4). The accuracy in the `fir64` workload is due to `fir64`'s limited number of inter-instruction effects. The `fir64` repeats the `MAC` instruction 63 times, with no intervening instructions, in a loop of 67 instructions. This behavior is very similar to the tight-loop test programs used to derive the instruction energy table. In the `fir4`, however, energy is underestimated by 25% because inter-instruction effects are significant. The `fir4` repeats the `MAC` instruction 3 times in a loop of 7 instructions, while the remaining 4 instructions are all different. Inter-instruction effects in the remaining instructions are not represented by the base model, leading to the observed error.

For each of the workloads, most of the inaccuracy for the base model occurs in the DSP's AGU (address-generation unit) and PCU (program-control unit) functional units. For the fir4, the energy for the AGU and PCU are underestimated by more than 30%. The error in these units can be understood by considering the microarchitecture (Figure 1). The AGU must generate an address by the end of pipeline stage 2, so no registers exist between the control logic and the data path (our implementation does this to increase performance). The PCU does not latch its control points for similar reasons. The lack of registers allows glitches in the control logic to propagate into the data path, causing many false transitions as an instruction word changes. In contrast, the ALU (data ALU functional unit) latches all of its control points, so glitches are confined to the control logic, making the base model more accurate.

3.2 The Pair Model

The impact of inter-instruction effects on power estimation has been noted before and can be compensated for by assigning a per-instruction overhead that accounts for inter-instruction effects [LEE97]. This overhead, O_{instr}, is added to the base energy cost if an instruction is not the same as the previous

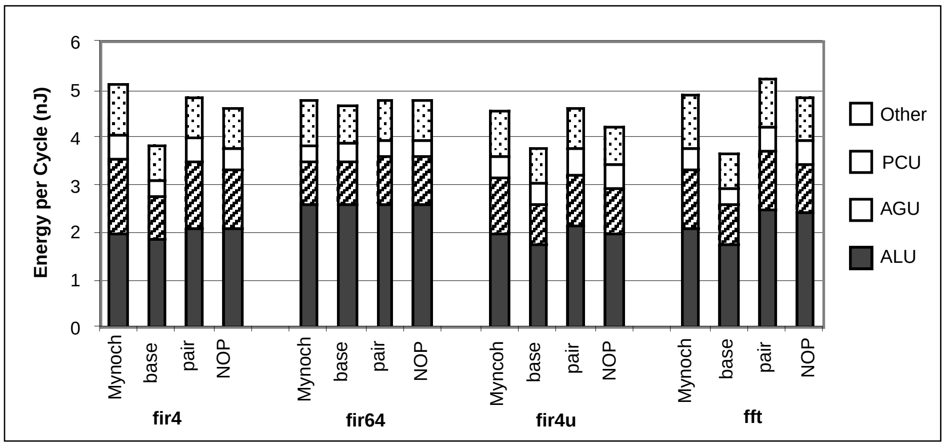

Figure 7: Different approaches to estimating inter-instruction overhead

This figure compares energy from Mynoch simulation with estimates from the base model (`base`), pair model (`pair`) and NOP model (`NOP`). The pair model measures overhead for each pair of instructions that appear in the program trace. The NOP model measures overhead for each instruction using NOP instructions

instruction. For example, the energy model for the code sequence:

MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*; target instruction*

would only use B_{MAC} for the second MAC operation, while the energy model for the code sequence:

MOVE X:(r1)+,X0 Y:(r4)+,Y0	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*; target instruction*

would use $B_{MAC} + O_{MOVE,MAC}$ because the `MAC` instruction is preceded by a different instruction, `MOVE`.

Similar to the base model, we measured O_{instr}, using tight-loop test programs. Each loop consisted of the target instruction and the instruction that preceded it in the execution trace, giving average energy per cycle for the loop E_{loop}(see Figure 6). Overhead was calculated by:

$$O_{previous,target} = E_{loop} - (B_{previous} + B_{target})/2$$

Using B_{instr} and O_{instr} where appropriate, the pair model estimated the energy for each of the workloads (Figure 7). The results, labeled as `pair`, are much more accurate than the base model, with error between 1% and 10% for all programs. However, generalizing this technique would require characterizing every possible pair of instructions, requiring a table of size $O(N^2)$, where N is the number of instructions and addressing

DO #<50	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	
MACR Y1,X1,a (r0)-	*;target instruction*

Figure 6: Loop used to find overhead: pair model

This loop was used to calculate the overhead cost of the target instruction, in this case the MACR instruction as it occurs in the FIR filters, under the pair model. The pair of instructions that appear in the workload programs was used and the overhead for this pair was assigned to the trailing instruction. Different source registers were used for the MAC and MACR instructions to ensure that both multiplier operands change, as in the FIR programs.

modes. For the 49 different instructions and addressing modes implemented in our 56K chip, a complete instruction energy table would contain 1176 entries. To reduce the table size, [LEE97] grouped instructions into classes and derived overhead costs between classes. This technique works well for simple machines, but is much more difficult to apply when dealing with the many complex addressing modes and instruction types found in a DSP such as the 56K.

3.3 The NOP Model

To avoid the difficulties of instruction grouping, we have developed a new approach that requires only one overhead cost for each instruction. This model is based on the assumption that the overhead cost for an instruction is not strongly dependent on the neighboring instruction, but does depend on whether the neighboring instructions are the same or different. This observation leads to the NOP model, which allows us to account for instruction changes without enumerating each pair of instructions.

Like the pair model, the NOP model calculates the energy for a particular operation with either B_{instr} or $B_{instr} + O_{instr}$, depending on the previous instruction. The NOP model differs in that we calculated one overhead cost for each instruction, O_{instr}, using loops which alternate the target instruction with `NOP` instructions (Figure 8). This techniques allowed us to capture the energy effects of changing instructions while keeping the size of the table to O(N). Power estimates using the NOP model are shown in Figure 7, labeled as `NOP`. The results show error between 1% and 8% on programs that previously had much larger errors with the base model. Considering that grouping instructions also decreases accuracy, this should compare favorably with any model based on instruction classes.

DO #<50	
NOP	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*;target instruction*

Figure 8: Loop used to find overhead: NOP model

This loop was used to calculate the overhead cost of the target instruction, in this case the MAC instruction as it occurs in the FIR filters, under the NOP model. A target instruction is paired with an NOP to calculate its overhead cost.

DO #<50	
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*; MACxy*
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0	*; MACxy*
DO #<50	
MOVE X:(r0)+,X0 Y:(r4)+,Y0	*; MOVEx+y+*
MOVE X:(r0)+,X0 Y:(r4)+,Y0	*; MOVEx+y+*

Figure 9: Loops used for general model

These loops were used to calculate the base cost of the MACxy and MOVEx+y+ instructions under the general model. The MACxy refers to a MAC instruction where both multiplier inputs change while MOVEx+y+ refers to two parallel moves with increment.

3.4 General Instruction Model

Having established the effectiveness of the NOP model, we generalized this approach to build tables that could be used for any program—a general instruction model. Unlike our previous models, where the instruction energy tables were built to match instructions as found in the workloads as closely as possible, the general instruction model creates a single instruction energy table that can be used across all programs. Such a table could be created by processor manufacturers and then used by a code generator to optimize power.

The general instruction model is similar to the NOP model, but extends the power analysis by accounting for packed instructions. Packed instructions present a problem when building general tables because any combination of arithmetic and parallel move is allowed. Our previous models used the actual packed instructions from each workload. When generalizing, the 23 arithmetic instructions and 24 types of parallel moves lead to 552 possible combinations, making a complete table fairly large. Fortunately, the two parts of a packed instruction are largely executed by different units within the 56K. Using this architectural knowledge, we separated the two parts of a packed instruction, building tables for the energy consumed by each of the functional units rather than the entire DSP.

The general instruction model consists of four tables, corresponding to the four significant functional units: ALU, AGU, PCU, and "Other." The first three units have been described above. "Other" refers to all remaining parts of the chip, primarily the clock and bus power. The ALU and PCU were characterized by the arithmetic portion of packed instructions only, ignoring the parallel move unless no arithmetic instruction was present. The AGU and "Other" were characterized by the parallel move portion, ignoring arithmetic instructions. Data was coarsely modeled in ways that would be visible to a code generator. Table entries for arithmetic operations were separated

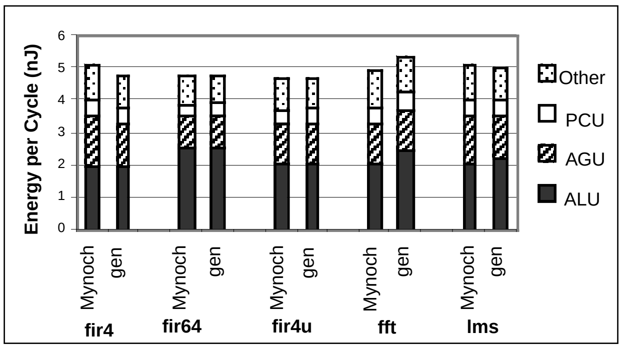

Figure 10: General Instruction Model

This figure compares Mynoch simulation energy (`Mynoch`) and the general instruction-level model (`gen`). Each component: AGU, ALU, PCU, and Other was estimated using separate tables.

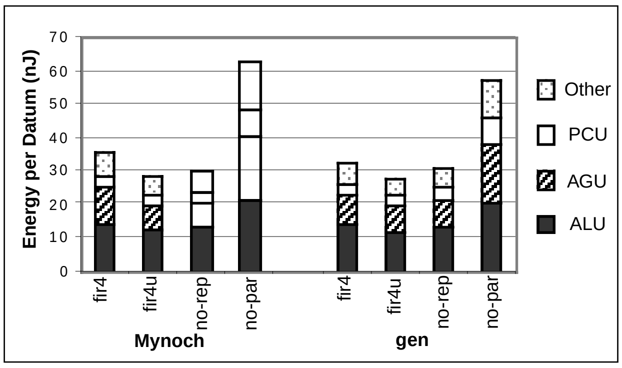

Figure 11: Implementations of a 4-tap FIR filter

This figure compares the energy to produce one datum of output for four different software implementations of the same 4-tap FIR filter. The `fir4` and `fir4u` implementations are described in Table 1. The `no-rep` implementation does not use the repeat instruction or store past inputs. The `no-par` implementation only uses one parallel move and does not use packed instructions. The number of cycles per datum for the `fir4`, `fir4u`, `no-rep`, and `no-par` programs are 7, 6, 6, and 15, respectively.

based on which operands changed value; move operations contained separate entries for the number of moves and type of update performed on address registers.

Energy costs were generated with loops similar to those described above (Figure 9). From each loop, the relevant energy costs were calculated for each unit. Uniform random data was used as input to the arithmetic unit. Under the general instruction model, the base energy cost of the instruction:

```
MAC Y0,X0,a X:(r0)+,X0 Y:(r4)+,Y0
```

was calculated by:

$$B_{\mathtt{ALU,MACxy}} + B_{\mathtt{AGU,MOVEx+y+}} + B_{\mathtt{PCU,MACxy}} + B_{\mathtt{Other,MOVEx+y+}}$$

Overhead energy costs, and whether an instruction has changed, was calculated for each unit in the same way.

Estimates based on these general instruction tables are shown in Figure 10. By making estimation automatic, we were able provide estimates for `lms` as well. Considering that program dependent information from Figure 7 has been removed, results are remarkably similar. Accuracy on all programs is within 10%.

4 Applications and Limitations

Section 3 developed an general instruction model that provides reasonable accuracy while limiting the table size. In this section we analyze this model from two perspectives. The first examines a possible use of such an energy model—evaluating the energy of code transformations within a code generator. The second looks into the importance of program data, which is not considered by the instruction-level model.

4.1 Code Transformations to Save Power

Comparing different implementations of a 4-tap FIR filter allows us to see if the energy models can recognize power savings due to code transformations (see Figure 11). While [TIWA94] looked at instruction reordering, more aggressive code transformations are used here. We implemented four versions of a 4-tap FIR filter which used the same coefficients and input data. Energy per datum processed is used as the metric to compare these programs to account for the different number of cycles required by different programs. Energy per datum is

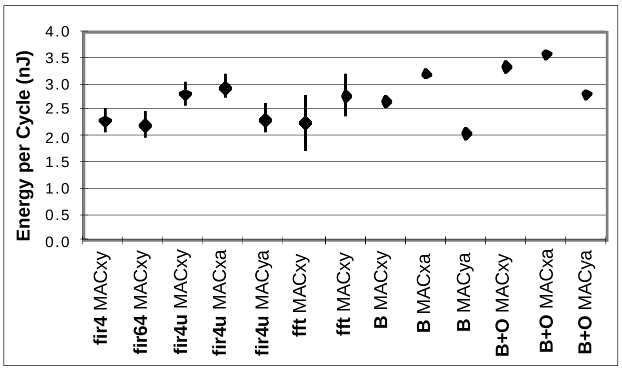

Figure 12: Energy per instruction of data ALU

The average energy for each instruction, based on Mynoch simulation of the programs, is given with standard deviation error bars to the left. The three instructions with the highest standard deviation showed a bimodal distribution. Base energy cost (B) and base plus overhead (B+O) from the general model are given to the right.

given by per-cycle energy multiplied by the number of cycles per datum.

Figure 11 shows that Mynoch power estimation predicts that loop unrolling, `fir4u`, consumes 20% less energy per datum than `fir4` while the version without packed instructions, `no-par`, consumes 75% more energy per datum. The difference in energy is due to both the energy per cycle and the number of cycles required to process one datum. The general model is able to recognize the difference in power, but does not show as dramatic an improvement for `fir4u` and `no-rep`. The lost accuracy comes from the general model underestimating the AGU's per-cycle energy for `fir4` while overestimating the AGU's per-cycle energy for `fir4u` and `no-rep`. While improved accuracy in AGU power estimation is needed, results show that a code generator using this model would choose the implementation using the least power under the general model we developed.

4.2 Data Dependent Variation

None of the models presented consider energy effects of program data. However, the power consumption of many units, such as the multiplier, can be highly data-dependent. Other research on DSP power consumption has noted the data dependent variation and analyzed the energy of individual functional units, such as the multiplier [KOJI95][LEE97]. Code transformations that keep one operand constant or reduce the number of "1" bits in a Booth-Encoded multiplier are ways that the compiler can change the data to reduce the multiplier's energy.

To gauge the importance of data, we used cycle accurate simulation to measure the power in the ALU when each MAC operation was active for the workloads. The average power consumed by each instruction, with standard deviation error bars, is shown in Figure 12 along with the energy costs for these instructions from the general model. Most instructions deviate between 8% and 12% from the mean, while two instructions deviate by 16% and 28%. The costs from the base model are generally within the one standard deviation of the workloads.

The `fft` has the largest standard deviation and has the least accurate ALU estimates under all models (c.f. Figure 7). The `fft` showed a bimodal distribution of energy in one of its MAC instructions, probably due to a large number of multiplications by zero. Improving accuracy for this problem would require moving to a model based on execution traces with sample program data. Increased accuracy of such a model would come at a cost of significantly longer simulation time, although techniques such as those proposed in [MARC96] could be used to reduce the trace length.

5 Conclusion and Future Work

We have presented several approaches for dealing with inter-instruction effects when building instruction-level energy models for a specific DSP design. The results show that using `NOP` instructions to model transitions between any two instructions give accuracy within 8% while reducing table size from almost 1200 to less than 100, and eliminates possible human error from other simplification methods. Using separate models on major units within the DSP avoided multiple table entries for different combinations of arithmetic and parallel move instructions and allowed us to build a general model. Such a model could allow code generators to recognize code transformations that reduce the energy consumed by programs.

Future work attempt to recognize when data dependent variation is likely to be important and include such variation within the model. Building models for other, non-DSP architectures would further validate the applicability of the ideas presented here.

6 Acknowledgments

Andrew Ryan, Chris Inacio, Jonathan Ying-Fai Tong and Bill Dougherty Jr. provided critical components for this study.

7 References

[**BAJW97]** R. S. Bajwa, N. Schumann, H. Kojima. Power Analysis of a 32-bit RISC Microcontroller Integrated with a 16-bit DSP. *Proceedings 1997 International Symposium on Low Power Electronics and Design,* pp. 137-142, 1997.

[KOJI95] H. Kojima, D. Gorny, K. Nitta, and K. Sasaki. Power Analysis of a Programmable DSP for Architecture/Program Optimization. *IEEE Symposium on Low Power Electronics, Digest of Tech. Papers*, pp. 26-27, Oct. 1995.

[KRIS97] Ram K. Krishnamurthy, *Mixed Swing Techniques for Low Energy/Operation Datapath Circuits*, Ph.D. Thesis, Carnegie Mellon University, December 1997.

[LEE97] M. T.-C. Lee, V. Tiwari, S. Malik, M. Fujita. Power Analysis and Minimization Techniques for Embedded DSP Software. *IEEE Trans. on VLSI Systems*, pp. 1-14, March, 1997.

[MARC96] Diana Marculescu, Radu Marculescu, Massoud Pedram. Stochastic Sequential Machine Synthesis Targeting Constrained Sequence Generation. *Proceedings of Design Automation Conference*, pp. 696-701, 1996.

[MOTO90] Motorola, Inc. *DSP56000/56001 Digital Signal Processor User's Manual*. 1990.

[PURS96] D. J. Pursley. A gate level simulator for power consumption analysis. M.S. Thesis, Carnegie Mellon University, May 1996.

[TIWA94] V. Tiwari, S. Malik, A. Wolfe. Power Analysis of Embedded Software: A First Step towards Software Power Minimization. *I994 ICCAD, Digest of Technical Papers*. pp. 384-390, 1994

[XANT97] T. Xanthopoulos, Y. Yaoi, A. Chandrakasan. Architectural Exploration Using Verilog-Based Power Estimation: A Case Study of the IDCT. *DAC 97*, pp. 415-420, 1997.