An Energy Estimation Method for Asynchronous Circuits with Application to an Asynchronous Microprocessor

Paul I. Pénzes, Alain J. Martin
California Institute of Technology
Computer Science Department
Pasadena, CA 91125, U.S.A.
penzes@async.caltech.edu, alain@async.caltech.edu

Abstract

This paper presents a simulator operating on a logical representation of an asynchronous circuit that gives energy estimates within 10% of electrical (`hspice`*) simulation. Our simulator is the first such tool in the literature specifically targeted to efficient energy estimation of QDI asynchronous circuits.*

As an application, we show how the simulator has been used to accurately estimate the energy consumption in different parts of an asynchronous MIPS R3000 microprocessor. This is the first energy breakdown of an asynchronous microprocessor in the literature.

1. Introduction

The increasing power consumption of modern VLSI systems requires better design methods to save energy. To guide the designer, energy estimation methods are needed to understand where and how the energy is consumed in a circuit. Clearly, there is a trade-off between the accuracy of the energy estimation and the level of detail at which the circuit is analyzed or simulated. The more detailed the description, the more accurate the simulation or analysis will be. But on the other hand, the more time consuming it will be. Also, the designer wants to make decisions as early as possible in the design process so as to avoid costly design backtracking.

This paper presents a simulator operating on a logical representation of the design—called "production-rules" (PRs)—that gives energy estimates within 10% of electrical (`hspice`) simulation. The PRs of a design can be either extracted from the layout or synthesized from a high-level description. Similarly, the physical parameters needed for evaluating the energy can be extracted from the layout or computed directly.

The simulator and the estimation method have been developed for the design of asynchronous circuits. Asynchronous circuits do not use clocks. Communication and synchronization among the units are implemented by handshake protocols. The particular type of asynchronous circuits this method is based on are called quasi-delay-insensitive, or QDI [1]. QDI circuit use no timing assumption on the delays of the operators and wires, with the exception of some forks, called *isochronic forks*, in which the delays on the different branches of the fork are assumed to be similar. QDI circuits are the most conservative asynchronous circuits in terms of the use of delays. But they are also the most robust to physical parameters variations because the circuit's dependence on delays is minimal. QDI circuits are interesting in the context of energy estimation because, first, they are inherently energy-efficient—absence of a global clock, locality of activity, automatic shut-off of inactive parts, absence of spurious transitions (glitches), and second, the data encoding required for QDI circuits reduces data-dependent variation in switching activity, hence making the energy estimation more accurate. Our simulator is the first such tool in the literature specifically targeted to efficient energy estimation of QDI circuits.

The paper is organized as follows. We first introduce PRs as both a programming language and a logical model for transistor networks. We present previous work in energy estimation and focus on some of the resulting difficulties such as glitches and data dependencies. We then show how energy estimation in the QDI asynchronous context is simpler and more accurate than in the synchronous context. We describe the `esim` simulator that executes a PR set. The simulator can be used for logical, timing, and energy simulation. We then describe the model used for estimating en-

ergy and the resulting accuracy—when applied to QDI circuits.

As an application, we show how the simulator has been used to estimate the energy consumption in the different parts of the MiniMIPS [2], an asynchronous MIPS R3000. This is the first known energy breakdown of an asynchronous microprocessor in the literature.

2. Production Rules as a Model for CMOS circuits

A "production-rule" (PR) is a construct of the form $G \mapsto S$, where S is a simple boolean assignment and G is a boolean expression called the guard of the PR [1]. For example, a *nand* gate with inputs x and y and output z is implemented by the two PRs: $x \wedge y \mapsto z\downarrow$ and $\neg x \vee \neg y \mapsto z\uparrow$. The PRs that set and reset the same variable, like $\neg g1 \mapsto z\uparrow$, $g2 \mapsto z\downarrow$ are implemented as one operator ($g1$ corresponds to the pull-up, while $g2$ to the pull-down circuitry of node z).

We use a PR set as the logical representation of QDI circuits. PRs closely correspond to the intended CMOS implementation of the circuit. The PR set of a given QDI circuit could be either synthesized from a high level representation or could be directly generated by the designer.

3. Energy Estimation

3.1. Previous work

The estimation of energy consumption using a simulator is complicated by the *pattern-dependence* problem. In a synchronous circuit, the switching activity of a node depends on the current data being processed and also on the initial state of that node. Furthermore, during a clock cycle, a node might switch several times before settling to the steady state value. This extra switching activity (glitches) contributes to the total energy dissipation. The extra energy added by glitches is typically 20%, but could be up to 70% of the total energy (in combinational adders) [3].

In general, the approach taken to alleviate the *pattern-dependence* problem is to assume some simplifying conditions about the input space and to attach user specified probabilities of transition [4] to the input nodes; probabilities that get propagated to all nodes of the circuit. In these methods two different assumptions might be made: a *spatial independence*, in which case input signals on the same clock cycle are assumed to be non-correlated and a *temporal independence*, in which case the value of the same signal in two consecutive clock cycles is assumed independent. Once the transition probabilities are computed using probabilistic [5] or statistical [6] techniques, the energy can be computed as $E_{av} = \frac{1}{2}V_{dd}^2 \sum_{i=1}^{m} C_i P(x_i)$, where C_i is the total capacitance at node x_i, $P(x_i)$ is the transition probability at node x_i and m is the total number of circuit nodes that are outputs of logic gates (the power can be computed as $P_{av} = \frac{E_{av}}{T}$, where T is the clock period). However, this estimation is a lower bound on the consumed energy, since there will be at most one transition per clock cycle for a given node; thus, glitching energy is not included. Some methods [7] use *transition density* instead of *transition probability* of a node to capture the glitching activity of the circuit.

All these methods are still pattern-dependent to some extent, except that now the user must supply information (in terms of probabilities) about a typical behavior at the circuit input. Furthermore, these techniques use simplified delay models which make the estimation of glitching energy (in particular) more prone to error.

3.2. Energy estimation in the asynchronous context

Our method of energy estimation is also based on simulation [8]; however, as opposed to the synchronous design case, our approach practically is *not* input *pattern-dependent* due to the QDI circuit design methodology. In the following, we describe why we believe our energy simulation method of QDI circuits yields superior accuracy and performance when compared to simulators of synchronous circuits of the same complexity.

CMOS circuits have three main sources of energy dissipation: dynamic currents (due to charge and discharge of capacitors), short-circuit currents and leakage currents. Some existing simulation tools include the energy dissipation due to short-circuit currents. We have found that—for the types of QDI circuits we are interested in—they are not an important source of energy dissipation. Leakage currents are due to the subtreshold behavior of the CMOS transistor. At the current state of technology they contribute a small fraction to the total energy consumption. For these reasons, for the purpose of this work we consider the contribution of both short-circuit currents and leakage currents to be null.

As in many other energy estimation tools, we focus our attention to the dynamic energy consumption

$$E_{dynamic} = \frac{1}{2}V_{dd}^2 \sum_{i=1}^{m} C_i n_i \qquad (1)$$

where C_i is the total capacitance at node i, n_i is the *transition count* of node i during the measured period of time and m is the total number of circuit nodes. The main difficulty with this formula for synchronous systems comes from estimating n_i. In the following, we will show how our method for estimating energy in QDI circuits deals with this difficulty.

3.3. Energy estimation error due to glitching

As mentioned earlier, energy simulation of synchronous circuits is complicated by spurious transitions (glitches). The main reason is that glitches are highly delay dependent; any attempt to properly estimate the energy due to them has to include an accurate timing model. For QDI circuits, the design methodology explicitly avoids glitches by enforcing the monotonicity of signal transitions [1]. As a consequence, the energy consumed due to glitches is null and there is no need to complicate the energy simulator with a timing model.

3.4. Energy estimation error due to *spatial dependence* (input correlation)

In a QDI circuit, data is encoded as dual-rail or 1-of-N signals. In general, we make no assumption about which rails of a data channel (as opposed to control channel) would be exercised more often. As a result, the corresponding CMOS circuitry is symmetric—in terms of transistor sizes—for each data rail within a channel. Furthermore, each activated node transitions exactly twice (not all signals become active, since some of them are inherently mutually exclusive). For these reasons, these types of circuits consume the same amount of energy, independently on the data being processed. Most of the QDI circuits we are interested in are of this type.

One important exception are adders. Consider a carry-lookahead adder. In such a circuit, the propagate rails have more load than the kill and generate rails, since—for uniform input distribution—a transition on the propagate rail is more likely. This effect generates some input pattern dependence on the consumed energy. However, when put in the realistic context of the QDI design (input/output completions, data/control acknowledges, internal enable) this dependence is very weak. In case of an `add` instruction of the MiniMIPS [2], the difference between the worst case (all propagates) and best case (no propagates) energy dissipation is 1.53% of the average case; correspondingly, for an `addi` instruction the same relative difference is 1.49%.

For control channels, the dependence could be stronger. This is expected, since different values of the control might activate different parts of the circuit. Consider an adder again. In case of a regular add instruction, the two inputs used are coming from the register file; in the case of an add immediate instruction one input comes from the register file, and the other from the immediate bus, and (depending on the implementation) this could take a different amount of energy. For these reasons, the energy corresponding to each distinct control signal of a given circuit block has to be estimated individually.

3.5. Energy estimation error due to *temporal dependence* (cycle-to-cycle dependency)

In a synchronous circuit, the switching of a signal depends not only on the data value on the current cycle, but also on the previous value of that node, causing a cycle-to-cycle dependency.

For a QDI circuit, each node that will be active during an operation will go through exactly two transitions (a charging and a discharging transition). By the end of the cycle, each node will be in the same state as it was at the beginning of the cycle—independently of the performed operation. This property eliminates cycle-to-cycle dependencies altogether.

There is one important exception, namely the case of writing state variables implemented as flip-flops. If the state bit has the same value as the value to be written to it, the bit will not flip. On the other hand, if the two values are different, the bit will transition and as a result energy will be consumed. Thus, depending on the initial condition of the state variable, an energy consuming event might or might not occur, causing a temporal dependence.

One way to deal with this problem is to simulate the system for the worst (all bits changing) and best (no bits changing) case and take as energy estimate the average of the two numbers. In practice, when such a flip-flop is integrated in its actual QDI environment, its energy contribution becomes small compared to the total energy dissipation. For example, the relative difference between the worst case and best case write to a register in the register file of the MiniMIPS is 6.44% of the average case. Thus, if it is assumed that on average 50% of the state bits change the worst case error will be $\pm 3.22\%$.

All in all, the energy variability due to temporal or spatial dependence of input data is localized and relatively small; thus, in general it can be ignored.

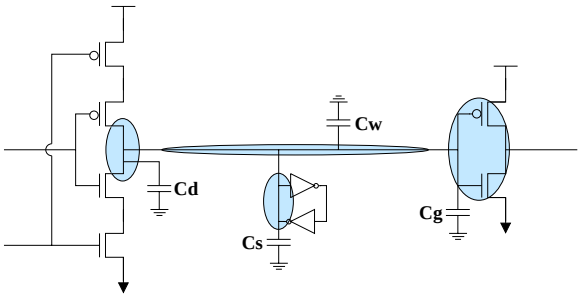

Figure 1. `esim` **capacitance model**

4. The `esim` simulator

4.1. Energy model used by `esim`

The `esim` energy simulator estimates the dynamic energy consumption of a QDI circuit using Equation 1.

The general operation of the simulator is as follows. Once the simulator is launched on a closed PR set, all nodes are initialized and assigned an energy `weight`. At any time, the ready queue contains all PRs that are ready to be fired, i.e. whose guards are true. A step of the simulator consists of taking one PR out of the ready queue and firing it, i.e. changing the value of the variable the PR operates on. This firing may cause other PRs to become enabled, in which case their PRs are added to the ready queue. Which PR is removed from the ready queue on a firing depends on the timing settings used (timed or random) to initialize the simulator. If the PR just taken off the ready queue is in the scope of the monitored set, its assigned energy weight is added to a counter that keeps track of the total energy consumed. Once the simulation is terminated, the energy counter contains the total energy consumed by the monitored region of the circuit.

There are two unknowns in Equation (1): the node capacitance C_i and the transition count n_i. The node capacitance C_i is determined statically—as will be explain later—when the simulator is launched. The transition count n_i is being taken care of automatically by the simulator: for each actual transition, the corresponding weight is added to an energy counter. Since our simulator handles transitions explicitly, each and every energy consuming signal switching is recorded.

As shown in Figure 1, each node in a PR set could have the following capacitive components: source/drain diffusion capacitance due to the PR's pull-ups and pull-downs (C_d), wiring capacitance (C_w), capacitance due to a staticizer ("keeper") if the node is state-holding (C_s) and gate capacitance due to the transistors the node is driving (C_g). There are other capacitive nodes present in the actual circuit that are not represented at PR level. They are mainly internal nodes of transistor chains. We choose to ignore the energy dissipated to charge and discharge these internal nodes.

Depending on the level of detail of the simulated design, each capacitive component may or may not be available. If only the unsized PRs are available, all capacitive terms are ignored except C_g that is estimated to be proportional to the actual (i.e. considering gate sharing) fanout of the corresponding node.

If a sized but unwired PR set is available for simulation, all capacitive components except C_w can be computed with good accuracy. The dynamic energy consumed by a node i can be written as $E = E_{C_g} + E_{C_d} + E_{C_s} = K * (\sum_{transistor\ width})$ where the technology-dependent constant K can be computed directly or calibrated using `hspice`. When `esim` is first run on a PR set, the value of $(\sum transistor\ width)$ is computed for each node and assigned to its corresponding `weight` variable in the data structure.

If a wired-up layout exists, the wire information can be added as explicit PRs into the original PR set. Given the capacitance of the wires (computed by the layout extractor) one has to transform these values into *transistor width* units. Once this is done, wires are treated by `esim` exactly as any other PR pair.

There are several energy dissipating sources `esim` is ignoring. Possibly the most important one is the energy consumed due to leakage currents. Some researchers have reported the increased importance of this energy dissipation source for submicron technologies. Energy consumption due to short-circuit currents is ignored as well. Our evaluation shows that in typical QDI circuits the energy dissipated due to short-circuit currents is no more than 1.5% of the total energy consumption. There are two elements relating to dynamic energy consumption that are ignored. First, energy due to charge-sharing is not captured, and second, the energy spent on charging and discharging internal nodes of transistor stacks are not accounted for. Our `hspice` simulations show that these dynamic energy dissipation sources can be safely ignored for most of the circuits we are interested in (in current technology).

4.2. Timing model used by `esim`

`esim` does not need a timing model for energy estimation. For this reason, the simulator does not explicitly compute the timing information of the PRs. However, if timing estimate is needed, the timing of each PR can be passed to `esim` using *after delay* rules; for example: *after* 123 $a \wedge\ b \mapsto \_c \downarrow$ i.e., $\_c$ will fire

123 time units after a and b become true. If no *after rules* are present, either random or unit timing is assumed depending on the simulation settings. The after rules can be assigned by a preprocessing phase using either a simple τ-model estimation or using an `hspice` characterization table. The same after rules can be used to assign RC delays to wires. There is extensive research done in accurately predicting the timing behavior of transistor netlists; however, for many of our experiments, it is sufficient to use a simple timing model based on counting transitions on the critical path, i.e. using `esim` with unit timings.

4.3. Accuracy and speed of `esim`

We have evaluated the accuracy of `esim` in estimating energy on several test circuits ranging from 192 to 2584 transistors, by comparing the results to the `hspice` results recorded on the same circuits. We have done our evaluation in the HP 0.6μ CMOS technology. We have evaluated different levels of detail in implementing the node capacitance computation [9]. The conclusion was that if C_g, C_d and C_s were included, the resulting error was in the range of 1.45% to 7.25%. From these results we have concluded that if the terms C_g, C_d and C_s are available to the simulation, the error in energy estimate between `hspice` and `esim` is less than 10%. The salient assumption of this claim is that wire capacitance (C_w) is an order of magnitude smaller than gate capacitance for the considered design. This assumption is true for most of the circuits sized for high speed, such as the MiniMIPS microprocessor. However, if this assumption is violated, wires have to be considered explicitly through PRs.

The discrepancy is due to all other sources of energy consumption that were ignored, mainly the wire capacitance C_w—not included in the simulated PR set.

The speed of `esim` is outstanding: more than three orders of magnitude faster than `hspice`. One instruction of the MiniMIPS (more than 2 million nodes) can be simulated in less than 1 second on a Pentium III Xeon 550MHz.

5. Energy Estimation of the MiniMIPS

In this section, we apply the `esim` simulator to obtain a breakdown of the energy consumption of the MiniMIPS. We believe that all the figures obtained by these simulations are no more than 10% smaller than the actual energy consumption figures of the fabricated chip. The MiniMIPS processor was fabricated in an HP 0.6μ CMOS process; the quoted numbers refer to this technology. Given the technology independent PR representation of QDI circuits, it is straight-forward—as long as the assumptions about leakage and short-circuit currents hold—to extend the simulation tool to other technologies by simply recomputing the constant K for the target technology.

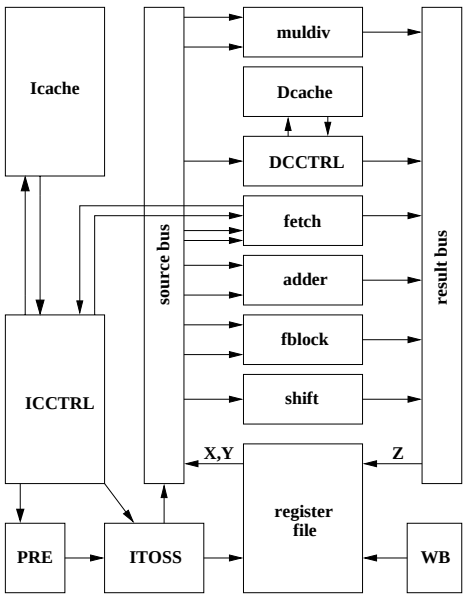

Figure 2. Block diagram of the MiniMIPS

We have divided the processor into 15 functional units, shown in Figure 2. These units are: Icache (4K pipelined instruction cache core), ICCTRL (instruction cache control, tag comparison), PRE (instruction decode), ITOSS (mispredicted instruction filter used for branch prediction), fetch (pc computation), register file (32 32-bit registers with locking, one-entry register bypass), source bus (instruction bus, source operand bus, immediate bus), adder, shifter, fblock (logical-function block), muldiv (array multiplier, iterative divider), DCCTRL (data cache control, exception pc queue, CP0 registers), Dcache (4K pipelined data cache core), result bus, WB (writeback unit).

In the following subsection, we describe the behavior and the associated energy breakdown of some generic groups of instructions.

5.1. `nop`

On a `nop`, the instruction is looked up in the instruction cache (Icache) based on the pc received from the fetch. Once the instruction cache control (ICCTRL) determines that the cache entry is valid, the instruction is sent to the decode (PRE). If the instruction

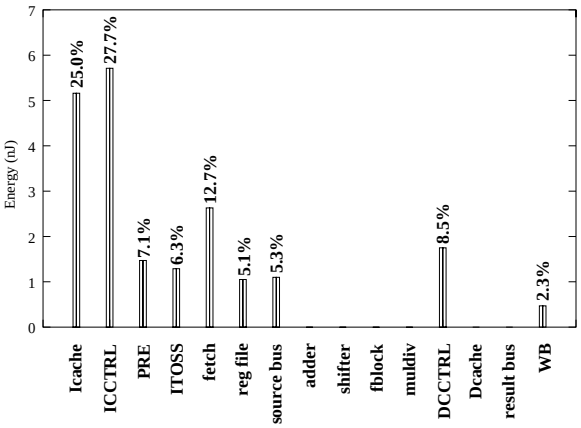

Figure 3. Energy of a `nop` instruction

is not a mispredicted branch, it passes through a filter (ITOSS) and is allowed execution. In parallel, the fetch is notified by the decode about the nature of the instruction such that it can compute the next pc. The register file is also notified about the `nop` instruction. In order to properly restore processor state in case of an exception or interrupt, the writeback unit (WB) is notified and the program counter is saved through the epc queue which is part of the data memory system (DCCTRL).

The total energy consumed by a `nop` executing from cache is 20.63nJ—the corresponding energy breakdown is shown in Figure 3 (an instruction cache miss adds 23.28nJ to the currently executing instruction). About half of the energy goes into bringing the instruction from the cache, and about one quarter of it goes into computing the next pc, and decoding and filtering the instruction. The epc queue ($\subset$ DCCTRL) consumes a relatively high percentage of the total energy due to the high amount of buffering needed to perform its function. The writeback unit consumes relatively little energy, not only for `nop`'s but for any other instruction. As previously mentioned, units not involved in the execution of the current instruction consume no energy; for a `nop` this is the case for 6 out of the 15 blocks of the processor. Even the units consuming energy might not consume their peak energy—an example being the register file which consumes only 1.10nJ while its peak energy consumption is 9.15nJ.

5.2. Arithmetic instructions

There are 4 different execution units in the MiniMIPS: the adder-subtracter, the logical-function block, the shifter and the multiplier-divider. Each instruction executed by one of these units is performed conceptually as follows. Once the instruction is launched for execution, the decode requests the register operands from the register file. Upon arrival of the operands, the execution unit performs the requested operation, and delivers the result to the register file on the result bus (except for the muldiv which writes the result to its local Hi/Lo registers). If the execution unit could potentially raise an exception (adder) the exception information is sent to the writeback. The operation result is passed to the register file bypass and eventually written back to the register file core once the writeback allows it.

In order to determine the energy cost of a given arithmetic instruction, it is necessary to know where the operands are coming from within the register file: bypass, core or both. As a results, depending on the context, the same instruction could consume different amounts of energy. We have determined the energy cost of the following individual operations:

operation	bypass	core	total
read from bypass	0.54nJ	—	0.54nJ
read from core	0.74nJ	2.11nJ	2.85nJ
write	1.53nJ	0.82nJ	2.35nJ
reg. file control	—	—	1.10nJ

Based on these individual results, a register file operation could take anywhere from 1.64nJ (reading one operand from the bypass) to 9.15nJ (reading two operands from the core and writing back one result).

The arithmetic instructions—based on their energy consumption—can be divided into (8) groups, group A: `add`, `addu`, `sub`, `subu`, `slt`, `sltu`, group B: `addi`, `addiu`, `slti`, `sltiu`, group C: `sll`, `sra`, `srl`, group D: `sllv`, `srav`, `srlv`, group E: `and`, `nor`, `or`, `xor`, group F: `andi`, `ori`, `xori`, group G: `mult`, `multu` and finally group H: `div`, `divu`. The breakdown of the energy consumption of each group of instruction is shown in Figure 4; the energy consumed by the muldiv unit was reduced by 100$\times$ to fit the graph.

While the energy spent in manipulating the instruction (cache, decode, fetch, epc queue, WB) is the same as for a `nop`, the cost of executing the instruction is different. For example, the total energy consumed by an `add` reading both operands from the register core is 34.33nJ, reading one operand from the register core and the other from the bypass is 32.02nJ and finally, reading both operands from the bypass is 30.21nJ. The energy consumed by the adder is less that 10% of the total energy of a group A or B instruction. The shifter consumes about 2 times, and the fblock about 1/3 of the adder's energy. The multiply part of the muldiv consumes about 10 times more energy than the adder, while the divider part of the muldiv consumes about

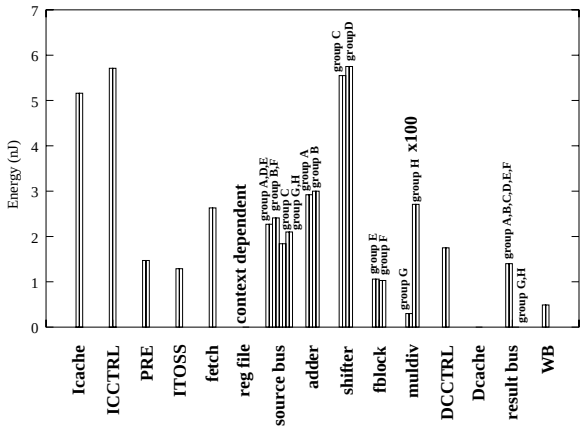

Figure 4. Energy of an arithmetic instruction

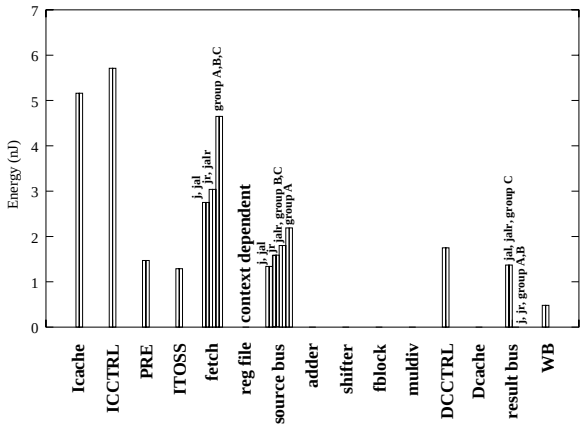

Figure 5. Energy of control-flow instruction

90 times more than the adder.

5.3. Control-flow instructions

There are two kinds of control-flow instructions in the MiniMIPS: unconditional jumps and conditional branches. An unconditional jump computes the pc of the next instruction either by concatenating the high order 4-bits of the delay slot's address, 26-bits of immediate and 2 zero bits (`j, jal`) or directly from a register (`jr, jalr`).

A conditional branch has the branch target address computed from the sum of the delay slot's address and a 16-bit offset, shifted left two bits and sign-extended to 32-bits. Depending on the type of the conditional branch, the contents of two registers or one register and zero are compared. In the MiniMIPS, each conditional branch is predicted taken if it is a backward branch or not taken if it is a forward branch. There are different energy costs for a correctly predicted branch and a mispredicted branch. Conditional branches—based on their energy consumption—can be divided into 3 groups, group A: `beq`, `bne`, group B: `bgez`, `bgtz`, `blez`, `bltz`, and group C: `bgezal`, `bltzal`. Figure 5 shows the energy breakdown for unconditional jumps and correctly predicted conditional branches. Conditional branches are comparable in energy consumption to less costly (E, F group) arithmetic instructions.

In case of a mispredicted branch, the instruction is canceled in the ITOSS unit and the fetch resends the—now correct—pc to the instruction cache. In this way, only the instruction fetching consumes energy (including the epc queue), while the rest of the processor does not. Once the correct instruction is brought back from the cache, the execution of the branch proceeds as in the correctly predicted case, except now the fetch consumes 6.04nJ for all conditional branches. The total energy cost of branch misprediction is 21.49nJ, just a bit more than that of a `nop`.

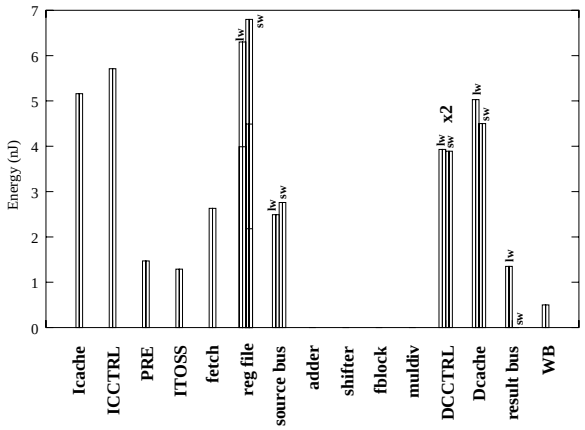

Figure 6. Energy of load/store instruction

5.4. Load/store instructions

Another group of instructions are the load/store's. The MiniMIPS implements only word operations to/from memory: load word (`lw`) and store word (`sw`). Their energy breakdown is shown in Figure 6. A data cache miss adds 30.86nJ to the current load instruction.

5.5. `esim` agreement with lab data

Thanks to Mika Nyström, we have collected lab data of the total energy consumed by the fabricated chip running certain instructions. The next table compares these results with the results estimated using `esim`.

instr	lab	`esim`	error
`nop`	22.71nJ	20.63nJ	9.16%
`addiu r2, r1, 0`	34.06nJ	31.57nJ	7.31%
`sll r1, r0, 0`	34.85nJ	33.58nJ	3.64%
`or r2, r1, r1`	33.78nJ	32.32nJ	4.32%
`ori r2, r1, 0`	31.04nJ	29.60nJ	4.64%
`mult r1, r1`	61.19nJ	56.77nJ	7.22%
`lw r2, 0(r1)`	44.87nJ	41.51nJ	7.48%

This comparison shows that the error between lab data and `esim` is as expected—given the accuracy of `hspice`—within 10%.

The application of `esim` to the PR set of the MiniMIPS gave us important insights in the specifics of energy consumption in an asynchronous processor. This knowledge will be useful for the contemplated redesign of the MiniMIPS.

6. Conclusion

We have presented a method and a simulator for the accurate energy estimation of QDI circuits.

The method is based on the "production-rule" (PR) notation, which gives an accurate logical description of the circuit. The PR set of a system can be either synthesized from a high-level description or extracted from layout. The `esim` simulator is both simple and efficient and can be parallelized easily. It can be used for logical, timing, energy, or combined Et^2 [2] simulation.

The models for capacitance and delays can be as simple or as sophisticated as necessary, and can be either estimated from a high-level description or extracted from layout. The accuracy of the physical modeling of switching activity is greatly improved by the absence of glitches and other beneficial properties of QDI circuits. Of course, estimating interconnect capacitances accurately remains a problem.

The method has been successfully applied to the energy estimation of a complete asynchronous 32-bit MIPS microprocessor, demonstrating the efficiency of the method for the simulation of large systems. The results also shed light on the energy budget of such a processor. In particular, it shows that 90% of the energy is consumed in communication and only 10% in actual execution of instructions.

These results and the accurate energy breakdown will be useful for the contemplated redesign of the asynchronous MIPS, which will be optimized for Et^2. We expect greatly improved energy consumption compared to the present design.

Acknowledgments

We wish to thank the members of the Asynchronous VLSI Group at Caltech for many stimulating discussions: Mika Nyström, Catherine Wong, and Karl Papadantonakis, and José Tierno from IBM, TJ Watson Research Center.

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency and monitored by the Air Force under contract F29601-00-K-0184.

References

[1] Alain J. Martin. Synthesis of Asynchronous VLSI Circuits. *Formal Methods for VLSI Design*, ed. J. Staunstrup, North-Holland, 1990.

[2] Alain J. Martin, et al. The Design of an Asynchronous MIPS R3000 Microprocessor. *Proceedings of the 17th Conf. on Adv. Research in VLSI*, IEEE Computer Society Press, 164-181, 1997.

[3] A.Shen, A.Ghosh, S.Devadas, K.Keutzer, "On average power dissipation and random pattern testability of CMOS combinational logic networks," IEEE/ACM Int. Conf. Computer-Aided Design, Santa Clara, CA, Nov. 8-12, 1992, pp 402-407

[4] M.A.Cirit, "Estimating dynamic power consumption of CMOS circuits," IEEE Int. Conf. Computer-Aided Design, pp. 534-537, 1987

[5] C.Y.Tsui,M.Pedran,A.M.Despain, "Efficient estimation of dynamic power consumption under a real delay model," IEEE Int. Conf. CAD, Santa Clara, CA, Nov 7-11, 1993, pp. 224-228

[6] R.Burch, F.Najm, P.Yang, T.Trick, "McPower: A Monte Carlo approach to power estimation," IEEE/ACM Int. Conf. Computer-Aided Design, Santa Clara, CA, Nov 8-12, 1992, pp. 90-97

[7] F.Najm, "Transition density: a new measure of activity in digital circuits," IEEE Trans. CAD, vol. 12, No. 2, pp. 310-323, Feb. 1993

[8] Matt Hanna, Eitan Grinspun, "A Production Rule Simulation" Caltech Computer Science Technical Report, 2000

[9] Paul I. Pénzes. *Energy-delay Efficiency of Asynchronous Circuits*, Ph.D. Thesis (in preparation), California Institute of Technology, 2002.