Power Efficient Comparators for Long Arguments in Superscalar Processors

Dmitry Ponomarev Gurhan Kucuk Oguz Ergin Kanad Ghose

Department of Computer Science
State University of New York, Binghamton, NY 13902–6000
e–mail:{dima, gurhan, oguz, ghose}@cs.binghamton.edu
http://www.cs.binghamton.edu/~lowpower

ABSTRACT

Traditional pulldown comparators that are used to implement associative addressing logic in superscalar microprocessors dissipate energy on a mismatch in any bit position in the comparands. As mismatches occur much more frequently than matches in many situations, such circuits are extremely energy–inefficient. In recognition of this inefficiency, a series of dissipate–on–match comparator designs have been proposed to address the power considerations. These designs, however, are limited to at most 8–bit long arguments.

In this paper, we examine the designs of energy–efficient comparators capable of comparing arguments as long as 32 bits in size. Such long comparands are routinely used in the load–store queues, caches, BTBs and TLBs. We use the actual layout data and the realistic bit patterns of the comparands (obtained from the simulated execution of SPEC 2000 benchmarks) to show the energy impact from the use of the new comparators. In general, a non–trivial combination of traditional and dissipate–on–match 8–bit comparator blocks represents the most energy–efficient and fastest solution. As an example of this general approach, we show how fast and energy–efficient comparators can be designed for comparing addresses within the load–store queue of a superscalar processor.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids – *layout, simulation*

General Terms

Design, Measurement

Keywords

Low–power comparators, superscalar datapath

ISLPED'03, August 25–27, 2003, Seoul, Korea

1. INTRODUCTION

Today's superscalar microprocessors make extensive use of associative matching logic and comparators to support out–of–order execution and virtual memory mechanisms. Comparators, either explicit or embedded into content–addressable logic, are in a pervasive use within the issue queues, load–store queues, translation lookaside buffers (TLBs), branch target buffers (BTBs), caches, reorder buffers and CAM–based register alias tables.

Specifically, the long comparators (comparing upwards of 8 bits) are in wide use in today's high–performance processor designs. They are used, either by themselves or embedded into a content–addressable logic, in at least the following key datapath components:

1) Within the translation lookaside buffer (TLB) to quickly translate virtual page numbers to physical page numbers in parallel with the access of a physical address–tagged cache.

2) Within the load–store queues (LSQ) to match the addresses of the pending load instructions against the addresses of previously dispatched store instructions to enable the loads to bypass previously dispatches stores, if the address of the load does not match the addresses of all such stores.

3) Within banks of instruction and data caches for some embedded CPUs like the Strong ARM SA 1100 (which uses a fully–associative 256–entry cache bank).

4) Within the branch target buffer (BTB) to obtain the target of a taken branch and continue fetching the instructions along the predicted path without interruption.

The traditional equality comparator circuit used for implementing associative logic in modern datapaths (or, for that matter, any digital comparison) is shown in Figure 1 [4]. These so called pull–down comparators pull down a precharged output, *out*, on a mismatch in any bit position when the evaluation signal (eval) goes high. The precharged output remains high on a match. Energy is thus dissipated on a mismatch in the compared arguments (comparands). No dynamic energy dissipation occurs on a full match; the only possible energy dissipation in this case is attributed to leakage.

Content–addressable memories (CAMs) also employ traditional dissipate–on–mismatch comparators that are embedded into the bitcells. Recent work have addressed the problem of minimizing energy dissipation in CAMs [6, 7]. In [7], the CAM words are effectively sub–banked and searches proceed on a subbank by subbank order. If a word–slice within a subbank does not match the corresponding bits in the search key, comparisons in slices of the same word in the following subbanks are disabled, thereby saving energy in extraneous comparisons. The approach of [6] extends this technique further. The common feature in both approaches is still the reliance on dissipate–on–mismatch comparators.

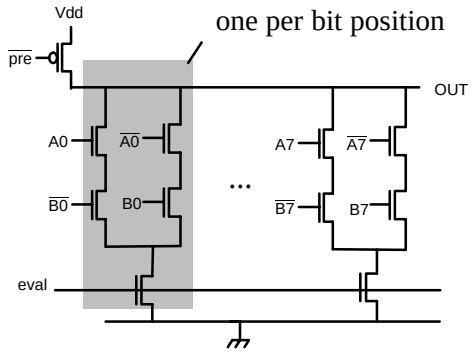


Figure 1. Traditional pull–down comparator

In many situations, where the circuit of Figure 1 is used in modern datapath (in the applications alluded to earlier) mismatches occur with a much higher frequency compared to full matches. Consequently, significant energy savings can be realized if comparators and associative logic can be designed to dissipate energy only on a full match and little or no energy on a mismatch. In recognition of this inefficiency, first noticed in [2], a series of such dissipate–on–match comparator (DMC) designs have been recently introduced [1], [5].

A limitation of the traditional comparator shown in Figures 1 is that in practice, the worst–case delays go up with the number of bits. This is because of the increase in the switched capacitance of the precharged node with the number of bits compared. Wider pulldown paths are needed to counter this in an effort to maintain the discharge time constant. Likewise, a wider p–device is also needed to maintain the precharging time constant. In reality, there are layout area considerations and other practical limitations (such as leakage energy considerations) that constrain the width of the p– and n–devices. The inevitable impact of these constraints is an increase in the overall worst–case delay of the traditional comparator with the number of bits compared.

In many applications within a modern superscalar processor (as well as in some scalar datapaths) much longer arguments have to be compared. Examples of such applications include the comparison of full effective addresses within the load–store queues and fully–associative TLBs and the comparison of long parts of a full address within caches and set–associative TLBs and BTBs. Somewhat different approach is needed to design these long comparators in an energy–efficient manner with acceptable delays. In some designs, compromises may have been made to avoid long comparison delays. For example, a few address bits may be compared when a LOAD's effective address is compared against the effective address of all pending STOREs ahead of the LOAD in the load–store queue. If any match occurs, a dependency is assumed, even though there may not be any had the full addresses been compared! When such a match is detected in comparing only a few bits of the addresses, the LOAD is not allowed to bypass the pending STOREs and overall performance may be limited in the case when there is no match had all of the address bits been compared.

The design of fast, energy efficient comparators for comparing long arguments in applications such as the above is exactly the subject of this paper.

The rest of the paper is organized as follows. In Section 2, we discuss a non–traditional comparator design that is used as a component in some long comparator circuits. Section 3 describes several design alternatives for the long comparators. These designs are based on the use of several traditional comparators, several non–traditional comparator designs (as introduced in [1] or [5]) or a combination of the traditional and non–traditional designs. The choice of a specific design for a long comparator is actually dependent on the distribution of the bit patterns that are compared, which are usually obtained through microarchitectural–level simulations. Our simulation methodology is presented in Section 4. In Section 5, to illustrate how a specific long comparator design has to be chosen, we evaluate the usage of long comparators for comparing the addresses of the load and store instruction in the load–store queue. Our conclusions are presented in Section 6.

2. NON–TRADITIONAL COMPARATORS

One of the artifacts of a superscalar processor, where the mismatches in the comparands are much more frequent than the full matches is the issue queue [1,5]. The issue queue thus demands the use of non–traditional fast comparators that dissipate energy only (or predominantly) on a full match and little or no energy on a partial match [1, 2, 5]. Such designs are significantly more energy–efficient than the traditional designs. In the case of the issue queue, only a few bits – no more than 8 – need to be compared.

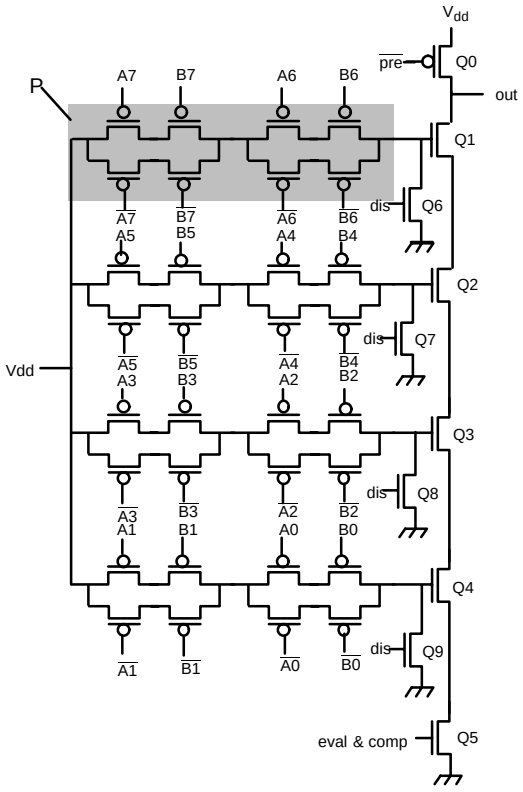
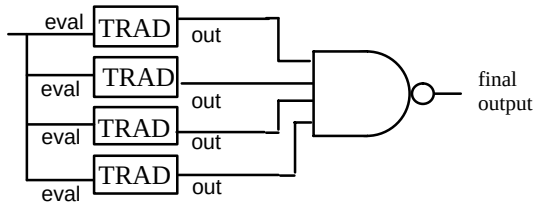


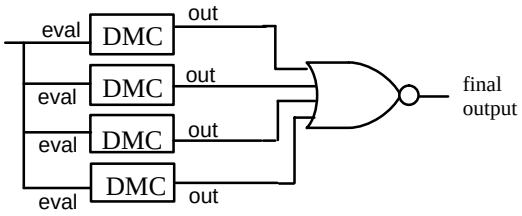
Figure 2. A 8–bit Dissipate–on–Match Comparator

A non–traditional comparator design, as introduced in [1], which is superior to the traditional design, both in terms of energy and delay, is shown in Figure 2. This circuit compares two 8–bit comparands, A7A6...A0 and B7B6...B0. The series pulldown structure consisting of the devices Q1, Q2, Q3 and Q4 conducts when all 8 bits of the comparands are equal. The output of this comparator, precharged to Vdd by Q0 is discharged when all bits of the comparands are equal and when the evaluate device, Q5, is on. The n–transistors Q6, Q7, Q8 and Q9 discharge any accumulated charges from the gates of Q1, Q2, Q3 and Q4 respectively during the precharge phase and prevent the series structure from discharging the output due to the presence of accumulated charges at the gates of the series device from past



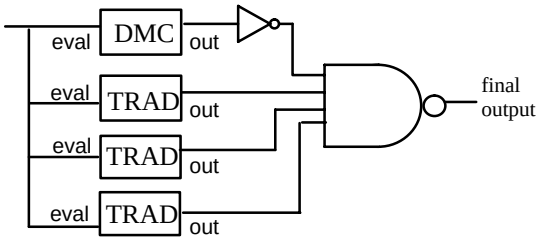
Each TRAD compares 8 bits

(a) 32–bit traditional comparator



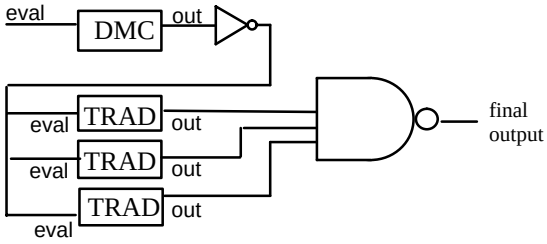
Each DMC compares 8 bits

(b) 32–bit dissipate–on–match comparator



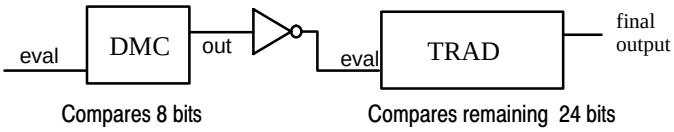
Each DMC and TRAD compares 8 bits

(c) 32–bit energy–efficient comparator: variation 1



Each DMC compares 8 bits

(d) 32–bit energy–efficient comparator: variation 2



(e) 32–bit energy–efficient comparator: variation 3

Figure 3. Some variations of energy–efficient comparators for comparing long arguments

matches, possibly partial. The data inputs to the comparator of Figure 2 (bits A0 through A7, B0 through B7 and their complements) are driven after buffering at the beginning of a clock cycle. Because of this and the sizing of the transistors within the P–blocks, circuit malfunctions due to charge sharing is avoided [1]. Since the outputs of all enabled comparators are latched at the end of every cycle, keeper devices are not needed within the comparator.

In [1], the circuit of Figure 2 is compared to the traditional comparator both in terms of both energy and delay. Significant energy savings result in situations where mismatches dominate; at the same time, the dissipate–on–match comparator design does not effect the critical path. In fact, the circuit of Figure 2 is slightly faster than the traditional comparator. As measured from SPICE simulations of our 0.18 micron TSMC layouts, the worst–case delay of the 8–bit comparator of Figure 2 is 108 ps in a full match case. This is actually faster than the response time of the traditional pull–down comparator of Figure 1, implemented in the same process, which has a delay of about 120 ps [1].

Like the traditional design of Figure 1, the energy–efficient comparator shown in Figure 2 is also limited to comparing only a few bits – up to 8 bits in practice. This restriction comes from the practical limit on the number of devices in a pass transistor logic (such as the P–structure of Figure 2) and from a limit on the number of n–devices that can be connected in series structure. These limitations can be partly overcome by using a Domino–style design (as shown in [1] and [5]), but practical limits are still present. The design of Figure 2 is thus applicable to scenarios where only a small number of bits are compared, as in the instruction issue queue and associatively–addressedregister alias tables (RATs).

3. LONG COMPARATOR DESIGNS

We now discuss several schemes for comparing longer arguments (upwards of 8 bits) in modern datapaths in an energy–efficient manner. Figure 3 shows the various possibilities for comparing 32–bit arguments. Some of these designs produce a high output on a full match, others produce a low output; the surrounding logic has to be designed accordingly.

The most obvious way to design a 32–bit comparator is to simply extend the circuit of Figure 1 to 32 bits. This, however, results in a very significant increase of the response time and power dissipation of the circuit, as discussed later. A better approach is to NAND the outputs of the four traditional (TRAD) 8–bit comparators from Figure 1, as shown in Figure 3(a). The comparator of Figure 3(a) outputs a logical “0” on a full match. The output can be inverted to generate a logical ”1” on a full match, but this increases the delay of circuit. Figure 3(b) shows a way of using a number of dissipate–on–match comparators for comparing longer arguments. Here, four 8–bit dissipate–on–match comparators from Figure 2 (DMCs) are used to compare four consecutive 8–bit segments of 32–bit comparands. The outputs of the DMCs are NOR–ed to indicate a full match. In contrast to the design of Figure 3(a), this

comparator outputs a high voltage level in the event of a full match. The comparator of Figure 3(b) is energy–efficient as long as the 8–bit segments are random, since mismatches dominate in all four blocks. However, it may not be energy–efficient in the case when some of the 8–bit blocks match often, as some of the DMCs will frequently detect a match and dissipate energy even when the complete arguments do not match.

Figure 3(c) depicts a possible solution for comparing long arguments in an energy–efficient manner in such cases. This design combines the dissipate–on–match comparators with traditional pull–down comparators. Here, one DMC and three TRAD comparators are used to detect a mismatch. As in the case of Figure 3(a), the NAND gate provides the required logic for producing an output of zero only on a full match. The delay of the inverter at the output of the DMC block is partially absorbed by the longer delay of the TRAD block, thus causing no significant increase in the overall response time. *Energy–efficiency is achieved by driving bits that are more likely to match into the TRAD comparators, while bits that are least likely to match are handled by the DMC comparator.* The mix of the number of DMCs and TRADs used can be changed as long as this basic principle is observed. In Section 5, we study the bit matching patterns of the comparands within the load–store queue and identify the most energy–efficient comparator design exploiting the comparand statistics.

The energy dissipation of the comparator of Figure 3(c) can be further reduced by using the conditional evaluations within some of the 8–bit blocks. In Figure 3(d), unless the first DMC produces a match, the other comparators are not evaluated. Energy savings occur in this case as the three bottom TRADs are not evaluated unless the DMC detects a full match, irrespective of the inputs to the TRADs. More explicitly, unless the top DMC detects a match, the (possibly partial) matches that the three bottom TRADs would otherwise detect do not cause any dissipation. The price paid, compared to the designs of Figure 3(b) or Figure 3(c) is a longer overall delay, since the evaluation delays of the top DMC, the inverter at the output of the DMC and the three bottom TRADs are all cascaded. In fact, according to the data obtained from our layouts, this increase in the delay is very significant, up to 100%, and we therefore did not consider this design despite its obvious advantages in terms of energy efficiency.

Figure 3(e) shows yet another way of comparing longer arguments in an energy–efficient fashion. Here a single DMC controls the evaluation of a single TRAD comparator that follows it. The delay of the NAND gate is eliminated, but the TRAD comparator adds a significant delay in the worst case. The overall delay is specific to the technology and the number of bits compared by the DMC and the TRAD. In our implementation, the delay of the circuit of Figure 3(e) is comparable to the delay of the circuit of Figure 3(d), again making it an unattractive design choice in high–speed pipelines. The comparator of Figure 3(e) produces the output of one on a full match, just like the traditional comparator of Figure 1 does.

The comparator designs shown in Figure 3 are examples of many others that are possible along similar lines. The main point we want to make here is that both the traditional comparator and the non–traditional (i.e., dissipate–on–match) comparators can be useful in designing fast, energy–efficient comparators for comparing long arguments. The exact combinations of these components or combinations using just one of these types are a function of the distribution of the bit patterns that are compared. The various design choices also trade off energy savings against delays. As an example of the application of fast, energy–efficient comparators in a modern out–of–order execution processor, we look at their use in detecting effective address matches in the course of a load bypassing earlier–issued stores in a load–store queue.

In the next Section, we describe our simulation methodology followed by the detailed evaluation of energy savings achievable within the long comparators used in the load–store queue.

4. EVALUATION METHODOLOGY

The Simplescalar simulator [3] was significantly modified to implement *true hardware level, cycle–by–cycle* simulation models for such datapath components as the ROB (integrating a physical register file), the issue queue, the load–store queue, the register alias table and the register file. The configuration of a simulated processor is shown in Table 1. The goal of the microarchitectural–level simulation was to capture the distribution of bit patterns that are compared in the course of load bypassing.

Table 1. Architectural configuration of simulated processors

Machine width	4–wide fetch, 4–wide issue, 4–wide commit
Window size	32 entry issue queue, 96 entry ROB, 32 entry load–store queue
Function Units and Latency (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 4 FP Add (2), 1FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
L1 I–cache	32 KB, 2–way set–associative, 32 byte line, 2 cycles hit time
L1 D–cache	32 KB, 4–way set–associative, 32 byte line, 2 cycles hit time
L2 Cache combined	512 KB, 4–way set–associative, 128 byte line, 8 cycles hit time
BTB	1024 entry, 4–way set–associative
Branch Predictor	Combined with 1K entry Gshare, 10 bit global history, 4K entry bimodal, 1K entry selector
Memory	128bit wide, 100 cycles first chunk, 2 cycles interchunk
TLB	64 entry (I), 128 entry (D), fully associative, 30 cycles miss latency

We simulated the execution of 10 integer *(bzip2, gap, gcc, gzip, mcf, parser, perlbmk, twolf, vortex* and *vpr*) and 8 floating point (*applu, apsi, art, equake, mesa, mgrid, swim* and *wupwise*) benchmarks from SPEC 2000 suite. Benchmarks were compiled using the Simplescalar GCC compiler that generates code in the portable ISA (PISA) format. Reference inputs were used for all the simulated benchmarks. The results from the simulation of the first 1 billion instructions were discarded and the results from the execution of the following 100 million instructions were used for all benchmarks.

For estimating the energy/power, the event counts gleaned from the simulator were used, along with the energy dissipations, as measured from the actual VLSI layouts using SPICE. CMOS layouts for the comparators in a 0.18 micron 6–metal layer CMOS process (TSMC) were used to get an accurate idea of the energy dissipations for each type of transition. A V_{dd} of 1.8 volts was used for all the measurements.

5. RESULTS

We now assess the usage of the long comparators in the load–store queue. The load–store queue (LSQ), which is used in modern superscalar processors to disambiguate the memory references, is an extremely comparator–rich structure. As the load and store instructions are dispatched, the entries are established in the LSQ in

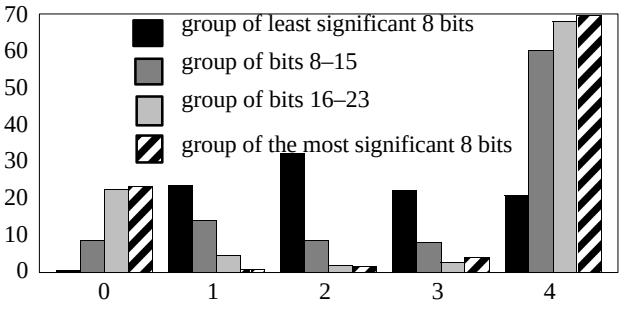


Figure 4. Percentage of partial and full matches in the 8–bit groups of the long comparators used within the load/store queue. Each set of bars represents the percentage of cases when a given number of bit pairs (implemented as P–blocks in Figure 2) match within each of the four 8–bit groups. Results are shown for the averages of all simulated SPEC 2000 benchmarks.

program order. To implement the load bypassing, the addresses of the load instructions are compared against the addresses of all previously dispatched store instructions and if no match is detected the load instruction is allowed to initiate its memory access ahead of the preceding stores. Such load bypassing reduces the effective memory latency. In case the address of one (or more) of the store instructions matches the address of the load, the value to be written to the memory by the most recent such store is forwarded to the load. Such local forwarding avoids the need to perform expensive memory accesses for some load instructions. To implement this mechanism, each LSQ entry is extended with a 32–bit wide address field and the long 32–bit comparators are used to perform the associative address matching.

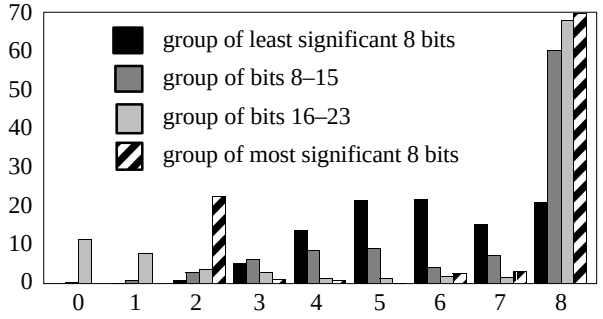


Figure 5. Distribution of comparison cases based on the number of matching bits. Each set of bars represents the percentage of cases when the number of matching bits is equal to the bar id. Results are shown for each of the four 8–bit groups independently. Results are shown for the averages of all simulated SPEC 2000 benchmarks.

The most obvious solution – the extended version of Figure 1, results in an evaluation delay of 464 ps, which is unacceptable for high frequency pipelines. The circuits of Figure 3(d) and 3(e) suffer from the same limitation – as discussed in Section 3. On the other hand, the first three designs shown in Figure 3 have acceptable delays (about 270 ps each) and we now attempt to identify the most energy–efficient choice among these three designs for the use in the LSQ. To understand the prevalent bit patterns of the comparands used in this case, we divided the 32–bit addresses into four groups of 8 bits each. We then collected the statistics about the matching patterns for each of the four groups independently. The results, averaged over all executed benchmarks, are shown in Figures 4 and 5. Figure 4 shows the percentages that are relevant for the DMC comparator and Figure 5 shows the percentages that are relevant for the traditional comparator.

Specifically, Figure 4 shows the percentage of cases when a given number of bit pairs (implemented as the P–blocks in Figure 2) match within each of the four 8–bit groups. It is interesting to observe from Figure 4 that the 8–bit groups exhibit quite disparate matching patterns depending on the significance of the bit positions constituting a group. The three most significant 8–bit groups match considerably more often than they mismatch: the match in the most significant 8 bits was recorded in more than 70% of all comparison cases, whereas the match in the next significant 8 bits occurs in 68% of the cases and the match in the group consisting of bits 8 to 15 occurs in about 60% of the cases. The match percentage is high because the higher–order bits specify the page number, which is typically identical for most of the neighboring load and store instructions, namely the ones that reside in the LSQ and whose addresses are being compared. Finally, a match in the least significant group occurs in only about 21% of the cases. These percentages have a direct implication on the choice of the most energy–efficient 32–bit comparators that can be used for the LSQ.

Table 2. Energy dissipations of the 8–bit DMC comparators used within 32–bit comparators.

Number of matching 2–bit groups in the comparands	Energy of the 8–bit DMC used to build a 32–bit comparator (fJ)
0	5.78
1	122.3
2	238.9
3	357.5
4	635.8

To estimate the energy impact, we used the layouts of several comparator variations presented in Figure 3. Treating the four blocks (DMC or TRAD) shown in Figures 3(a) and 3(b) as independent 8–bit comparators, we estimated the total power of the 32–bit comparator by combining the dissipations within the four blocks and the NOR (or NAND) gate driven by these blocks. We computed the energy dissipated within the 8–bit DMC block used to implement a 32–bit comparator of Figure 3(b) in the following manner. We assumed that the most significant 24 bits of the comparands mismatched in all bit positions, effectively resulting in no energy dissipation within the three bottom comparators of Figure 3(b). The entire dissipation of the circuit is then attributed to the dissipation of the top DMC comparator and the NAND gate. We then varied the bit combinations within 8 least significant bits of the comparands to obtain the dissipation of the circuit in various cases, depending on the number of 2–bit groups that match. Result are summarized in Table 2. Table 2 shows the various dissipations within the 8–bit DMC block if it used as a building block for a 32–bit comparator. The energy numbers are shown depending on the number of bit pairs (implemented by the P–blocks in Figure 2) that matched in the course of comparison. For each such matching pair, the charge has to be removed at the end of the cycle from the gate of the corresponding n–device (see Figure 2) to avoid false match in the next cycle. Results are applicable to all four DMC blocks used to

implement the circuit of Figure 3(b) – we actually verified this fact by simulations. Combined with the results of Figure 4, the energy numbers of Table 2 were then used to compute the energy dissipation within each 8–bit DMC block, as well as the dissipations within the entire 32–bit comparator. The results presented in Table 2 were obtained through SPICE simulations of the actual comparator layouts.

Table 3. Energy dissipations of the 8–bit traditional comparators used within 32–bit comparators

Number of matching bits in the comparands	Energy of the traditional comparator (fJ)
0	979.2
1	934.7
2	885.5
3	839.5
4	796.4
5	748.8
6	709.3
7	660.4
8	13.9

In a similar fashion, we estimated the power dissipation of the comparator shown in Figure 3(a). We assumed the full match in the most significant 24 bits (again, resulting in no dissipation within the three bottom TRAD blocks of Figure 3(a), and then estimated the power dissipated within the top TRAD block for various bit matching patterns within the least significant 8–bits. Results, similar to those presented in Table 2, are shown in Table 3. Combined with the percentages of Figure 5, results of Table 3 allow us to compute the power dissipation within the individual 8–bit TRAD blocks as well as the total power dissipation of the circuit of Figure 3(a). In a similar manner, we can also compute the dissipations of the comparator of Figure 3(c) and any other design that uses the combination of TRAD and DMC comparators. Notice that in the results presented in Table 3, the dissipation of the 8–bit TRAD comparator depends on the number of matching bits. As the number of matching bits increases, the energy dissipation decreases, because the drain nodes of the evaluate transistors corresponding to the matching bits are not precharged. (During the precharge phase, if the new inputs arriving in this cycle match, there is no direct path from the Vdd to the drain of the corresponding evaluate transistor).

For the group of the least significant 8 bits of the effective addresses that are compared for load bypassing, the DMC comparator of Figure 2 dissipates 43% less energy than the traditional comparator used for the same bits (on the average across all simulated benchmarks). This is because the percentage of matches in this group is fairly low. The situation is different for the other three groups – the traditional comparator is a more energy efficient solution there because of the high percentage of matches. On the average across all benchmarks, the traditional comparator is 78% more energy efficient than the DMC comparator for the second least significant group, 111% for the third least significant group and 142% for the most significant group. Overall, the design of Figure 3(a) is 30% more energy efficient (on the average across all simulated benchmarks) than the design of Figure 3(b). The design shown in Figure 3(c) is the most energy efficient solution for the 32–bit comparator used in the LSQ, because it takes into account the properties of the comparands within the individual 8–bit groups. Comparing the total energy dissipations within the 32–bit comparators, the design of Figure 3(c) is 19% more energy efficient than the design of Figure 3(a), if the comparators are used within the load–store queue.

6. CONCLUDING REMARKS

It is well recognized that traditional dissipate–on–mismatch comparator designs are energy inefficient when the majority of comparison cases result in full matches in the comparands. To address this deficiency, a number of dissipate–on–match comparators have been introduced, but those designs were limited to comparing at most 8–bit wide arguments.

In this paper we proposed a series of energy efficient long comparator designs for superscalar microprocessors. Our designs combine the use of 8–bit blocks built using traditional comparators with 8–bit blocks built using dissipate–on–match comparators. We then evaluated the use of these circuits within the address comparison logic of the load–store queues. We found that for the same delay, the hybrid design consisting of one dissipate–on–match and three traditional 8–bit comparators is the most energy efficient choice for the use within the load–store queue, resulting in 19% energy reduction compared to the use of four traditional 8–bit blocks. The results presented in this paper can be easily extended to the TLBs, highly–associative caches and the BTBs.

7. ACKNOWLEDGMENTS

This work was supported in part by DARPA through contract number FC 306020020525 under the PAC–C program, the NSF through award no. MIP 9504767 & EIA 9911099, and by IEEC at SUNY–Binghamton.

8. REFERENCES

[1] Ergin, O., Ghose, K., Kucuk, G., Ponomarev, D., "A Circuit–Level Implementation of Fast, Energy–Efficient CMOS Comparators for High–Performance Microprocessors", in Proc. of ICCD, 2002, pp.118–121.

[2] Brooks, D.M., Bose, P., Schuster, S.E. et al, "Power–Aware Microarchitecture: Design and Modeling Challenges for Next–Generation Microprocessors", IEEE Micro Magazine, 20(6), Nov./Dec. 2000, pp. 26–43.

[3] Burger, D., and Austin, T. M., "The SimpleScalar tool set: Version 2.0", Tech. Report, Dept. of CS, Univ. of Wisconsin–Madison, June 1997 and documentation for all Simplescalar releases.

[4] "Design of High–Performance Microprocessor Circuits", edited by A. Chandrakasan et.al, IEEE Press, 2001.

[5] Kucuk, G., Ghose, K., Ponomarev, D. and Kogge, P., "Energy–Efficient Instruction Dispatch Buffer Design for Superscalar Processors," in *Proc. ISLPED*, 2001, pp. 237–242.

[6]. Lin, K. and Wu, C., "A Low–power CAM Design for LZ Data Compression", IEEE Transactions on Computers, Vol. 10, 2000, pp.1139–1145.

[7] Zukowski, C., Wang, S., "Use of Selective Precharge for Low–Power on the Match Lines of Content–Addressable Memories",IEEE Int'l Workshop on Memory Technology, Design and Testing, 1997.