Xor-Masking: A Novel Statistical Method for Instruction Read Energy Reduction in Contemporary SRAM Technologies

Joonas Multanen, Timo Viitanen,
Pekka Jääskeläinen, Jarmo Takala
Tampere University of Technology, Finland
Email: joonas.multanen@tut.fi, timo.2.viitanen@tut.fi,
pekka.jaaskelainen@tut.fi, jarmo.takala@tut.fi

***Abstract*—Pervasive computing calls for ultra-low-power devices to extend the battery life enough to enable usability in everyday life. Especially in devices involving programmable processors, the energy consumption of integrated memories often plays a critical role. Consequently, contemporary memory technologies focus more on the energy-efficiency aspects with new custom CMOS SRAM cells with tailored energy consumption profiles constantly being proposed.**

This paper proposes a method that exploits such contemporary low power SRAM memories that are energy optimized for storing a certain logic value to improve the energy-efficiency of instruction fetching, a major energy overhead in programmable designs. The method utilizes a low overhead xor-masking approach combined with statistical program analysis to produce optimal masks to reduce the occurrence of the more energy consuming bit values in the fetched instructions.

In comparison to the "bus invert" technique typically used with similar SRAMs, the proposed method incurs minimal area overhead while still reducing the total energy consumption of an example LatticeMico32 core up to 5%. The improvement to instruction memory energy consumption alone is up to 13% with a set of benchmarks.

I. Introduction

With the emerging *Internet of Things* (IoT), more and more devices are becoming wearable, intelligent and wireless. Having small, usable devices that can process data, communicate wirelessly and operate for extended periods of time without external power source creates challenges for battery and processor technologies. Improved intelligence in devices often results in increased battery consumption due to the additional required digital logic. Since limitations of the current battery technologies inflict a hard upper bound on device usage time without having an external power source available, low-power and energy-efficient operation is an important requirement for embedded devices.

While processing devices are getting smaller, the amount of data being processed is growing, which makes power consumption of memories a critical aspect in ultra-low-power compute devices. It is not rare for on-chip memories to be responsible for half of the power consumption in CPU-based digital designs [1], [2]. In order to keep memories up to speed with other components, new technologies such as *Spin-Transfer Torque RAM* (STT-RAM) and *Embedded DRAM* (eDRAM), are studied as possible replacements to SRAM. However, these technologies are not yet mature, and process challenges such as cost, durability, and variability control must be overcome before they can be widely adopted. Unlike SRAM, they are *dedicated-process* [3] technologies, where significant modifications to the fabrication process need to be performed.

According to ISSCC, SRAM is still the technology of choice for fast on-chip memories and caches. However, also within the space of SRAM designs there is room for optimization. Traditional cell designs do not consider that instruction and program data are often biased towards containing either high or low logic values. If statistical information about the data to be stored exists, memory cells can be designed in an *asymmetric* fashion, so that they consume less power when holding, reading or writing the more often occurring logic value in that data, allowing reduction of the total energy consumed. These asymmetric cells have been designed and fabricated for memories and caches [4]–[7].

This paper takes under closer inspection the asymmetric SRAM designed and fabricated by Mori et al. [6], where reading the logical value one results in lower energy consumption compared to the value zero. In order to reduce the total energy consumption, the authors added a majority voting logic to increase the amount of logical ones stored, and thus read from their proposed memory.

In this paper we show that this type of asymmetric SRAMs can be beneficially exploited also for program instruction memories in statically scheduled processor cores by adding a very small area hardware logic, and utilizing offline program binary analysis. The paper is organized as follows. Section II reviews previous methods for low-power instruction and data encoding. Section III introduces the proposed method. It is evaluated and compared to an existing low-power encoding in Section IV. Section V concludes the paper.

II. Related Work

Previous work on low-power encoding has mostly concentrated on instruction address bus power and program data bus power reduction. The methods can be divided into static and dynamic methods, depending of the time the encoding is performed. Static encoding happens at program compile time, and dynamic at runtime using additional hardware logic.

Bus-invert encoding [8] was introduced by Stan and Burleson in 1995. They presented the encoding as a method to lower off-chip data bus power consumption. Later work has applied the encoding to segments of divided buses [9], [10]. In the bus-invert method, data is dynamically re-encoded based on the number of toggling bits between two consecutive data words (also known as the *Hamming distance*). If the number of toggling bits is more than half of the total bits, the data word is inverted with a logical NOT operation. This is done using a logic connected to the memory block, at the end of the bus. At the other end of the bus, where the data is consumed (typically a load-store unit or an instruction fetch), NOT is performed again to restore the original word, in case indicated by an extra control bit. The most apparent drawback of this technique is the additional control bit that needs to be stored per each word in the memory, and transferred to the consumers.

Petrov and Orailoglu [11] proposed a static method where the instruction data bus was encoded with 16 possible data transformation operations. The optimal set of transformations were found by using an exhaustive search considering two consecutive instruction words at a time. The encoded words were stored in memory when loading the program in, and the transformations to perform were communicated to the processor decode unit either in the beginning of the program or before application hot-spots, such as loops. This method needs frequently executed loops to perform well, also the reduction of toggled bits is better with small basic block sizes in comparison to large ones. In terms of energy consumption, the benefits from reduced successive word toggle activity can be significant when applied to an off-chip DRAM, or when the words are transferred via a long on-chip bus. However, the effect of bit toggles in the case of mostly reading data from a SRAM residing close to the consumer, a typical scenario with on-chip instruction memories, is small. In addition, the energy overhead of the additional decoding logic to implement the 16 transformations was omitted from the analysis.

Su et al. [12] used Gray coding and Cold scheduling to reduce bit switching on the instruction address bus. Gray coding exploits the fact that instructions are often fetched from consecutive memory addresses. By reordering the instructions in memory so that sequential addresses are located in Gray coded addresses, the instruction address bus toggling can be reduced. The authors combined Gray coding with Cold scheduling, where the program compiler schedules instructions using their relative energy costs to each other to minimize the total energy.

Musoll et al. [13] introduced an encoding based on the observation that usually programs operate on sets of same addresses, or *working zones*, repeatedly at a time. Like in Gray coding, this can reduce the address bus toggling. Working zones are given identifiers and these along with address offsets are sent to the instruction control logic to signal which working zone to use. This method can be expensive logic-wise, since the encoding and decoding algorithms are complex.

Benini et al. [14] analyzed instruction address traces to create custom encoder and decoder logic for a given processor. According to the authors, the method is generic and can be applied to various processors. This method showed little savings in the total energy when taking into account the energy required by the additional encoding and decoding logic.

Regarding instruction memory power consumption optimization, previous work mostly concentrates on the address bus power reduction, and, in particular, minimizing its bit switching activity. Many of the original methods focus on off-chip data bus energy reduction and disregard the energy overhead of the extra logic needed for encoding and decoding. In contemporary compute devices, the instruction data bus energy consumption might not be relevant due to instruction memories or caches integrated close to the consumer, the instruction fetch unit. On the other hand, in ultra-low-power designs, the energy and area overhead of the additional encoding and decoding logic required might be an important factor, since a complex implementation of the method can reverse the achieved gains.

Unlike previous solutions, our proposed method is able to utilize statistical analysis of the individual bit positions in instruction execution to produce an optimal, low-power encoding during compile-time with minimal required additional dynamic decoding logic.

III. Proposed Method

Most of the existing bus encoding algorithms concentrate on instruction address bus and program data bus encoding. They aim to reduce the bit switching activity on the buses, whereas the proposed method maximizes the occurrence of one logical value over the other. For the instruction address bus, the existing approaches exploit the fact that instruction addresses are often accessed in a sequential order, where techniques such as Gray coding the addresses can help. The fetched instruction data, on the other hand, is usually less sequential – the individual bits do not typically correlate with the previously fetched instruction word.

The proposed method we call *xor-masking*, uses statistical information about instruction memory accesses for individual programs to statically determine an optimal xor-mask to encode the instructions in each particular program to maximize the appearance of the desired logical value. The method is intended for designs in microcontroller or signal processing, where no operating system is used. This allows simpler handling of the *xor-mask*, since no dynamically linked code, system calls or context switches are used.

This encoding logic was used dynamically by Mori et al. [6] for general data. It resembles the original bus-invert logic, where the inverting decision is based on the Hamming distance between words.

The Hamming distance for two words of data can be calculated by taking the logical XOR between the words, and counting the '1' bits of the result. This is presented in Fig. 1. Here the previous (top) word has been inverted and we calculate the Hamming distance to the next word, including the added bit. In this case, we would invert the next (bottom) word, since the Hamming distance is greater than half of the word length. Counting the bits can be done with majority voters. Their implementation cost depends on the width of the data, and the type of the voter [15]. The decoding is done by XORing a given word with its added bit. For their SRAM, Mori et al., modified the original bus-invert by not minimizing the amount of toggles, but maximizing the amount of '1's. This corresponds to calculating the Hamming distance with the other word constantly being all '1's. This simplified the logic,

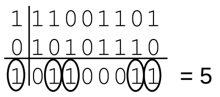

Fig. 1. Calculating the Hamming distance for two words for bus-inverting. Left-most bits are *toggle bits*. The two words are XORed and ’1’ bits in the resulting word are counted to make the toggle decision.

since the previous instruction or data value was not compared to the current one, thus saving a register and a logical XOR between the words. In this paper, this method is referred to as *Majority Voter Encoding* (MVE).

Encoding words for low power like this is not ideal, when the SRAM is used as an instruction memory. Statistical instruction analysis, such as in the proposed method, can lead to better maximizing of the low read-energy logic values in the memory. The analysis (see Fig. 3) starts by taking the instruction memory bit image of the optimized program. An instruction address trace with an unmodified instruction memory image is produced by simulating the targeted program using typical input data. Then, for each bit index, the total amount of logical ’1’s and ’0’s is weighted according to execution count is calculated. This process is illustrated in Fig. 2. If the amount of ’0’s is greater than half of total bits in the word, a ’1’ is assigned to that index of a *xor-mask* that is applied to the instruction word. That is, all the bits in that index will be inverted in the memory. The xor-mask is created for each application separately. For the memory examined in this paper, logic ’1’ reads are preferred for low energy. However, the method can equally efficiently be applied to cases, where logic ’0’ is the less energy consuming alternative.

Like the static instruction transformation of Petrov and Orailoglu [11], the proposed method conveys application-specific information to the processor. In our case, this information is reduced to a mere *xor-mask*, which is used to decode the encoded words during the instruction fetch stage. Fig. 4 depicts the logic required. Decoding the words is done by performing a logical XOR between the mask word and each encoded word. In addition to incurring extremely little additional logic, this allows easy implementation of the method to existing architectures.

The update of the XOR mask can be integrated with the program image, so that after reset the first instruction address holds the xor-mask. This is fetched and treated as a mask word which is stored to the mask register. After this, the succeeding

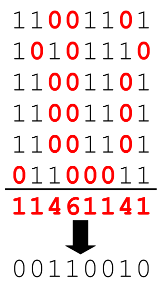

Fig. 2. Forming the xor-mask for an example instruction word trace. If the total amount of ’0’s for a given bit index is more than half (here three) of total bits for that index, it is xor-masked with a ’1’.

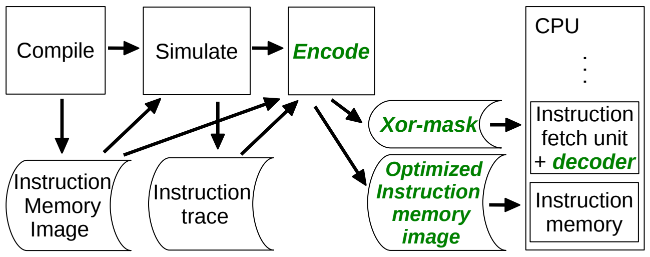

Fig. 3. Flow of the proposed encoding method. Stages and files for the proposed method are in bold italics.

words are treated normally as instructions. For even easier integration with program compilers, the mask word and a jump to reset address can be placed to the last addresses in instruction memory to avoid relocating the program instructions. For real world applications, program code for the static analysis should be weighted realistically and simulated with a typical input. If the simulated execution does not match a realistic scenario, the resulting suboptimal mask could cause an increase in energy consumption.

The decoding logic overhead in the proposed method is very low. Moreover, encoding the words incurs no additional hardware logic, since the encoding is performed offline as a final pass over the instruction bit image.

As a side effect of an encoding that aims to maximize one logical value over another, the proposed method often reduces bit toggling between successive instruction words. In case of a multi-level instruction memory hierarchy, the reduced toggling can save additional energy since off-chip buses can consume considerable amounts of energy.

IV. Evaluation

Xor-masking was implemented on LatticeMico32 processor [16]. LatticeMico32 is a RISC architecture with an open licensing agreement and freely available RTL source code. Detailed specifications of the evaluated processor are listed in Table III .The evaluation platform was a minimal setup, where instruction and data memories were scaled to be large enough to accommodate all of the individual bechmark program instructions and data at a time. The benchmark programs were chosen to represent typical applications in microcontrollers in low-power scenarios. The benchmarks are listed in Table IV.

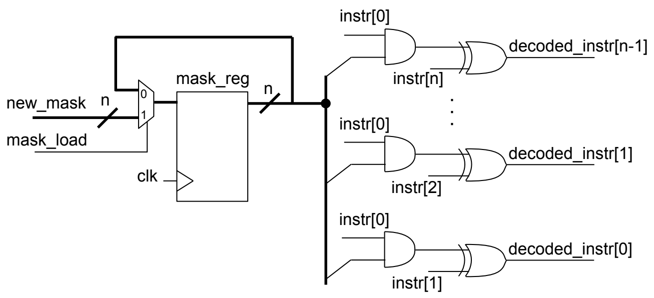

Fig. 4. Implementation of the decoding logic. First word after reset is read in as a new mask. Mask is active, if first bit in instruction is logic ’1’. Instructions are decoded by XORing them with the mask.

TABLE I. ENERGY CONSUMPTION.

Benchmark	Original (pJ)	MVE Δ(%)	Proposed Δ(%)
adpcm	178 000	-53.0	-57.2
aes	103 000	-54.5	-57.6
blowfish	1 751 000	-57.1	-61.2
coremark	1 071 000	-56.7	-59.3
matrix	664 000	-55.3	-63.1
fir	1 000	-62.9	-67.3
gsm	56 000	-56.0	-59.9
jpeg	5 967 000	-55.2	-58.4
lms	2 000	-55.2	-63.1
mips	51 000	-54.7	-57.9
sha	1 459 000	-54.6	-61.5

TABLE II. BIT SWITCHING ACTIVITY.

Benchmark	Original	MVE Δ(%)	Proposed Δ(%)
adpcm	789 000	-3.2	-6.4
aes	480 000	-13.9	-13.6
blowfish	6 773 000	-7.6	-7.8
coremark	5 771 000	-16.9	-16.3
matrix	3 283 000	-8.7	-8.8
fir	9 000	-48.7	-50.0
gsm	291 000	-20.7	-21.1
jpeg	28 492 000	-12.2	-13.1
lsm	12 000	-36.3	-36.5
mips	239 000	-10.8	-11.8
sha	5 971 000	-2.6	-5.4

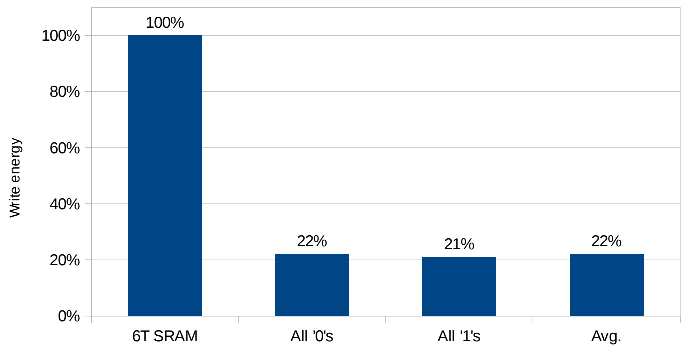

Fig. 5. Write energy consumption [6] for the referred SRAM.

The energy consumption was evaluated by calculating energy costs for reading a single '1' and '0' based on the previously published measurements for the original SRAM [6]. In the original 64kB SRAM, for a 16-bit word, energy consumption for reading all '0's and reading all '1's was 1440 fJ/cycle and 148.5 fJ/cycle, respectively. Dividing these by 16 yielded 90.00 fJ/cycle and 9.28 fJ/cycle for the cost of reading a single '1' bit and a '0' bit, respectively. Next, instruction address traces for the benchmark programs were produced from Modelsim simulations. Using the calculated read energies, instruction traces and instruction memory bit images encoded according to each of the two methods, the total SRAM energy consumption for each benchmark program was calculated. These are presented in Table I.

Considering the write energy consumption is relevant for processors with an instruction cache. Cache misses translate to writing cache lines. The difference in energy reduction for writing all '0's and all '1's to the referred SRAM, compared to a regular SRAM, was reported as negligible, 1%. This is illustrated in Fig. 5. The proposed method does not affect the time of replacing cache lines compared to the referred method. In this sense, write energy evaluation is not interesting, since the difference between the proposed method and the referred one is not significant.

If both the instruction memory and the instruction cache are implemented with the referred SRAM technology, the total energy reduction depends on the combined energy consumption of the two. In this case, the decoding logic would be implemented after the instruction cache.

In the case of the proposed method, the numbers also include the energy consumed by the majority logic proposed with the referred SRAM technology. In our case, this majority logic is not used and instead, the decoding logic would be implemented in the instruction fetch unit. This overhead is present in our energy numbers for the proposed method. The improvement in energy consumption of our proposed method compared to the referred method would, therefore, be better than the results presented in this paper. However, this overhead is difficult to estimate, since the authors did not report the majority logic energy consumption individually, but rather the overall SRAM's. Moreover, the majority logic relies on a pull-down network and a sense amplifier, However, regardless of this overhead, our proposed method still achieves lower energy consumption in all 11 benchmark programs.

Energy comparison normalized to MVE for the benchmark programs is presented in Fig. 8. As is expected, the energy consumption depends on the dynamic instruction mix in each of the benchmarks. The more there are instructions resembling each other on the bit level, the more the proposed method can save energy. The worst case is when the occurrence of '1' and '0' at each bit position is exactly the same. In this case, inverting the bit index results in no savings in energy.

The energy overhead of the added logic is small compared to the overall energy, on average 1.0% of the SRAM energy. The best energy reduction, 13.3%, was achieved in *matrix* benchmark and the lowest reduction, 1.5%, in *coremark*. On average, the reduction was 6.2%. To estimate the CPU total energy consumption, the described LatticeMico32 was synthesized on a 28nm ASIC standard cell technology. The instruction memory consumed 37.7% of the total energy after synthesis for this particular implementation. Using this number, the effect on the total CPU energy was calculated. This is presented in Fig. 6. The largest total energy reduction, 24.8%, was achieved with *fir*. In *matrix*, total CPU energy consumption was 5.0% less compared to MVE. On average, this reduction was 2.4%.

TABLE III. LATTICEMICO32 FEATURES.

Clock frequency	18.2MHz
Instruction set architecture	RISC
Instruction width	32 bits
Instruction memory	On-chip SRAM, 32kB
Data memory	On-chip SRAM, 400kB
Dedicated hardware	Hardware multiply unit Hardware divide unit Pipelined barrel shifter

TABLE IV. BENCHMARK PROGRAMS.

Suite	Programs
CHStone [17]	adpcm, aes, blowfish, gsm, jpeg, mips, sha
DSPStone [18]	matrix
Coremark [19]	coremark

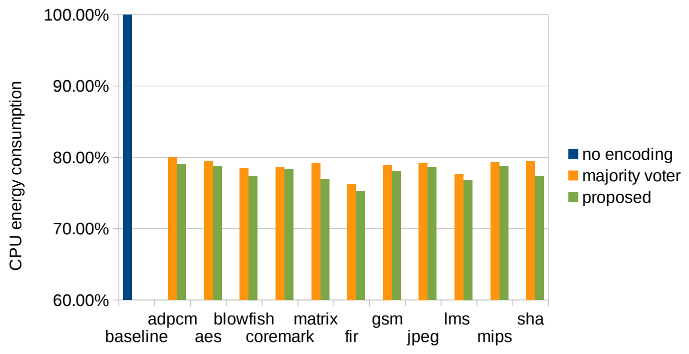

Fig. 6. Comparison of total CPU energy for MVE and the proposed method. Normalized to the level of CPU with unencoded data in SRAM.

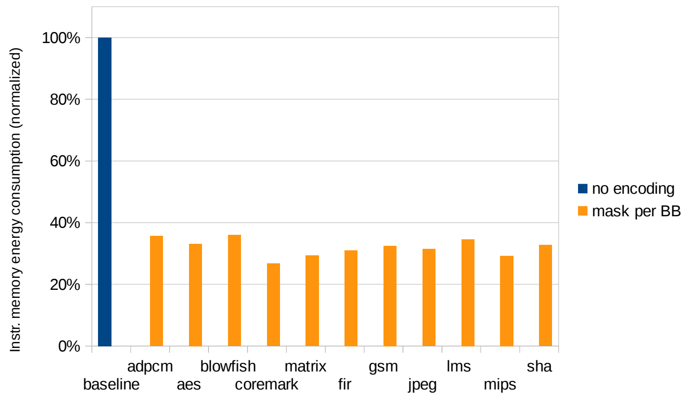

Fig. 7. Normalized energy consumption for benchmark programs, each basic block with individual mask. The energy overhead of mask updating is not considered.

The reduction in bit switching activity is compared to the original instruction words and the majority-voter-encoded words. This is presented in Table II. Both of the encoding methods add a toggle bit to the unencoded words, increasing the total amount of bits read. The proposed method reduces the bit switching activity in all but two benchmark programs compared to the MVE. The reductions are small and in the best case 3.3%. In a realistic implementation in an ultra-low-power IoT device, off-chip memory would be unlikely to be used for storing instructions and the effect in energy reduction for an on-chip bus would be negligible.

To evaluate the possible energy reduction from using multiple masks, basic blocks for the benchmark programs were formed from the instruction traces. A xor-mask was computed for each basic block. Then, energy consumption was again calculated using the numbers for the referred SRAM. The results are presented in Fig. 7. The numbers do not include the energy overhead from mask updating. The results seem encouraging, and suggest that further investigation could be beneficial, since theoretically, up to 74% reduction could be achieved (in coremark benchmark).

V. Conclusions

Energy consumption of on-chip and off-chip memories offers optimization opportunities in pervasive compute devices. In this paper, a novel statistical method, xor-masking was

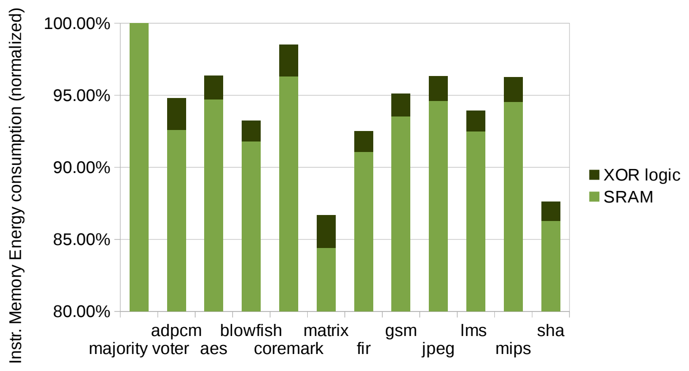

Fig. 8. Proposed method's instruction memory energy consumption for benchmark programs, normalized to the corresponding consumptions with MVE.

proposed to reduce the instruction fetch energy consumption in asymmetric SRAM technologies. The proposed method was evaluated on LatticeMico32 RISC soft-core with 11 benchmark programs.

Including the energy overhead of the decoding logic, the proposed method consumes up to 13% less energy compared to the state-of-the-art majority voter encoding on the same SRAM. The total CPU energy reduction is up to 5% compared to majority voter encoding. The energy consumption with the proposed method was smaller in all benchmark programs compared to the majority-voter-encoding. In addition, the proposed method reduces instruction data bus toggling up to 3.3% compared to the referred method.

Initial evaluation of using multiple xor-masks per program suggest further possibilities for energy reduction. Future work involves researching the use of multiple masks for a given program. This involves at least investigating the granularity of the masking (basic block/instruction level), number of masks and mask updating strategies for maximal energy reduction. Careful consideration of the mask updating overhead is required.

VI. Acknowledgments

The authors would like to thank the Finnish Funding Agency for Technology and Innovation (project "Parallel Acceleration 3", funding decision 1134/31/2015) and ARTEMIS JU under grant agreement no 621439 (ALMARVI).

References

[1] D. Bol, J. De Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J. Legat, "SleepWalker: A 25-MHz 0.4-V Sub-mm^2 7- μm^2 μW/MHz microcontroller in 65-nm LP/GP CMOS for low-carbon wireless sensor nodes," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 20–32, Jan. 2013.

[2] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, June 23-25 2010, pp. 21–21.

[3] L. Benini, A. Macii, and M. Poncino, "Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 1, pp. 5–32, Feb. 2003.

[4] N. Azizi and F. Najm, "An asymmetric SRAM cell to lower gate leakage," in *Proceedings of the 5th International Symposium on Quality Electronic Design*, Hangzhou, China, Mar. 15-16 2004, pp. 534–539.

[5] M. Imani, S. Patil, and T. Rosing, “Hierarchical design of robust and low data dependent FinFET based SRAM array,” in *Proceedings of the International Symposium on Nanoscale Architectures*, Boston, MA, July 8-10 2015, pp. 63–68.

[6] H. Mori, T. Nakagawa, Y. Kitahara, Y. Kawamoto, K. Takagi, S. Yoshimoto, S. Izumi, K. Nii, H. Kawaguchi, and M. Yoshimoto, “A 298-fJ/writecycle 650-fJ/readcycle 8T three-port SRAM in 28-nm FD-SOI process technology for image processor,” in *Proceedings of the IEEE Custom Integrated Circuits Conference*, San Jose, CA, Sep. 28-30 2015, pp. 1–4.

[7] A. Teman, A. Mordakhay, J. Mezhibovsky, and A. Fish, “A 40-nm subthreshold 5T SRAM bit cell with improved read and write stability,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 12, pp. 873–877, Dec. 2012.

[8] M. Stan and W. Burleson, “Bus-invert coding for low-power I/O,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 3, no. 1, pp. 49–58, Mar. 1995.

[9] Y. Shin, S.-I. Chae, and K. Choi, “Partial bus-invert coding for power optimization of application-specific systems,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 9, no. 2, pp. 377–383, Apr. 2001.

[10] J. Gu and H. Guo, “A segmental bus-invert coding method for instruction memory data bus power efficiency,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Taipei, Taiwan, May 24-27 2009, pp. 137–140.

[11] P. Petrov and A. Orailoglu, “Application-specific instruction memory customizations for power-efficient embedded processors,” *IEEE Design Test of Computers*, vol. 20, no. 1, pp. 18–25, Jan. 2003.

[12] C.-L. Su, C.-Y. Tsui, and A. M. Despain, “Saving power in the control path of embedded processors,” *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 24–30, Winter 1994.

[13] E. Musoll, T. Lang, and J. Cortadella, “Working-zone encoding for reducing the energy in microprocessor address buses,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 6, no. 4, pp. 568–572, Dec. 1998.

[14] L. Benini, G. De Micheli, E. Macii, M. Poncino, and S. Quez, “System-level power optimization of special purpose applications: the beach solution,” in *Proceedings of the International Symposium on Low Power Electronics and Design*, Monterey, CA, Aug. 18-20 1997, pp. 24–29.

[15] B. Parhami, “Design of m-out-of-n bit-voters,” in *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems and Computers*, vol. 2, Pacific Grove, CA, Nov. 4-6 1991, pp. 1260–1264.

[16] Lattice Semiconductor. (2016, Feb.) Latticemico32. http://www.latticesemi.com/en/Products/Design SoftwareAndIP/ IntellectualProperty/ IPCore/IPCores02/LatticeMico32.aspx.

[17] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, “Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis,” *Journal of Information Processing*, vol. 17, pp. 242–254, Oct. 2009.

[18] V. Zivojnovic, J. Martinez, C. Schlger, and H. Meyr, “DSPstone: A DSP-oriented benchmarking methodology,” in *Proceedings of the International Conference on Signal Processing Applications and Technology*, Dallas, TX, Oct. 18-21 1994, pp. 715–720.

[19] EEMBC – The Embedded Microprocessor Benchmark Consortium. (2016, Feb.) Coremark benchmark. Http://www.eembc.org/coremark.