NBTI-Aware Data Allocation Strategies for Scrathpad Based Embedded Systems

Cesare Ferri, Dimitra Papagiannopoulou, and R. Iris Bahar
School of Engineering
Brown University
Providence, RI 02912
Email: cesare_ferri@brown.edu

Andrea Calimera
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy 10129
Email: andrea.calimera@polito.it

***Abstract*—While performance and power continue to be important metrics for embedded systems, as CMOS technologies continue to shrink, new metrics such as variability and reliability have emerged as limiting factors in the design of modern embedded systems. In particular, the reliability impact of pMOS negative bias temperature instability (NBTI) has become a serious concern. Recent works have shown how conventional leakage optimization techniques can help mitigate NBTI-induced aging effects on cache memories. In this paper we focus specifically on scratchpad memory (SPM) and present novel software approaches as a means of alleviating the NBTI-induced aging effects. In particular, we demonstrate how intelligent software directed data allocation strategies can extend the lifetime of partitioned SPMs by means of distributing the idleness across the memory sub-banks.**

I. Introduction and Background

Memory subsystems have long been known to be a critical component in the overall performance for embedded platforms. However, more recently, these memory systems have also been demonstrated to be the major contributor to the total power budget. The problem of power dissipation in memories is exacerbated in ultra-deep sub-micron CMOS technologies, where static power due to leakage currents (and sub-threshold current in particular) is coupled with high dynamic power dissipation [1].

The leakage power is extremely critical for memories, where the high density of integration translates into high power density. The latter is the main source of heat generated across the substrate, which, if not quickly removed through packaging and/or cooling architectures, may induce drastic increases in temperature. Working at high on-chip temperatures affects both performance (MOS transistors and global wires get slower at high temperatures [2], [3]) and static power consumption (leakage current increases exponentially with temperature [1]). For these reasons, several leakage-aware memories solutions, ranging from software and system level hierarchy optimization to architecture and circuit level structures, have been proposed over the past few years (e.g., [4], [5]).

While performance and power continue to be important metrics for embedded systems, as CMOS technologies continue to shrink down below 65nm, new metrics such as *variability* and *reliability* have emerged as limiting factors in the design of modern embedded systems. This is especially true when considering memory architectures; access to/from memory is involved in every executed instruction (e.g., data and instruction fetch), and once the memory system becomes unreliable, the reliability is compromised for the system as a whole.

Although process variation is the most evident source of variability, and thus, unreliability [6], [7], time-dependent deviations in the operating characteristics of nanoscale MOS devices [8] have recently been shown to be one of the more critical issues in determining the lifetime of CMOS circuits. In particular, the reliability impact of pMOS negative bias temperature instability (NBTI) has become a serious concern [9].

In CMOS logic, NBTI effects occur on pMOS transistors when they are negatively biased (i.e., a 0-logic is applied to the gate of the pMOS, resulting in $V_{gs} = V_{DD}$), and manifest themselves as an increase in the threshold voltage V_{th} over time [9]. Such a V_{th} variation has a direct impact on the long-term stability of traditional 6T-SRAM cells, whose static noise margin (SNM) degrades over time, thus altering the capability of a cell to reliably store a correct logic value [10]–[12]. Experimental data report variation of V_{th} of about 10-15% per year, which translates into more than a 10% SNM degradation (after 3 years) depending on the target technology and the operating conditions [11].

The push to embed *reliable* and *low-power* memories architectures into modern systems-on-chip is driving the EDA community to develop new design techniques and circuit solutions that can address these critical issues. Very recent works [13], [14] have shown how conventional leakage optimization techniques [4], [5] can offer a valuable solution to mitigate NBTI-induced aging effects on memories while still obtaining reductions in power dissipation. More specifically, they have quantified the beneficial effects of popular power management schemes such as *power-gating* and *dynamic voltage scaling* (DVS) on aging. Power gating, when implemented using sleep transistors, has the effect of completely nullifying the aging effects [15]. Similarly, but with a smaller impact, voltage scaling improves NBTI-induced aging because a reduced V_{DD} corresponds to a smaller bias voltage. As reported in [13], reduction in SNM as a function of V_{dd} is roughly linear; the degradation of SNM under a "drowsy" voltage $V_{dd,drowsy}$ is about 60% of the degradation at the nominal V_{dd}.

Although the majority of these works focus on cache memories, there exist other types of SRAM memory structures whose efficiency also impacts the total power-performance tradeoff, as well as on the lifetime of the embedded systems. Among them, scratch-pad memory (SPM) is the most significant example. SPMs are widely used in embedded systems for image and video processing applications that make heavy use of multi-dimensional arrays of signals and nested loops. For these classes of applications, the flexibility of caches in terms of workload adaptability is often not needed; instead, performance predictability, power consumption, and implementation costs play a much more critical role.

Even though caches and SPMs perform the same function (namely, temporarily holding small chunks of data for high-speed, frequent retrievals), their implementation and management is completely different. The main differentiating factor is that, while caches guarantee full transparency at the cost of more hardware resources and non-deterministic latency, SPMs, which do not need any caching logic, are faster, more predictable and less power consuming. Note that for SPMs, it is the designer that decides the mapping of addresses to locations into the SPM, not the hardware.

This new degree of freedom in the design space opens a completely unexplored area in the field of concurrent power-aging memory optimization. Namely, it allows for the use of innovative software approaches as a means of alleviating the NBTI-induced aging effects. In contrast to previous works that focus on pure circuit/architectural cache solutions (e.g., [14], [16]), in this paper we investigate new software controlled NBTI-aware data managing solutions for low-power SPMs.

Building off of previous findings that SRAM cells in a low-leakage state (i.e., idle state) are less affected by NBTI stress [13], we demonstrate how intelligent data allocation strategies can extend the lifetime of partitioned SPMs by means of distributing the idleness across the memory sub-banks. The basic reasoning behind this approach is that, while from a leakage viewpoint it is the total number of idle blocks that matters, for aging, it is the *distribution of idleness* across the memory banks that can help maximize the lifetime of the entire SPM, and thus the system as a whole.

To support our claims we developed a dedicated library of C-functions that implement NBTI-aware data allocation through a dedicated *malloc*, called *SPM_malloc*. This function is aware of the current aging of each memory bank and maps the heap of each task such that all the banks age at the same rate. More specifically, the data are allocated such that all the banks can spend the same amount of time in the idle state (low-V_{DD} state). Dedicated lookup tables containing pre-characterized NBTI-induced SNM degradation of a standard 6T-SRAM cell mapped into an industrial 45nm technology are used to estimate the aging of the SPM's sub-banks. We demonstrate through a motivational example the effectiveness of our approach and show that overall idle times across all banks of the SPM can be more evenly distributed, thereby preventing some banks from aging faster than others and increasing the reliability of the memory system overall.

II. Architectural Overview

Our target architecture is shown in Figure 1. The baseline configuration consists of an ARM7 CPU, coupled with a L1 cache and a block of scratch-pad memory. A Direct-Memory Access engine (DMA) is also present, and it is in charge of accelerating the data movements between the on-chip and off-chip memories. This type of architecture has many applications (e.g., smartphones, cameras, game consoles etc.), and it is widely adopted in the embedded domain. Next, we describe the proposed hardware and software extensions.

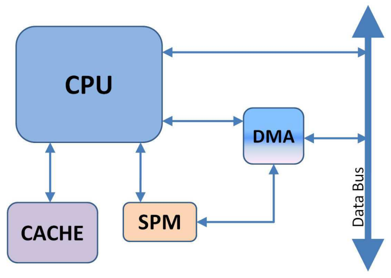
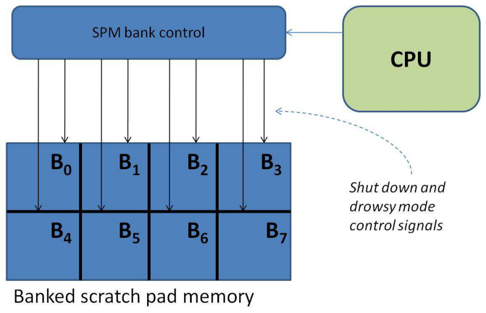


Fig. 1. Architectural overview.

A. Hardware Extensions

Similarly to [17], we propose a multi-banked type of scratch-pad memory, with independently powered banks. This kind of structure has been studied also in caches (e.g., [18]), mainly for power concerns.



SPM bank control
CPU
B_0
B_1
B_2
B_3
B_4
B_5
B_6
B_7
Shut down and drowsy mode control signals
Banked scratch pad memory

Fig. 2. Scratchpad memory configuration.

As shown in Figure 2, a special control unit sets the power state (i.e., *active*, or *drowsy*) for each bank. We deliberately didn't consider a shutdown state, since that configuration would complicate the design of the scratch-pad memory. The insertion of extra sleep transistors in the memory cells, in fact, would not only impact on the complexity but also on the performance of such critical component.

The control unit is memory mapped, and can be programmed with regular write operations. In order to simplify the programming process, we implemented some interfacing functions (described in Section II-C). We estimate that reactivating a bank from the *Drowsy* state incurs a realistic performance

TABLE I
FUNCTIONS VISIBLE TO THE PROGRAMMER.

Name	**Description**	**Example of Utilization**
void* **SPM_malloc**(int size, unsigned int** requestor);	Allocates data in SPM. Two input parameters: 1) Size 2) Reference to the pointer of the object that must be allocated	int *A= (int*)SPM_malloc(sizeof(int),&A)
int **SPM_free**(void *point);	Frees memory in the SPM	SPM_free(A);
void **SPM_initialize_blocks**();	Initialize metadata. Must be called at boot time.	

TABLE II
FUNCTIONS NOT VISIBLE TO THE PROGRAMMER (I.E., INTERNAL TO THE LIBRARY).

Name	**Description**	**Example of Utilization**
void* **SPM_malloc_bank**(int size, int bank);	Input parameters:. 1) size 2) which bank in SPM	int *A=(int*)SPM_malloc_bank(sizeof(int),0)
int **SPM_obj_table_move**(unsigned int *obj, unsigned int *dst, int USE_DMA=0);	Input parameters: 1) object address 2) destination address 3) optional: DMA flag	int *A=(int*)SPM_malloc(sizeof(int),&A); int *B=(int*)SPM_malloc(sizeof(int),&B); SPM_obj_table_move(A,B);
float **get_aging**(float psleep, float time)	Input parameters: 1) sleep time/total time 2) total time Output: SNM degradation	float SNM=get_aging(psleep,time)
int **SPM_banks_manager**();	Main function that should rearrange the data inside the blocks to minimize the overall SNM of the SPM.	SPM_banks_manager();
SPM_SET_BANK_STATE(int ID, int state)	Changes the power state {ON,DROWSY} of the bank ID by writing on the SPM control unit	SPM_SET_BANK_STATE(0, SPM_BANK_STATE_DROWSY)
SPM_RETURN_BANK_STATE(int ID)		

penalty of 0.2μs. This overhead is in line with that of other commercial embedded processors (e.g., [19], [20]).

B. Extension to Memory Allocation

Our sturdy is based on the assumption that a relevant portion of embedded systems do not have *a priori* knowledge of the running tasks (e.g., downloading new applications on a smartphone). It may also be that the memory required by a task is unpredictable at compile time (e.g., I/O triggered memory allocation). For such systems, static (i.e., compile-time) memory allocation strategies can not be applied. Several dynamic approaches have been proposed in the past, but none of them were dealing with software exposed multi-banked SPM architectures.

From the programmer point of view, we propose a simple extension to the regular *malloc()*, as described in Table I. We replace the call to *malloc()* with the new function *SPM_malloc()*. The *SPM_malloc()* method accepts a new parameter: the address of the 'requestor' (i.e., the pointer to the object to allocate). The address of the requestor is fundamental in case we want to move the data from one bank to another. In that case, in fact, also the pointers to the objects allocated (i.e., requestors) in the source bank should be updated with new values.

Updating the requestors is done automatically by several library functions that are hidden to the programmer. Table II reports these main library functions. Note that extra metadata was required to manipulate the banks and hence he object stored in the banks. For this reason, we create two new tables, namely *spm_obj_table* and *spm_bank_info*, containing the actual state of each bank and object, as shown in Table III.

TABLE III
EXTRA METADATA.

Name	**Description**
SPM_OBJ_TABLE_T **spm_obj_table**[SPM_MAX_OBJ];	Table of objects allocated in the scratchpad.
SPM_BANK_INFO_T **spm_bank_info**[SCRATCH_BANKS];	Table of registers for each bank.

Next, we will describe the most important library function: *SPM_block_manager()*.

C. The SPM_block_manager() Procedure

The key feature of this method is to move automatically the data from one partition to another whenever the aging indices of two banks differ for a quantity bigger than a given threshold. Figure 3 shows the call precedence graph for managing such a function. At the moment the function is called from within the *SPM_malloc()* method (i.e., every time an object must be allocate), but this is not a requirement. The OS could potentially invoke that functions after regular intervals of time (e.g., once every hour).

The algorithm starts with all the banks of the scratchpad memory turned off. At the beginning of the algorithm, one bank must be turned on for the first time. After we turn on the first bank, the algorithm estimates the maximum difference

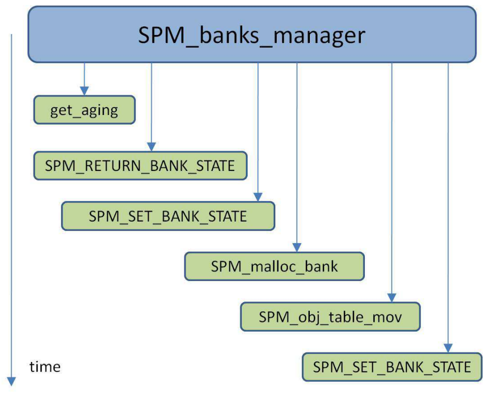


Fig. 3. Calling precedence graph.

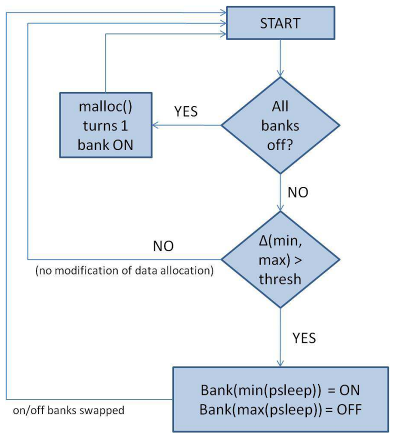


Fig. 4. Decision flow for data allocation in SPM banks.

in aging between all the turned on and turned off banks. This is done by calculating the value of *psleep* for all banks ($0 < i <$ number of banks) where

$$psleep_i = \frac{total\_idle\_time_i}{total\_time\_of\_operation}, \quad (1)$$

and $total\_idle\_time_i$ is the sum of all the time intervals that bank i has been turned off until now, or

$$total\_idle\_time_i = \Sigma(idle\_intervals_i). \quad (2)$$

We expect that the smaller the *psleep* value for a bank, the greater its aging, based on the fact that a bank that is turned off for a smaller percentage of overall operation will be exposed to more aging effects.

The algorithm then computes the difference between the *psleep* value of the off bank with the maximum *psleep* value (i.e., the least aging) to the *psleep* value of the on bank with the minimum *psleep* value (i.e., the greatest aging). Mathematically, this can be expressed as:

$$\begin{aligned} psleep_{max} &= \max(psleep_j) \quad j \in OFF\ banks \quad (3)\\ psleep_{min} &= \min(psleep_k) \quad k \in ON\ banks \quad (4)\\ \Delta psleep &= psleep_{max} - psleep_{min} \quad (5) \end{aligned}$$

If $\Delta psleep$ is greater than a given threshold, then these two banks with min/max *psleep* values must switch data and status. That is, the selected turned off bank will be turned on and will accommodate the data written in the selected turned on bank and this turned on bank will now be turned off. If the difference between spleep values is negative, then no modification will be made on the data that are allocated in the banks, or the status of the banks. Figure 4 illustrates the decision flow of the algorithm.

At the end of the algorithm we expect that the *psleep* values will be more evenly distributed among the banks and that the worst *psleep* value (i.e., the smallest one) will have increased, indicating a smaller worst case bank aging than before.

III. Preliminary Results

Due to time limitations we will not be able to give extensive results here. Instead, we will have to rely on a motivational example that shows how we can influence the shift in static noise margin with our software managed NBTI-aware memory allocation policy.

As explained in SectionII-B, we are targeting systems which do not have a compile-time knowledge of the running tasks. To mimic this system behavior, we implemented a microbenchmark that executes a parameterizable number of tasks, each of them instantiating a certain amount of objects. The tasks are chosen from a set of basic matrix manipulation functions (ie. multiplication, addition and lu decomposition). Each simulation accepts for input the following parameters:

1) percentage of SPM utilization (20, 40, 60, 80)
2) SNM threshold that triggers the bank change

We investigated three different bank management policies:

vanilla:
The SPM contains a single bank always turned ON.

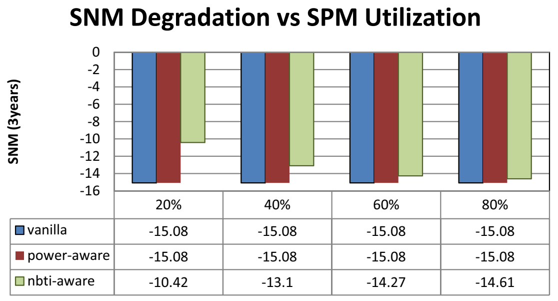
power-aware:
The banks can be set in the Drowsy state to save power. This policy is unaware of any aging effect.

NBTI-aware:
The aging status of the banks is monitored by the *SPM_bank_manager()* function. The banks are set in Drowsy or Active state according to the SNM degradation.

For each simulation, we collect the worst of the SNM degradations values, since the the worst bank (i.e., the one with highest aging) defines the aging of the entire SPM.

Figure 5,shows the SNM degradation after three years for different percentages of SPM utilizations. As expected, we see that the proposed NBTI-aware policy reduces significantly the SNM degradation. Note that, from an aging prospective, the power-aware and the vanilla policies are equivalent, since in the minimum value for the idleness (which is proportional to the aging) is in both case 0. The proposed NBTI-aware technique, instead, tries com equalize the different aging indexes by triggering changes in the power states of the banks.

Figure 6 shows the minimum *psleep* value, i.e., the percent of idle time for the bank that has been active for the most time.



	20%	40%	60%	80%
🟦 vanilla	-15.08	-15.08	-15.08	-15.08
🟥 power-aware	-15.08	-15.08	-15.08	-15.08
🟩 nbti-aware	-10.42	-13.1	-14.27	-14.61

Fig. 5. SNM vs. percentage of SPM utilization

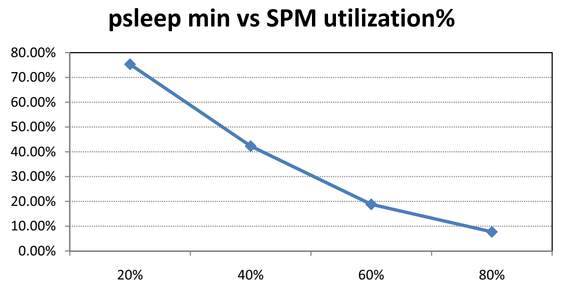


Fig. 6. *psleep* min. vs. percentage of SPM utilization.

As expected, as we increase the percentage of utilization of the SPM, the benefits of our proposed solution diminish. The chances of finding free space to allocate and move data from another bank (which will be also set in the Drowsy state) to another are in fact reduced.

IV. Conclusions and Future Work

In this paper we have introduced a new method for managing memory allocation in scratchpad sub-banks such that the effect of NBTI aging are reduced. Since the SPM is managed by software in terms of address mapping, it is appropriate that we present a method for NBTI aware data allocation that is also managed by software. Through simple extension to the regular *malloc()* we show through a simple motivational example that the idle times for each bank in the SPM is more evenly distributed, compared to a NBTI-oblivious memory allocation scheme. This even distribution prevents some banks from aging faster than others, thereby increasing the reliability of the memory system overall. For future work, we plan to experiment with a broader range of benchmarks and determine how utilization, number of banks, and different *psleep* thresholds can affect the overall aging in terms of static noise margin of the SPM.

References

[1] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits," *Prooceding of the IEEE*, vol. 91, pp. 305–327, 2003.

[2] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware computer systems: Opportunities and challenges," *IEEE Micro*, vol. 23, pp. 52–61, November 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=1435619.1436215

[3] K. Banerjee and A. Mehrotra, "Global (interconnect) warming," *IEEE Circuits and Devices Magazine*, vol. 17, pp. 16–32, 2001.

[4] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings of the 2000 international symposium on Low power electronics and design*, ser. ISLPED '00, 2000, pp. 90–95. [Online]. Available: http://doi.acm.org/10.1145/344166.344526

[5] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings of the 28th annual international symposium on Computer architecture*, ser. ISCA '01, 2001, pp. 240–251. [Online]. Available: http://doi.acm.org/10.1145/379240.379268

[6] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[7] D. S. Boning and S. Nassif, "Models of process variations in device and interconnect," in *Design of High Performance Microprocessor Circuits, chapter 6*. IEEE Press, 1999.

[8] M. Alam, "Reliability- and process-variation aware design of integrated circuits," *Microelectronics Reliability*, vol. 48, no. 8, pp. 1114–1122, 2008.

[9] G. Chen, M. Li, C. Ang, J. Zheng, and D. Kwong, "Dynamic nbti of p-mos transistors and its impact on mosfet scaling," *IEEE Electron Device Letters*, vol. 23, no. 12, pp. 734–736, 2002.

[10] S. Kumar, K. Kim, and S. Sapatnekar, "Impact of nbti on sram read stability and design for reliability," in *ISQED-06: International Symposium on Quality Electronic Design*, 2006, pp. 213–218.

[11] K. Kang, M. Alam, and K. Roy, "Characterization of NBTI induced temporal performance degradation in nano-scale SRAM array using IDDQ," in *ITC-07: International Test Conference*, 2007, pp. 1–10.

[12] V. Huard, C. Parthasarathy, C. Guerin, T. Valentin, E. Pion, M. M. N. Planes, and L. Camus, "NBTI degradation: from transistor to SRAM arrays," in *IEEE 46th Annual International Reliability Physics Symposium*, 2008, pp. 289–300.

[13] A. Calimera, M. Loghi, E. Macii, and M. Poncino, "Aging effects of leakage optimizations for caches," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, ser. GLSVLSI '10, 2010, pp. 95–98. [Online]. Available: http://doi.acm.org/10.1145/1785481.1785504

[14] A. Ricketts, J. Singh, K. Ramakrishnan, N. Vijaykrishnan, and D. K. Pradhan, "Investigating the impact of nbti on different power saving cache strategies," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, 2010, pp. 592–597. [Online]. Available: http://portal.acm.org/citation.cfm?id=1870926.1871065

[15] A. Calimera, E. Macii, and M. Poncino, "NBTI-aware power gating for concurrent leakage and aging optimization," in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '09, 2009, pp. 127–132. [Online]. Available: http://doi.acm.org/10.1145/1594233.1594264

[16] A. Calimera, M. Loghi, E. Macii, and M. Poncino, "Dynamic indexing: concurrent leakage and aging optimization for caches," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '10, 2010, pp. 343–348. [Online]. Available: http://doi.acm.org/10.1145/1840845.1840916

[17] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu, "Banked scratch-pad memory management for reducing leakage energy consumption," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, ser. ICCAD '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 120–124. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2004.1382555

[18] C.-L. Su and A. M. Despain, "Cache designs for energy efficiency," in *Proceedings of the 28th Hawaii International Conference on System Sciences*, 1995, pp. 306–. [Online]. Available: http://portal.acm.org/citation.cfm?id=795694.798059

[19] Freescale-QE, "Freescale low-power QE family processor," http://www.freescale.com/files/microcontrollers/.

[20] STMicroelectronics-Cortex, "STMicroelectronics Cortex-M3 CPU," http://www.st.com/mcu/inchtml-pages-stm32.html.