A Hybrid HW-SW Approach for Intermittent Error Mitigation in Streaming-Based Embedded Systems

Mohamed M. Sabry[†], David Atienza[†], and Francky Catthoor[‡]
† Embedded Systems Lab (ESL), Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
‡ imec, Leuven, Belgium.
email: {mohamed.sabry, david.atienza}@epfl.ch, catthoor@imec.be.

***Abstract*—Recent advances in process technology augment the systems-on-chip (SoCs) functionality per unit area with the substantial decrease of device features. However, features abatement triggers new reliability issues such as the single-event multi-bit upset (SMU) failure rates augmentation. To mitigate these failure rates, we propose a novel error mitigation mechanism that relies on a hybrid HW-SW technique. In our proposal, we enforce SoC SRAMs by implementing a fault-tolerant memory buffer with minimal capacity to ensure error-free operation. We utilize this buffer to temporarily store a portion of the stored data, named a data chunk, that is used to restore another data chunk in a fully demand-driven way, in case the latter is faulty. We formulate the buffer and data chunk size selection as an optimization problem that targets energy overhead minimization, given that timing and area overheads are restricted with hard constraints decided beforehand by the system designers. We show that our proposed mitigation scheme achieves full error mitigation in a real SoC platform with an average of 10.1% energy overhead with respect to a base-line system operation, while guaranteeing all the design-time constraints.**

I. Introduction and Related Work

Future processing technologies permit the increase in systems functional complexity by integrating more transistors within the same unit area. However, with CMOS scaling, different reliability issues, such as Negative Bias Temperature Instability (NBTI) [1], have become major challenges in robust systems design and operation. To overcome the recent reliability challenges that result in different error types such as soft errors and wear-out [2], it is crucial to design robust systems that reduce, and eventually eliminate, the high failure rates at a reasonable cost (area, energy, and performance).

Single-event single-bit upsets (SSUs) have been a dominant cause of error in on-chip SRAMs (e.g. SPMs). To alleviate SSUs, error correction codes (ECCs) have been widely used in different memory levels [3] as ECCs provide single-bit correction with feasible area, energy and timing overhead. For example, previous work [4] shows that single-error-correction double-error-detection (SECDED) ECC adds 15% area overhead when used to protect L1 SRAMs.

As technology scaling increases, single-event multi-bit upset (SMU) rate increases significantly [5], debilitating the SECDED ECC mitigation capability. Although multi-bit ECC circuits can be used to mitigate SMU resulting errors, multi-bit ECC circuitry demands significant area, energy and timing overheads. These overheads can be feasible in high-capacity memories (e.g. L2) [6]. However, these overheads are unacceptable, from an industrial perspective, for low capacity embedded SRAMs (e.g. 64KB). For example, the area overhead of an 8-bit ECC integrated to a 64KB SRAM is reported to be more than 80% [7].

Other HW-based approaches may consider module redundancy. For example, May et. al [8] use resource duplication and triplication in reliability-aware design of a *low density parity-check (LDPC)* code decoder. The amount of resource redundancy is based on the protection priority of the corresponding resource. The controller and functional units in the LDPC are triplicated, while only the most significant byte of the SRAM memory is duplicated (or triplicated), to withstand a small mean-time-between-failures (MTBF) value. However, this approach has a significant area overhead that is also accompanied by a substantial increase in the leakage power.

Since HW-based error mitigation techniques are cost inefficient, SW-based mitigation techniques are more promising in this situation especially when applied in a fully demand-driven way. However, soft mitigation techniques are dependent on the target applications, and may imply significant timing and energy overheads. Thus, the quality-of-service (QoS) of the application is degraded significantly. For example, Gupta et al. [9] propose a delayed commit and rollback mechanism to overcome soft errors. The authors rely their solution on a violation detector that has a time lag (D). If a data element is correct for a time period D and no violation is detected, it is considered correct. Otherwise, it is considered faulty and a rollback is performed. Although this approach seems interesting, this approach has a performance loss that reaches 18%. Another proposal uses checkpoints and rollback or forward error drop for error mitigation [10]. This work uses a partially protected cache (PPC) to store a portion of the streaming data, such that it is used to recover from an error in the unprotected cache along with checkpoints. However, in this work the size and impact of the protected memory word is not considered on the overall system cost.

In this paper we propose a hybrid, fully demand-driven, HW-SW error mitigation mechanism to overcome the increased SMU error rates. In our proposal, we enforce the

This research is partially funded by the Nano-Tera RTD project CMOSAIC (ref.123618), financed by the Swiss Confederation and scientifically evaluated by SNSF, the PRO3D EU FP7-ICT-248776 project, and a long term PhD grant from imec.

system-on-chip (SoC) SRAMs by implementing a fault-tolerant memory buffer with minimal capacity to ensure error-free operation, taking into account the processing nature of the target applications. This small memory buffer is implemented with a multi-bit ECC that, in this case, has a negligible area overhead to the system due to the minimal buffer capacity. Then, we use this fault-tolerant buffer to temporarily store a data segment, named a data chunk, that is used to restore another corrupted data chunk. We formulate the buffer size selection as an optimization problem to minimize the energy overhead, given that the time and area overheads are restricted with hard constraints decided beforehand by the system designers. We examine this proposal on a low-power embedded system running streaming applications as case studies. Simulation results show that we can achieve full error mitigation within the hard time and area constraints, with maximum 22% and average 10.1% energy overheads with respect to a base-line system operation, while guaranteeing all the design-time constraints. On the contrary, conventional approaches that guarantee the reliable operation with strict hardware, or software, solutions require an energy overhead of more than 100%.

II. Proposed Mitigation Scheme

In the proposed mitigation mechanism, three concepts are combined. Thus, we define them to prevent any possible ambiguity as follows:

- **Checkpoint.** It is a software-based trigger ($CH(i)$) that indicates the termination of certain computation phase in time and the commencement of another computation phase.
- **Data chunk.** It is the data segment ($D_{CH}(i)$) that is generated in computation phase(i) and/or should be alive between two consecutive computation phases (e.g. flow control registers, intermediate streaming data,...).
- **Rollback.** It is the process of restarting the system operation from the last successfully committed checkpoint.

Our proposed methodology relies primarily on the insertion of a number of periodic checkpoints N_{ch} within a task execution. At each checkpoint $CH(i)|i \in [1, N_{ch}]$, a data chunk is stored in a protected memory buffer that we integrate to the system. We refer to this buffer with $L1'$. When checkpoint $CH(i)$ is being committed, $D_{CH}(i)$ is buffered to $L1'$ to overwrite $D_{CH}(i-1)$ while the task is being executed. However, if $D_{CH}(i)$ is faulty, it is regenerated using the error-free $D_{CH}(i-1)$, starting from $CH(i-1)$.

For illustration purpose, Fig. 1 shows an example of dividing a computational task T_1 into 5 phases $P_i, i \in [1,5]$. After each phase P_i, L cycles are elapsed to check for error. If the data is error free, the data chunk $D_P(i)$ is buffered while executing P_{i+1}. If an error occurs as in P_3, only data chunk $D_P(3)$ is re-computed. Thus, the deadline violation previously occurred due to the introduced intermittent error is avoided in this case.

This checkpoints inclusion and data division into chunks imply timing and energy overheads. First, the data chunk

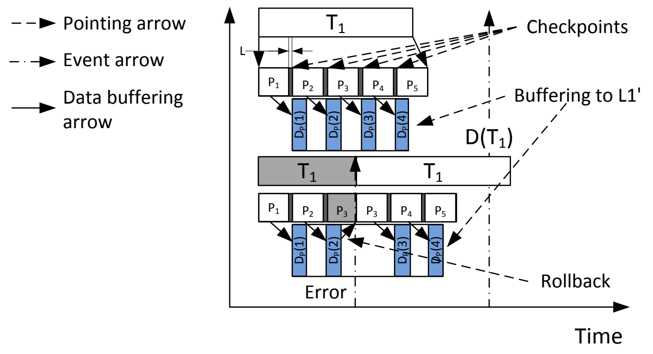


Fig. 1. An example of dividing the data of T_1 into 5 chunks, showing the impact on intermittent error mitigation.

produced at a certain phase ($CH(i)$) consumes additional energy to be stored for a possible upcoming mitigation at the consequent phase ($CH(i+1)$). Second, checkpoints at the end of each computation phase add additional time and energy overheads to the overall execution time and energy consumption. Finally, if an error occurs, the system is penalized additional time and energy to re-compute a faulty data chunk. In order to make our proposal is energy, time, area efficient, an optimum chunk size, as well as optimum number of checkpoints, must be selected to minimize the aforementioned overheads.

A. Chunk size selection

To select the optimum chunk size, and optimum number of checkpoints, for a guaranteed error-free operation, we quantitatively identify the overhead costs. We identify this cost into storage and computation costs, such that the storage cost (C_{store}) is the cost introduced due to storing each data chunk in the introduced buffer at each checkpoint ($CH(i)$). The computation cost (C_{comp}) is the cost resulted from two items; the triggering of a checkpoint and the re-computation of a data chunk to mitigate the runtime error. We use the energy consumption to quantify both storage C_{store}, and the computation C_{comp} costs. We compute the storage energy cost as follows:

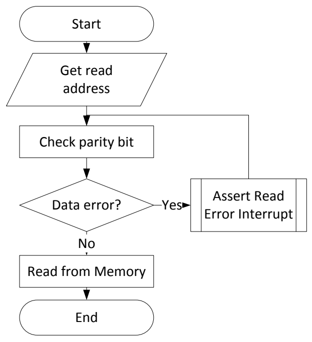
$$C_{store} = (N_{CH} \cdot S_{CH} + err) \cdot E(S_{CH}) \qquad (1)$$

where N_{CH} is the number of checkpoints, S_{CH} is the chunk size (in bytes), err is the expected number of chunks that will be faulty within a running task, and $E(S_{CH})$ is the consumed energy in accessing the buffer of size S_{CH}. We define the computing energy cost as follows:

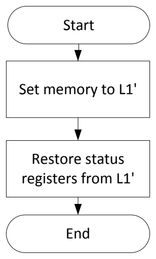
$$C_{comp} = N_{CH} \cdot E_{CH} + err \cdot (E_{ISR} + E(F(S_{CH}))) \qquad (2)$$

where E_{CH} is the additional energy consumed at each checkpoint, E_{ISR} is the energy consumed by the mitigation routine triggered when an error occurs, and $E(F(S_{CH}))$ is the energy consumed to re-compute a data chunk.

In our optimum chunk size selection, the additional overheads in terms of storage and computation cycle must be kept



(a) System read transaction



(b) Interrupt service routine

Fig. 2. System operation flow when a memory read is issued to apply our mitigation proposal.

within acceptable ranges. Thus, in our problem formulation, we present the required overheads as inequality constraints to guarantee that the area overhead of the optimum buffer size implementation $A(S_{CH})$ is less than the affordable area overhead in the system, while the cycle overhead required for error mitigation $D(S_{CH})$ is maintained within the allowed cycle overhead. We formulate the chunk size S_{CH} and number of checkpoints N_{CH} optimization as follows:

$$\begin{aligned} \min_{S_{CH},N_{CH}} \quad J &= C_{store} + C_{comp} && (3)\\ Subject \;\; to: & && \\ A(S_{CH}) &\leq OV_1 \cdot M && (4)\\ D(S_{CH}) &\leq OV_2 \cdot S_{CH} && (5)\\ S_{CH} &= K \cdot W_{size} && (6)\\ N_{CH},\; K &\in \mathbb{Z}^+ && (7) \end{aligned}$$

where S_M is the total size of the system storage.

B. Hybrid demand-driven mitigation implementation

As mentioned earlier, our proposal involves a hybrid HW-SW mitigation mechanism that adds to the system additional HW modules and SW routines. In our proposal, we integrate an additional intermediate memory storage layer, namely $L1'$, between the $L1$ and the processing unit. $L1'$ has an a multi-bit ECC circuitry that, due to the small $L1'$ capacity, has extremely limited size. $L1'$ is used to buffer the chunk(s) at checkpoint $CH(i)$, that are essential and sufficient to mitigate an error occurred between checkpoints $CH(i)$ and $CH(i+1)$.

Our mitigation mechanism is activated at every faulty memory read, as shown in Fig. 2(a). When a read command is issued within checkpoints $CH(i)$ and $CH(i+1)$, the read memory word is checked for error. If it is faulty, an interrupt titled *Read Error Interrupt* is triggered, that rollbacks the system to checkpoint $CH(i)$. At each checkpoint, the status registers as well as the data chunk (if it is error free) are stored in $L1'$, such that they can be used later in case of a read failure between the this checkpoint and the subsequent one.

The flow of *Read Error Interrupt* service routine is briefly shown in Fig. 2(b). First, the system replaces the stored status registers due to context switching (a cause of the interrupt) with their values stored at the last checkpoint. Then, the routine enables accessibility to $L1'$ to read the protected chunk. It is worth mentioning that in some processor designs, a pipeline flush is required, to stop the executed instructions when an error is detected, for a successful rollback. When the processor restores from this routine, it executes the program segment that starts at the last committed checkpoint.

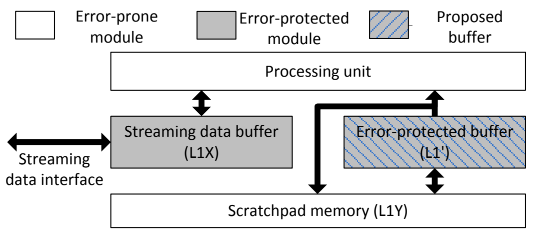


Fig. 3. Schematic diagram of the system used in the aforementioned case-studies.

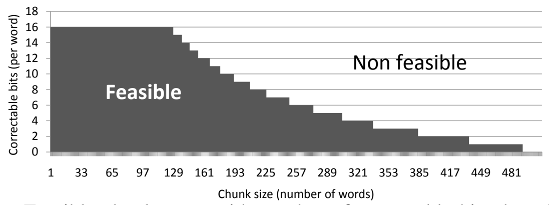


Fig. 4. Feasible chunk areas with number of correctable bits, based on the 5% area overhead.

III. Experimental Results

In our evaluation, we use streaming applications that represent typical periodic tasks. In particular, we deploy several selected applications from the MediaBench benchmark suite [11].

A. Target system architecture and constraints

We run the aforementioned applications on a single-core NXP SoC platform [12]. The targeted platform is based on the 32-bit ARM9 processor, which operates at a maximum frequency of 250 MHz. In our experiments, we fix the operating frequency to 200 MHz. A schematic diagram of the target system with the proposed L1' buffer integrated is shown in Fig. 3. This system has a 64KB L1 SRAM, which we select as our targeted vulnerable memory. We use CACTI 6.5 [13] to estimate the L1 SRAM and $L1'$ area, energy, and access time using $65nm$ process technology.

In our evaluation, we select the affordable area overhead (OV_1) as 5%, which is the maximal affordable area overhead provided by our industrial partners [12]. This overhead restricts the storage capacity of $L1'$, as well as the maximum number of correctable bits per word, as shown in Fig. 4. This figure shows the feasible $L1'$ size values with different error correcting capabilities, which we use in our optimization problem.

Based on the used case studies and the target platform characterizations, we select the affordable cycle overhead (OV_2) as 10%. In our experimentations, we use an error rate

TABLE I
OPTIMUM CHUNK SIZE OBTAINED FOR DIFFERENT BENCHMARKS

Benchmark	Optimum protected buffer size (words)	Benchmark	Optimum protected buffer size (words)
ADPCM encode	11	ADPCM decode	11
G721 encode	16	G721 decode	32
JPG decode	44		

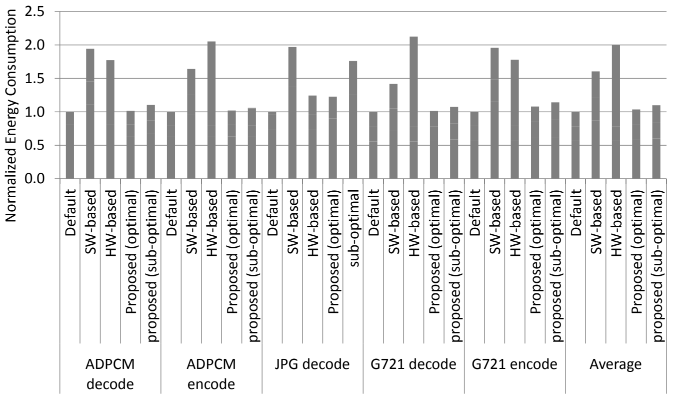


Fig. 5. Normalized energy consumption with respect to the default case for different benchmarks.

of 10^{-6} word per cycle, which is an upper bound comparable to the rate values mentioned in previous work [14]. Thus, this is the worst-case situation to consider.

B. Energy consumption and performance

We compare our results with three different cases: (1) **Default** case, where the system is operating with no error mitigation; (2) **HW-mitigation** case where the targeted L1 is fully protected, but at the cost of a (too) large area overhead; and (3) **SW-mitigation** case where the memory has minimal ECC capability, while the mitigation is performed by task restarting. Moreover, we show the results of our proposed methodology with the optimal chunk size and number of checkpoints, and with sub-optimal values as well.

Based on the problem formulation mentioned in Subsection II-A and the conditions mentioned above, we use the MATLAB optimization toolbox, to get the optimum chunk size and number of checkpoints. Table I shows the obtained optimum chunk sizes for the used benchmarks.

In our exploration, we perform our simulations using MPARM [15], which is a cycle-accurate SystemC-based multiprocessor simulator. MPARM is capable of generating accurate timing waveforms, and good energy consumption estimations of different modules. We set the simulation parameters in MPARM (simulated architecture, memory size, operating frequency,...) to simulate our targeted system (c.f. Subsection III-A).

Fig. 5 shows the energy consumption of the target system with different benchmarks and the aforementioned mitigation techniques. This figure shows that our proposed methodology manages to meet the area and timing constraints, while maintaining an error-free processing with a maximum of 22% energy overhead, with respect to the default case. On average, we observe that the energy overhead is 10.1%. However, the energy overhead of both *HW-mitigation* and *SW-mitigation* are, on average, more than 70% higher than the *Default* case. The maximum energy overhead of both *HW-mitigation* and *SW-mitigation* is more than 100%. This is due to the huge energy overhead introduced to fully protect the L1 SRAM in *HW-mitigation* case, and the substantial increased accessibility to the L1 SRAM to mitigate the errors by *SW-mitigation*.

We observe the execution time of different mitigation scenarios, when running various benchmarks. Our mitigation scheme manages to maintain the execution time overhead constraint we provide at design-time. On the contrary, the observed overheads in *HW-mitigation* and *SW-mitigation* exceeds the timing constraints by values reaching up to 100%.

IV. Conclusion

We have proposed in this paper a novel error mitigation mechanism that relies on a hybrid HW-SW mechanism. We enforce the error-prone on-chip SRAMs with a fault-tolerant memory buffer with minimal capacity to ensure error free operation. We utilize this buffer to temporarily store a data chunk, that can be used to restore another data chunk, in case the latter is faulty. We optimize the data chunk size to minimize the energy overhead, subject to system constraints that are decided beforehand by the system designers. We show that our proposal achieves full error mitigation with only 10.1% average energy overhead (and 22% overhead in the worst case) with respect to a baseline system operation, while guaranteeing all the design-time constraints.

References

[1] M. Agostinelli et al. Random Charge Effects for PMOS NBTI in Ultra-Small Gate Area Devices. In *IRPS'05*, 2005.
[2] S. Mitra. Globally Optimized Robust Systems to Overcome Scaled CMOS Reliability Challenges. In *DATE'08*, 2008.
[3] P. Kongetira et al. Niagara: a 32-Way Multithreaded SPARC Processor. *IEEE Micro*, 25, 2005.
[4] S. S. Pyo et al. 45nm Low-Power Embedded Pseudo-SRAM with ECC-Based Auto-Adjusted Self-Refresh Scheme. In *ISCAS'09*, 2009.
[5] E. Ibe et al. Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule. *IEEE Transactions on Electron Devices*, 57, 2010.
[6] S. Paul et al. Reliability-Driven ECC Allocation for Multiple Bit Error Resilience in Processor Cache. *IEEE Transactions on Computers*, 60, 2011.
[7] J. Kim et al. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *MICRO-40*, 2008.
[8] M. May et al. A Case Study in Reliability-Aware Design: A Resilient LDPC Code Decoder. In *DATE'08*, 2008.
[9] M. S. Gupta et al. DeCoR: A Delayed Commit and Rollback Mechanism for Handling Inductive Noise in Processors. In *HPCA'08*, 2008.
[10] K. Lee et al. Mitigating the Impact of Hardware Defect on Multimedia Application - A Cross-Layer Approach. In *MM'08*, 2008.
[11] Ch. Lee et al. MediaBench: a Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *MICRO'97*, 1997.
[12] NXP ARM-Based Microntrollers. http://www.nxp.com/documents/data_sheet/LH7A400_N.pdf.
[13] CACTI: an Integrated Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model for Cache Architectures. http://www.cs.utah.edu/ rajeev/cacti6/.
[14] L. Leem et al. ERSA: Error Resilient System Architecture For Probabilistic Applications. In *DATE'10*, 2010.
[15] L. Benini et al. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *Journal of VLSI Signal Processing Systems*, 41(2), 2005.