Journal of
Low Power Electronics
Vol. 15, 30–39, 2019

Full System Emulation of Approximate Memory Platforms with AppropinQuo

Giulia Stazi*, Antonio Mastrandrea, Mauro Olivieri, and Francesco Menichelli

Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome, Rome, 00184, Italy

(Received: 15 December 2018; Accepted: 10 February 2019)

In this work we present an emulation framework for hardware platforms provided with approximate memory units, called AppropinQuo. The specific characteristic of AppropinQuo is to reveal the effects, on the hardware platform and on software, of errors introduced by approximate memory circuits and architectures. The emulator allows to execute software code without any modification with respect to the target physical board, since it includes the CPU, the memory hierarchy and the peripherals, capturing as well software–hardware interactions and faults due to approximate memory units. The final scope is reproducing the effects of errors generated by approximate memory circuits, allowing to evaluate the impact (quality degradation) on the output produced by the software. In fact, output quality is related to error rate, but their relationship strongly depends on the application, the implementation and its data representation on physical memory. The idea behind approximate memory circuits and approximate computing in general is to trade off energy consumption at the expense of computational accuracy and degradation of output quality. Memory is accounted for a large part of total power consumption in advanced architectures and it is supposed to increase as new memory hungry applications migrate toward the implementation on embedded systems (embedded machine learning, high definition video codecs, etc.). By relaxing design constraints regarding error probability on bit cells, researchers have proposed techniques that significantly reduce memory energy consumption. These techniques, which can be accounted in the general topic of approximate memory design, are implemented at circuit or architecture level, and are specific to the memory technology (i.e., SRAM or DRAM memories). However, the level of acceptable output degradation is the final metric that must be used to assess if, and to what extent, an approximate memory technique can be introduced. Our emulator allows to run actual applications as on the physical platform, to expose the effects of specific approximate memory circuits and architectures on output quality and to vary their parameters (e.g., error rate, number of affected bits, etc.). By exploring the approximate memory design space and its effects on the output of a software application, it is possible to characterize the application behavior, as a step toward the determination of the trade-off between saved energy and output quality (energy-quality tradeoff).

Keywords: Approximate Memory, Full System Emulation, Design Space Exploration.

1. INTRODUCTION

In modern digital systems memory represents a significant contribution to overall system consumption.[1] This is mainly going in parallel with the increasing performance of computing platforms, which put under stress memory bandwidth and capacity. Applications as deep learning, high definition multimedia, 3D graphic, contribute to the demand for such systems, with added constraints on energy consumption.

Techniques for reducing energy consumption in SRAM and DRAM memories have been proposed in many works. A class of very promising approaches, generally called approximate memory techniques, is based on allowing controlled occurrence of errors in memory cells.[2]

The effective introduction of approximate memory units in a hardware platform relies on the possibility of using them for allocating selected data structures in software applications. Although memory errors may degrade the quality of output results, the effects can be tolerated by applications known as ETAs (error tolerant applications). The output degradation can be measured in different ways, depending on the specific output (e.g., SNR in case of

*Author to whom correspondence should be addressed.
Email: g.stazi@uniroma1.it

 1546-1998/2019/15/030/010 doi:10.1166/jolpe.2019.1595

a digital signal processing application), however its relationship with respect to approximate memory parameters (i.e., architectures, error rate, number and weight of affected bits) is not straightforward since it depends on many factors as the kind of application, the implementation details and how data are represented in memory by the compiler. The result is that it is possible to characterize the behavior of a complex, real world application with respect to the presence of errors introduced by approximate memories only by executing it on the target hardware platform (including approximate memories), or on an emulator that can model the complete platform.

1.1. Approximate Memories

Approximate memories are memory circuits where cells are subject to hardware errors (bit flips) with controlled probability. In a wide sense the behavior of these circuits is not different from standard memories, since both are affected by errors. However, in approximate memories the occurrence of errors is allowed by design and traded off to reduce power consumption. The important difference between approximate and standard memory reside in the order of magnitude of error rate and its consequences: in approximate memories errors become frequent and are not negligible to software applications.

1.2. Error Tolerant Applications

Error Tolerant Applications (ETAs) are defined as applications that can accept a certain amount of errors during computation without impacting the validity of their output. Multimedia applications are an important class of ETAs: they process data that are already affected by errors/noise and, due to limitations of human senses, can introduce approximations on their outputs (e.g., lossy compression algorithms). Moreover, they tend to require large amounts of memory for storing their buffers and data structures. Another example of ETAs is represented by closed loop control applications; again, they process data that are affected by noise (sensor reading) and produce outputs to actuators affected by physical tolerances and inaccuracies.

2. RELATED WORKS AND CONTRIBUTION

2.1. Approximate Memory Circuits and Architectures

Approximate memory circuits have been proposed in research papers since some years. By relaxing requirements on data retention and faults, many circuits and architectures have demonstrated to significantly reduce power consumption, for both SRAM and DRAM technologies.

Considering SRAMs, in Refs. [3, 4] the authors propose memory circuits where voltage scaling is applied to bit cells. Along with voltage scaling, multiple bit-level techniques are proposed that enable dynamic management of the energy-quality tradeoff. Examples of this techniques are multiple V_{DD}, bit-dropping, selective negative bitline boosting. Results on some application (i.e., H.264 hardware video decoder) were produced by modeling the architecture in Matlab and the SRAM bitcell failures were injected according to the measures on test chips. In Ref. [5] a dynamically reconfigurable SRAM array is suggested. The proposed solution uses a dual voltage architecture where nominal voltage is applied to cells storing higher order bits while a reduced voltage is applied to cells storing low-order bits. The number of bits with under voltage power supply is reconfigurable at run-time to change the error characteristic. Results are produced by storing static images in the array of cells and measuring the quality degradation induced by injected bit-flips.

In Ref. [6] SRAM approximate caches are explored. The work focuses on relaxing the guard-bands required for masking the effects of manufacturing variations as a way of reducing leakage energy of SRAM caches. Errors are allowed by exploiting the tolerance of specific applications and energy savings up to 74% are claimed. Results where produced using the gem5 simulator[7] by modifying the cache architecture, however the details of the models are not given.

As regards DRAMs, different techniques to reduce refresh rate have been proposed.[1,8–11] In fact, in exact DRAMs refresh time interval is set according to the worst case access-statistics of the most leaky cells. Commercial DRAM modules, for example, have a worst case retention time of 64 ms determined by the leakiest cells in the entire array. This high refresh rate, which guarantees a storage without errors at the expense of power consumption, could not be required in some applications.

In Ref. [1] the authors propose to partition DRAM rows and apply different refresh rates; rows containing leaky cells are refreshed at normal rate, while most rows are refreshed with reduced rate. In Ref. [8] the idea of partitioning critical and non-critical data is explored; an application-level technique named *Flikker* enables software developers to specify critical and non-critical data in programs, that are then allocated in separate parts of memory. Again, regular refresh rate is applied to the portion of memory containing critical data, while the portion containing non-critical data is refreshed at lower rates.

In Ref. [9] the authors abandon the worst case design paradigm, showing the benefits that can be achieved by relaxing refresh time interval at the expense of increasing error rates. In particular, tests on 8 chips of GC-eDRAMS show that, admitting an error rate of 10^{-3} and relaxing refresh rate from 11 ms (worst case retention time) to 24 ms, 55% of energy can be saved; while an error rate of 10^{-2} guarantees energy savings up to 75%. In Ref. [10], after an experimental characterization of memory errors as a function of the DRAM refresh-rate, the authors propose a methodology for constructing a quality configurable approximate DRAM system. Experiments were performed on a FPGA board where a soft-processor (Nios II) and

a DDR3 memory controller (UniPHY) are programmed; they revealed a reduction in DRAM refresh power of up to 73% on average.

The specific use of approximate DRAM architectures is studied in Ref. [11] for deep learning applications, since they are tolerant to the presence of data errors. In particular, DRAM organization is modified to support the control of the refresh rate according to the significance of stored data. Simulations were performed injecting errors at algorithm level, using random bit flips with uniform distribution, without reference to the internal structure of DRAM cells (see Section 3.1).

2.2. Simulation Environments for Digital Platforms

Simulation environments for embedded system platforms are considered fundamental tools to support the design and optimization flow, because the typical hardware/software interactions that are present in this field of application make the debug process, the characterization of performances and the optimization difficult. These tools allow a complete emulation of the physical platform, including instruction set architecture (ISA) emulation and hardware units emulation (e.g., memory, I/O units, timers, etc.).

They are needed, among others, during the first design phases since they provide the ability to explore different architectures and ideas, allowing to collect data regarding functionality, performance, energy consumption, with reduced costs and time compared to physical prototypes. This is true for functional and performance emulators,[7,12–14] energy consumption emulators,[15,16] faults emulators.[17,18]

In Ref. [19] a simulation environment for the investigation on approximate DRAMs is presented. The simulator uses the gem5[7] framework as functional simulator and includes DRAMSys and DRAMPower.[13,16] Thanks to the addition of DRAMSys and DRAMPower, the emulation environment is capable of modeling data retention starting from memory physical parameters and producing at run-time power consumption data. Presented results demonstrate that, for the two cases taken as case study, refresh rate can be completely disabled with a negligible degradation on output quality. However, the tool is focused on technology parameters and variations, it is limited to DRAM models and no details are added with respect to the support for approximate techniques that rely on bit-weights or in general on architectural level techniques.

The contribution of the work described in this paper is to fill the gap in hardware platform emulators, adding specific support for approximate memories, showing the valuable information that can be obtained. Even if emulators including faults in memory units have already been developed, the fault models are not specific to the area of approximate memories and many effects of approximate memory circuits and architectures cannot be emulated using general fault models. Our models include the ability of emulating the effects of different approximate designs and implementations, which depend on the internal structure and organization of memory cells.

As discussed in Section 2.1, previous works on approximate memory tend to use ad-hoc or limited solutions when results are produced, injecting bit flips at algorithmic level (without emulating the actual hardware platform) or using restricted modifications on existing emulators. While injecting bit flips at algorithmic level can be quite efficient in terms of emulation speed, if the hardware platform is not emulated all results that depend on the actual data allocation and implementation can be lost. This is particularly true for relatively advanced approximate memory strategies, as those using bit-weights to calibrate fault rates. Moreover, faults that depend on memory accesses (as typical for SRAMs, see Section 3.2) cannot be correctly emulated if the real memory access patterns are not reproduced.

Our emulator, based on QEmu[12] and supporting ×86 and ARM based platforms, allows to run real applications and operating system, to analyze application behavior and to expose the effects on output quality of different memory error rates and approximate techniques. By exploring the design space and its effects on output, a complete characterization of the application is possible, allowing the determination of the trade-off between the level of approximation and output quality. With respect to the work presented in Ref. [20] this paper extends the implementation in different directions: (a) a complete implementation for modeling approximate SRAM is presented; (b) DRAM models for approximate memories are added; (c) bit dropping models for SRAM and DRAM are added.

3. ERROR INJECTION MODELS FOR APPROXIMATE MEMORIES

AppropinQuo is an extension of QEmu,[12] which is a generic and open source emulator of complete hardware architectures capable of running different operating systems. In AppropinQuo we developed specific units to model approximate memories inside the architecture; in particular the approximate memory units are implemented as QEmu *MemoryRegion*s, mapped in the I/O memory space, that receive faults according to the error injection models.

The target memory subsystem is emulated as an acyclic graph of QEmu *MemoryRegion* objects; in particular the address space of the guest memory is represented by the *system_memory MemoryRegion*, from which SRAM and DRAM memories are allocated as single *MemoryRegion* objects. To emulate approximate memory inside the target platform we implemented a new *MemoryRegion* object, called *approx_mem*, and added it to the *system_memory* container. Every time the *approx_mem* object is allocated, a specific init function is called in order to initialize the approximate memory region. In this phase the configurable

parameters of the error injection models, such as the kind of faults, fault rates, errors on access and other that will be described in the following sections, are parsed from a configuration file, passed to the emulator and used to define the memory map layout, which depends on the target architecture. Effectively, the *approx_mem* region is mapped in the I/O memory space and at run time all read-write accesses are intercepted. This is required since some faults are generated during memory access operations (see, for example, Section 3.2).

The fault injection models the we currently implemented take into account the techniques and the circuit implementations proposed in present research literature for approximate DRAMs and SRAMs. As an example, reducing refresh rate in DRAMs and allowing errors due to cell leakage is a strategy for reducing power consumption that has produced many research works in the field of approximate memories.[1,10,19] The effects of these techniques at bit level can vary depending on DRAM architectures and will be discussed in Section 3.1. As regards SRAMs, the techniques are more varied and applied at circuit level.[3,4,21] In general, energy reduction is obtained by scaling supply voltage, but the effects at bit level can be different, depending on circuit design. They are described in Section 3.2.

Along with technology dependent approximate memory circuits, some techniques specifically proposed for approximate memories work at architectural or logic level and can be applied, to some extent, regardless of the technology. They will be discussed in Section 3.3 and Section 3.4.

3.1. DRAM Orientation Dependent Models

DRAM memory cells use a single transistor and a single capacitor to store a bit, represented as charge on the capacitor. Lowering the refresh rate of DRAMs determines that the charge loss induced by leakage current will proceed until discharge. The effects on bit value depend on the DRAM circuit architecture, and will be discussed briefly.

In DRAM, single cells are organized in arrays (memory banks) and are connected to an equalizer and a sense amplifier (Fig. 1). Being differential, every sense amplifier is connected to two bitlines in order to determine whether the charge of one of them should be interpreted as logical 0 or 1: when a bitline is activated, the other holds the reference precharge voltage ($V_{DD}/2$). The sense amplifier architecture, which is a specific manufacturer design choice, determines the DRAM cells orientation. In particular the following implementations exist:

- *true-cells*: cells store a logical value of '1' as V_{DD} and a logical value of '0' as 0 V;
- *anti-cells*: cells store a logical value of '0' as V_{DD} and a logical value of '1' as 0 V;
- *mixed-cells*: a combination of both true-cells and anti-cells.

When lowering refresh rate, the corresponding charge loss appears, at logic level, as a bit flip, whose orientation depends on the internal DRAM array structure. In particular, the following errors can emerge and are implemented in our model:

- *true-cell error model*: when a cell loses charge, a '1' to '0' bit flip is observed;
- *anti-cell error model*: when a cell loses charge, a '0' to '1' bit flip is observed;
- *mixed-cell error model*: both '1' to '0' and '0' to '1' bit flip occurs.

The orientation of DRAM cells is specific of the vendor array architecture. The capability of emulating DRAM faults according to its internal structure becomes indeed fundamental to correctly reproduce the effects on the bit cell and consequently on data at software level. In our model, each bit flip direction is characterized by an error probability (fault rates p_01, p_10 are defined, respectively, for the emulation of true-cell and anti-cell error effects). The probability distribution is assumed uniform in the array of cells, as showed in many works (e.g., Ref. [22]), and expressed as an error rate $(errors/(bit \times s))$.

Faults are implemented to occur with a uniform distribution. In particular, after a time interval has elapsed (implemented as a QEmu internal timer and determined by the corresponding error probability), a callback is invoked. Depending on which model is enabled (true-cell

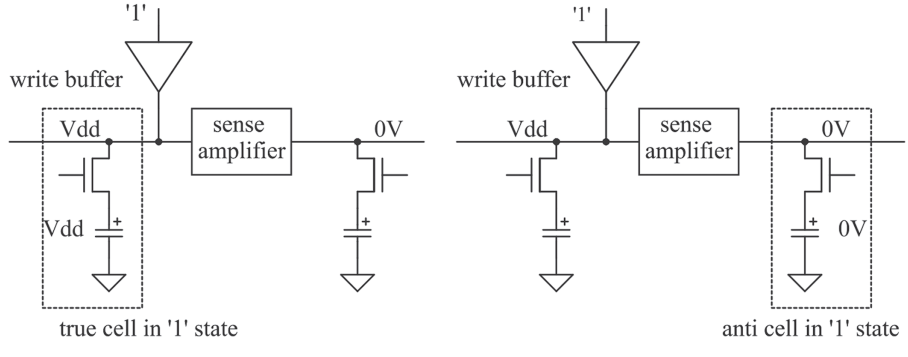

Fig. 1. DRAM true cell and anti cell.
Data Source: Ref. [34].

or anti-cell), the callback generating the corresponding fault is executed, while, in case of a mixed cells architecture, both callbacks are invoked. Since the model is integrated with the *looseness level* model, faults occur only if the bit selected to be flipped has a '1' in the corresponding bit position of the *looseness mask* (see Section 3.4).

3.2. SRAM Models

SRAM approximate memories are designed with aggressive supply voltage scaling. In SRAM bitcells, read and write errors are caused by low read margin (RM) and write margin (WM).[23] Since process variations affect RM and WM in opposite directions, the corner defines which is the critical margin (i.e., the slowfast (SF) corner makes the bitcell write critical, the fastslow corner makes it read critical). Under voltage scaling, WM and RM are degraded, increasing read and write bit error rates (BERs). The degradation is in general abrupt (BER increases exponentially at lower voltages), but techniques have been proposed to make such degradation graceful.[24]

Given this behavior, our fault injection model implements an error on access mechanism, which happens when a cell is activated to perform a read or write operation. Depending on the access we distinguish three kind of errors:

- *Error on write*: introduced during a write operation, the bit stored in the cell is flipped with respect to the bit coming from the data bus;
- *Destructive error on read*: introduced during a read operation, the bit stored in the cell is flipped and passed to the data bus (both cell and data bus contain the corrupted bit);
- *Non-destructive error on read*: introduced during a read operation, the bit stored in the cell is not corrupted during the operation, but it is flipped when passed to the data bus.

For each one of these access errors, a uniform probability distribution in the array of cells is assumed, expressed as *errors/access*.

This fault mechanism is implemented inside the *approx_mem* memory region access callbacks, described in the corresponding QEmu *MemoryRegionOps*. This structure provides two function pointers, *read* and *write*, which process the IO operations on the emulated memory: every time a read or a write operation from the approximate memory region is required, the corresponding callback is invoked. In particular, the fault mechanism related to the destructive and non-destructive error on read is implemented in the read callback while the fault injection mechanism related to error on write is implemented in the write callback. As for the DRAM error orientation models, due to the integration with the *looseness level* model, faults occur only if the bit selected to be flipped has a '1' in the corresponding bit position of the *looseness mask* (see Section 3.4).

3.3. Bit Dropping Fault Model

Bit dropping is a bit-level technique which consists in completely disabling some memory bitlines. The approach showed to be interesting since cells can be completely powered off or even omitted.[4,25,26] The dropped bitlines correspond to a certain number of LSBs in each word, since the impact of errors is exponentially lower for smaller bit weights. This can be paired with the consideration that in many applications, such as machine learning, big data and multimedia, the quality is defined essentially by the MSBs. The technique is independent of technology and can be applied to both SRAM and DRAM memory circuits.[25]

For SRAM memory cells the precharge circuit of the selected LSBs is disabled during read and write operations. This approach is quite different from the traditional dual V_{DD} scheme where the supply voltage of both precharge circuits and bitcells of the selected columns is reduced to a lower value. In Refs. [4, 26] the implementation of a bit dropping precharge circuit is proposed: the drop signal, (corresponding to transistors M1 and M2 in Fig. 2), connects the bitline of the approximated cell to the ground, eliminating the dynamic energy.

In DRAM memory instead, the refresh operation is completely disabled on the dropped bitlines. In Ref. [27] the authors demonstrate that, for dedicated applications, the refresh operation can be switched off with a negligible impact on the application performance.

In our model, bit dropping is implemented as follows:

- in SRAMs, when a word in memory is read or written and the bit dropping is enabled, a given subset of bits is always set to '0'.

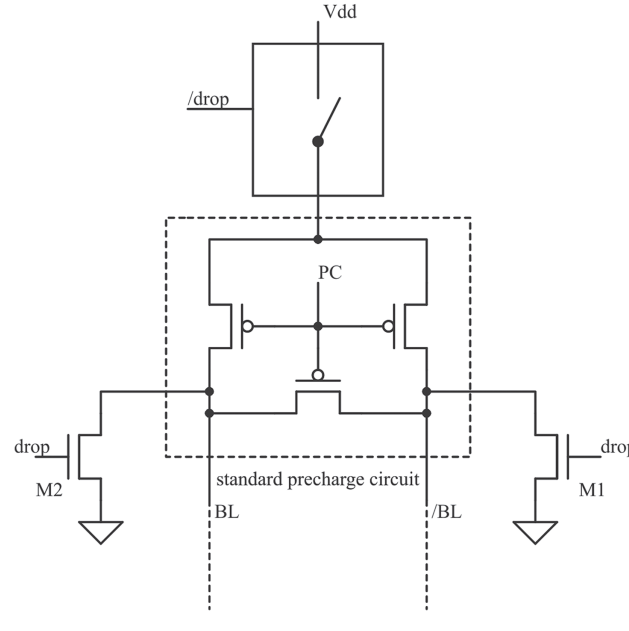

Fig. 2. SRAM precharge circuit for bit-dropping technique. *Data Source*: Ref. [26].

• in DRAMs, when a word in memory is read or written and the bit dropping is enabled, a given subset of bits is set to '0' or '1' depending on memory cell orientation distribution.

This fault injection mechanism is implemented in the callbacks provided by the *approx_mem MemoryRegionOps*: every time a read or write operation is performed in the approximate memory, if the bit dropping error model is enabled, a specified number of bits is always set to '0' or '1', depending on the memory technology. The number of dropped bits is defined by the *looseness mask*, a configurable parameter described in the following section.

3.4. Memory Looseness Level and Fault Models

Bit level approximate techniques have been introduced in order to exploit the exponential weight that bits assume in data words. Effectively, the approach introduces a new level of freedom in the approximate memory design space, that can be explored in search of better trade-offs.

As said in the previous section, a fault in a cell that stores one of the most significant bits (MSBs) has a larger impact on the value, with respect to a fault occurring in one of the least significant bits (LSBs). In Ref. [4] selective voltage scaling is proposed in order to modulate error rate, at the cost of an increase in circuit complexity; in Ref. [28] DRAM banks are reorganized and refresh rate modulated in order to obtain a similar effect. From the results, the technique appears effective but the actual implementation is dependent on the microprocessor ISA and its data representation and organization in memory.

In order to support the emulation of bit level techniques, we have inserted the concept of *looseness level* and *looseness mask* in AppropinQuo. The looseness level (i.e., the number of bit that are affected by errors in a data word) is supported by introducing a 32-bit configurable mask (constant for the whole memory array) that is applied to every 32-bit word in memory (Fig. 3). Its scope is the selection of bits affected by faults (i.e., the MSBs). Bits within a word are not affected by faults when the corresponding bit in the looseness mask is set to zero (i.e., with a looseness mask set to $0 \times 0FFFFFFF$, the 4 MSBs are exact, while the 28 LSBs are affected by faults). The structure of the looseness mask allows to effectively tune approximation at bit level for 32-bit, 16-bit and 8-bit data.

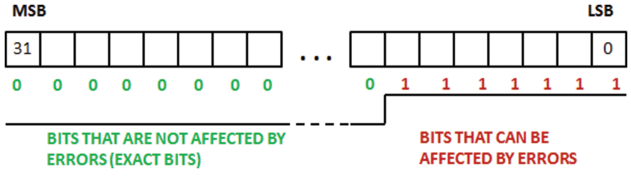

Fig. 3. Example of looseness mask on big Endian architecture.

We underline the fact that having a single looseness mask defined for the whole approximate memory array means having a single looseness level for the array. This is mainly due to the consideration that practical implementations are limited to this configuration, since it is has a large impact in the design of the whole memory layout. Another consideration specific to this model is that MSBs and LSBs in memory are not uniquely defined but depend on microprocessor ISA (i.e., data endianess) and also data size (i.e., in a 32 bit memory word 8 bit data can be packed in groups of four). The choice of defining the memory *looseness level* using a 32 bit *looseness mask* provide the flexibility required by the above mentioned cases.

4. EXPERIMENTAL RESULTS

In order to investigate the impact of approximate memory architectures and configurations and to demonstrate the data that can be obtained with AppropinQuo, in this Section we present two different case studies based on error resilient applications: a digital FIR filter of and audio signal and a H.264 video encoder. They are both implemented in C and executed on a $\times 86$ based system, emulated in AppropinQuo as target architecture.

4.1. Digital FIR Filtering

As first case study, we show the impact of different approximate memory configurations on an audio signal processing application (digital FIR filter). The digital FIR filter consists of 100 taps implemented using 32-bit integer arithmetic. The input buffer, output buffer and internal tap registers have been allocated in approximate memory, for a total of about 200 KByte of memory space.

As quality metric for this application we used output SNR, measured considering noise as the difference between the output of the exact filter and the output of

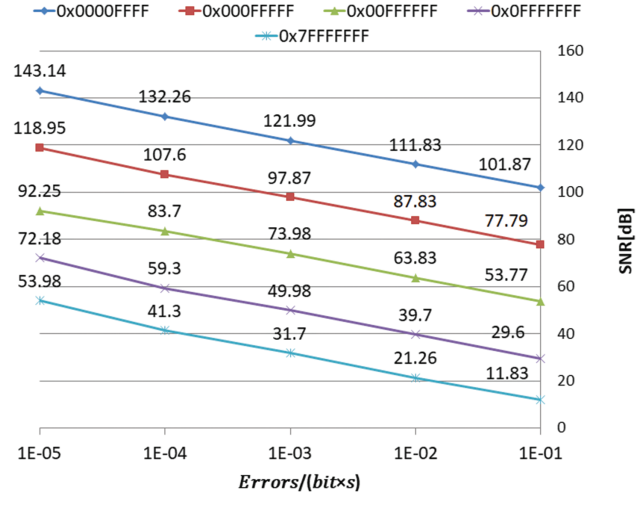

Fig. 4. FIR, output SNR [dB] for approximate DRAM (anti cells).

Table I. FIR, output SNR [dB] for SRAM.

	Looseness level	Fault rate [*errors/access*] 10^{-1}	10^{-2}	10^{-3}	10^{-4}
EOW	0 × 0000FFFF	94.6	107.3	117.6	127.3
EOR		90.6	104.2	114.9	125.0
EORnd		94.5	107.2	117.5	127.5
EOW	0 × 000FFFFF	70.5	83.4	93.5	104.2
EOR		66.4	80.3	90.9	101.4
EORnd		74.2	84.0	94.1	103.7
EOW	0 × 00FFFFFF	46.5	59.6	69.3	80.3
EOR		42.6	56.3	66.8	77.5
EORnd		45.9	59.8	69.9	79.9
EOW	0 × 0FFFFFFF	22.6	35.3	45.5	56.4
EOR		18.9	32.9	42.8	53.4
EORnd		25.2	33.0	45.8	56.0
EOW	0 × 7FFFFFFF	4.6	17.2	27.6	38.2
EOR		1.0	14.8	24.9	35.5
EORnd		6.2	17.5	27.8	37.8

the approximated filter. As a common consideration on the value of minimum required SNR, this strictly depend on application. Since, in this case, the filtering is applied to an audio signal, a value from 60 dB to 90dB could be considered of interest by most applications.

4.1.1. Impact on Output Using Approximate DRAM

Figure 4 shows the output SNR, with respect to the exact case, for an hardware platform using approximate DRAM memory. In the hypothesis of a circuit consisting of anti-cells, different error rates and looseness levels are explored. As reference, an error rate of about 10^{-3} is obtained in case of a 60× increase in refresh period.[10] The figure also shows that the looseness level can be used in order to rise SNR orthogonally to error rate, at the cost of keeping some bits exact. As expected in this application, each exact bit as an impact of +6 dB on SNR.

4.1.2. Impact on Output Using Approximate SRAM

Table I reports the output SNR in case of an approximate SRAM memory. The SRAM parameters are explored in the corners, i.e., cells affected only by (1) error on write (EOW) (2) destructive error on read (EOR) (3) non destructive error (EORnd). In this case, due to the access pattern of the application, it is possible to note that the EOW and the EORnd cases produce higher SNR than the EOR case. Again, by modulating looseness level, higher SNR values can be obtained for the same error rate.

Table II. FIR, SNR [dB] for SRAM bit dropping.

# of dropped LSBs 4 bits	8 bits	12 bits	16 bits	20 bits	24 bits	28 bits
134.7	122.4	106.1	82.2	52.9	28.2	4.0

Table III. ×264 heap memory usage.

Video resolution	×264 option (preset)	Peak heap [MB]	Peak usefulheap [MB]
176 × 144	Medium	15.6	15.4
1920 × 1080	Veryfast	269.0	238.6

4.1.3. Impact on Output Using Approximate SRAM with Bit Dropping

Table II reports the output SNR in case of the bit dropping technique applied to an approximate SRAM. The first column lists the value of the SNR obtained by keeping 24 bits exact and dropping the first four LSBs. Further columns were obtained by increasing the number of dropped bits four at a time. The results confirm that LSB dropping is a valid approach in case of high BER (as can happen in case of V_{DD} scaling at voltages below the minimum operating voltage[4]), since it completely eliminates the energy associated with dropped bitlines.

4.2. H.264 Video Encoder

The ×264 encoder is a free software library and application for encoding video streams into H.264 standard.[29] This application has been modified in order to allocate selected data buffers in approximate memory[30] and was executed in the AppropinQuo emulator, running a Linux kernel with support for approximate memory management.[31] The total amount of memory allocated by ×264 during the encoding process depends on video resolution and encoding options, Table III summarizes the results obtained for the cases under study. After the modification, about one third of memory formerly allocated on heap space was moved to approximate heap space (heap memory space mapped on approximate physical memory by the operating system).

Considering that, in this application, pixel values stored by approximated buffers have 8-bit size and are packed in 32-bit locations by the compiler, we set the looseness level as a repetition of an 8-bit submask (e.g., 0 × 0F0F0F0F). In order to produce reference values for the output, we first run the original ×264 encoder with buffers allocated

Table IV. ×264, video output PSNR [dB] for approximate DRAM (true cells).

Looseness mask	Fault rate [$errors/(bit \times s)$] 10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	Bit dropping
0 × FFFFFFFF	10.87	16.02	22.98	27.99	29.41	–
0 × 7F7F7F7F	15.30	20.34	26.90	29.25	29.60	–
0 × 3F3F3F3F	19.97	25.18	28.84	29.56	29.63	–
0 × 1F1F1F1F	24.47	28.01	29.43	29.62	29.65	–
0 × 0F0F0F0F	27.35	29.13	29.59	29.64	29.65	24.14
0 × 07070707	28.96	29.52	29.63	29.65	29.65	26.30
0 × 03030303	29.47	29.61	29.64	29.65	29.65	28.32
0 × 01010101	29.61	29.64	29.64	29.65	29.65	29.34

(a) Exact frame (b) Looseness Mask 0x0F0F0F0F

(c) Looseness Mask 0x1F1F1F1F (d) Looseness Mask 0x3F3F3F3F

Fig. 5. ×264, output frame with different looseness levels and fault rate 10^{-4} $[errors/(bit \times s)]$.

in exact memory. As quality metric, we used peak signal-to-noise ratio (PSNR), defined as the ratio between the maximum pixel value and rms of corrupting noise that affects the fidelity of its representation.[32]

4.2.1. Impact on Output Using Approximate DRAM

The results illustrated in this subsection are referred to HD videos (1980 × 1080 resolution) selected from the Xiph.org Video Test Media (derf's collection).[33] In this case the global PSNR value obtained on output videos for exact compression is 29.69 dB and this should be considered an upper bound to evaluate the ×264 performances with the present settings. Tests were executed considering an approximate DRAM composed by true-cells, varying fault rate and bit level error masking (looseness level).

Table IV shows the results of the same encoding using approximate memory. Global PSNR values are reported for each fault rate/looseness level combination. Figures 5 and 6 provide a visual result of user perceived video quality, comparing the original frame (Figs. 5(a) and 6(a)) and the same frame using different approximate memory error rates and looseness levels.

(a) Exact frame (b) Looseness Mask 0x0F0F0F0F

(c) Looseness Mask 0x1F1F1F1F (d) Looseness Mask 0x3F3F3F3F

Fig. 6. ×264, output frame with different looseness levels and fault rate 10^{-3} $[errors/(bit \times s)]$.

Table V. ×264, video output PSNR [dB] for approximate SRAM (error on access).

Looseness mask	Fault rate [*errors/access*]				
	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
0 × FFFFFFFF	38.65	47.58	56.75	63.48	64.41
0 × 7F7F7F7F	44.17	53.92	61.70	64.41	64.41
0 × 3F3F3F3F	50.14	59.58	63.92	64.41	64.42
0 × 1F1F1F1F	56.09	62.67	64.34	64.42	64.42
0 × 0F0F0F0F	61.79	64.11	64.41	64.43	64.44

In particular Figure 5 refers to an error rate of 10^{-4} and three different looseness levels (0 × 0F0F0F0F, 0 × 1F1F1F1F, 0 × 3F3F3F3F). For this fault rate, a looseness level set to 0 × 3F3F3F3F (Fig. 5(d)), which allows errors on the six LSBs of each pixel, still produces an output visually very close to the original frame. Lowering the looseness level does not have a significant impact on output while implying a larger number of exact bit cells. Figure 6 is obtained for a fault rate set to 10^{-3}; in this case we can see that, for higher looseness levels (i.e., 0 × 3F3F3F3F, Fig. 6(d)), differences in output quality are starting to be noticeable.

4.2.2. Impact on Output Using Approximate SRAM

The results illustrated in this subsection are referred to low resolution videos (176 × 144 resolution), selected again from the Xiph.org Video Test Media (derf's collection).[33] Tests were executed considering an approximate SRAM, where errors occurs both on read (destructive and non-destructive) and write accesses. As for the previous case, we first run the original ×264 encoding obtaining a global PSNR value of 64.46 dB for the exact compression.

Table V shows the global PSNR for several fault rate/looseness level combinations; in particular simula-

Fig. 7. ×264, output frame coded with exact (top left) and approximate SRAM (0 × FFFFFFFF looseness mask), fault rate 10^{-6} (top right), 10^{-4} (bottom left) and 10^{-2} (bottom right) [*errors/access*].

tions were performed setting the fault rates (EOW, EOR, EORnd) to the same value just to reduce the number of cases presented in the table.

These global PSNR values can be compared visually to effective output frames: Figure 7 illustrates the same frame obtained with exact compression (top left), and with three different fault rates using the higher looseness level (i.e., all bits cells are affected by errors). As expected, the output quality for 10^{-6} (top right) is not much different from the exact one. This consideration can also be extended to the case of 10^{-4} (bottom left), which presents only minor artifacts despite it is more than 15 dB under the exact PSNR. Finally the output frame obtained with a fault rate of 10^{-2} (bottom right) is the only one that clearly exhibits artifacts due to approximate memory.

5. CONCLUSION

In this paper we presented an emulation framework for embedded platforms with approximate memory, that includes error injections models derived from approximate memory circuits proposed in literature for DRAMs and SRAMs. Given an application, it is possible to explore the impact of memory errors and approximate memory techniques on its output, discovering the relation between memory errors and output quality. The fault models have been introduced in a modular way in the emulator, in order to allow extensions as new techniques and circuits for approximate memories will be proposed.

We showed the impact of approximation techniques on two real case applications, producing a figure of output degradation dependent on memory technology, error rates, looseness level. Results show the importance of exploring the relation between these parameters and output quality, since the grade of approximation (and hence power savings) is dependent on the amount of errors that the application can tolerate. The exploration allows indeed to select the point of work of approximate memory techniques, dependent on accepted output quality.

As future work, the introduction of energy consumption models for approximate memories in the emulator will allow to explore and directly determine the energy-quality trade off of a given combination of application and hardware platform.

References

1. J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, RAIDR: Retention-aware intelligent DRAM refresh, *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA'12)*, IEEE Computer Society, Washington, DC, USA **(2012)**, pp.1–12.
2. C. Weis, M. Jung, É. F. Zulian, C. Sudarshan, D. M. Mathew, and N. Wehn, The role of memories in transprecision computing, *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE **(2018)**, pp. 1–5.
3. F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, SRAM for error-tolerant applications with dynamic energy-quality management in 28 nm cmos. *IEEE Journal of Solid-State Circuits* 50, 1310 **(2015)**.
4. F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, Approximate SRAMs with dynamic energy-quality management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 2128 **(2016)**.
5. M. Cho, J. Schlessman, W. Wolf, and S. Mukhopadhyay, Reconfigurable SRAM architecture with spatial voltage scaling for low power mobile multimedia applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 161 **(2011)**.
6. M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, Exploiting partially-forgetful memories for approximate computing. *IEEE Embedded Systems Letters* 7, 19 **(2015)**.
7. N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, Muhammad Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 1 **(2011)**.
8. S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, Flikker: Saving DRAM refresh-power through critical data partitioning. *ACM SIGPLAN Notices* 47, 213 **(2012)**.
9. A. Teman, G. Karakonstantis, R. Giterman, P. Meinerzhagen, and A. Burg, Energy versus data integrity trade-offs in embedded high-density logic compatible dynamic memories, *Proceedings of the 2015 Design, Automation and Test in Europe Conference and Exhibition*, EDA Consortium **(2015)**, pp. 489–494.
10. A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, Quality configurable approximate DRAM. *IEEE Transactions on Computers* 66, 1172 **(2017)**.
11. D. T. Nguyen, H. Kim, H.-J. Lee, and I.-J. Chang, An approximate memory architecture for a reduction of refresh power consumption in deep learning applications, *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE **(2018)**, pp. 1–5.
12. F. Bellard, Qemu, a fast and portable dynamic translator, *USENIX Annual Technical Conference, FREENIX Track* **(2005)**, pp. 41–46.
13. M. Jung, C. Weis, and N. Wehn, DRAMsys: A flexible DRAM subsystem design space exploration framework. *IPSJ Transactions on System LSI Design Methodology* 8, 63 **(2015)**.
14. D. Burger and T. M. Austin, The simplescalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News* 25, 13 **(1997)**.
15. S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman, System-level power estimation tool for embedded processor based platforms, *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ACM **(2014)**, p. 5.
16. K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, and K. Goossens, DRAMpower: Open-source DRAM power and energy estimation tool, URL: http://www.drampower.info **(2012)**, Vol. 22.
17. K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas, Gemfi: A fault injection tool for studying the behavior of applications on unreliable substrates, *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE **(2014)**, pp. 622–629.
18. A. Höller, A. Krieg, T. Rauter, J. Iber, and C. Kreiner, Qemu-based fault injection for a system-level analysis of software counter measures against fault attacks, *2015 Euromicro Conference on Digital System Design (DSD)*, IEEE **(2015)**, pp. 530–533.
19. M. Jung, D. M. Mathew, C. Weis, and N. Wehn, Efficient reliability management in socs-an approximate DRAM perspective, *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, IEEE **(2016)**, pp. 390–394.
20. F. Menichelli, G. Stazi, A. Mastrandrea, and M. Olivieri, An emulator for approximate memory platforms based on qemu, *Lecture Notes in Electrical Engineering, International Conference on Applications in Electronics Pervading Industry, Environment and Society, APPLEPIES 2016*, Springer **(2017)**, Vol. 429, pp. 153–159.
21. J. Kwon, I. J. Chang, I. Lee, H. Park, and J. Park, Heterogeneous SRAM cell sizing for low-power h.264 applications. *IEEE*

Transactions on Circuits and Systems I: Regular Papers 59, 2275 **(2012)**.

22. D. M. Mathew, M. Schultheis, C. C. Rheinländer, C. Sudarshan, C. Weis, N. Wehn, and M. Jung, An analysis on retention error behavior and power consumption of recent ddr4 DRAMs. *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2018*, IEEE **(2018)**.

23. K. Itoh and M. Horiguchi, Low-voltage scaling limitations for nano-scale cmos lsis. *Solid-State Electronics* 53, 402 **(2009)**.

24. F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, 13.8 a 32 kb SRAM for error-free and error-tolerant applications with dynamic energy-quality management in 28 nm cmos, *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, IEEE **(2014)**, pp. 244–245.

25. X. Yang, N. Huang, Y. Chen, F. Qiao, and H. Yang, A priority-based selective bit dropping strategy to reduce DRAM and SRAM power in image processing. *IEICE Electronics Express* 13, 20160990 **(2016)**.

26. F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, Better-than-voltage scaling energy reduction in approximate SRAMs via bit dropping and bit reuse, *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, IEEE **(2015)**, pp. 132–139.

27. M. Jung, É. Zulian, D. M. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn, Omitting refresh: A case study for commodity and wide i/o DRAMs, *Proceedings of the 2015 International Symposium on Memory Systems*, ACM **(2015)**, pp. 85–91.

28. J. Lucas, M. Alvarez-Mesa, M. Andersch, and B. Juurlink, Sparkk: Quality-scalable approximate storage in DRAM. *The Memory Forum, 41st International Symposium on Computer Architecture (ISCA-41)*, Minneapolis, Minnesota, June **(2014)**, pp.1–9.

29. L. Merritt and R. Vanam, ×264: A high performance h.264/avc encoder, online http://neuron2.net/library/avc/overview_x264_v8_5.pdf. **(2006)**.

30. G. Stazi, L. Adani, A. Mastrandrea, M. Olivieri, and F. Menichelli, Impact of approximate memory data allocation on a h.264 software video encoder, *International Conference on High Performance Computing*, Springer **(2018)**, pp. 545–553.

31. G. Stazi, F. Menichelli, A. Mastrandrea, and M. Olivieri, Introducing approximate memory support in linux kernel, *2017 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, IEEE **(2017)**, pp. 97–100.

32. S. Winkler and P. Mohandas, The evolution of video quality measurement: From PSNR to hybrid metrics. *IEEE Transactions on Broadcasting* 54, 660 **(2008)**.

33. C. Montgomery and H. Lars, Xiph.org video test media (derf's collection), the xiph open source community, 1994, Online, https://media.xiph.org/video/derf.

34. J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. *ACM SIGARCH Computer Architecture News*, ACM **(2013)**, Vol. 41, pp. 60–71.

Giulia Stazi

Giulia Stazi *received the M.S. degree in electronic engineering (summa cum laude) in 2016 from the University of Rome "La Sapienza." Currently she is working towards the Ph.D. degree in Digital Electronic Systems at University of Rome "La Sapienza." Her scientific research is focus on the simulation of digital and embedded system platforms and on the design on low-power architectures based on Approximate Computing.*

Antonio Mastrandrea

Antonio Mastrandrea *received the masters (Laurea) degree (cum laude) in electronics engineering and the Ph.D. degree, from the Sapienza University of Rome, Rome, Italy, in 2010 and 2014, respectively. He is a Research Assistant with the Department of Information Engineering, Electronics and Telecommunications, Sapienza University of Rome. His current research interests include digital system-on-chip architectures and nano-CMOS circuits oriented to high-speed computation.*

Mauro Olivieri

Mauro Olivieri *(IEEE M98 SM'16) received the Master (Laurea) degree in electronics engineering "cum laude" in 1991 and the Doctorate degree in electronics and computer engineering in 1994 from the University of Genoa, Italy, where he was assistant professor from 1995 to 1998. In 1998 he joined Sapienza University of Rome as associate professor, teaching digital electronics and VLSI system architectures. He is a visiting researcher at the Barcelona Supercomputing Center within the European Processor Initiative. His research interests are digital system-on-chip design, microprocessor core design and digital nano-scale CMOS circuits. He authored over 100 papers and a textbook in three volumes. He has been a member of the TPC of IEEE DATE and was General Co-Chair of IEEE/ACM ISLPED15. He is an evaluator for the European Commission in the ECSEL Joint Undertaking.*

Francesco Menichelli

Francesco Menichelli *received the M.S. degree in electronic engineering (summa cum laude) in 2001 and the Ph.D. in electronic engineering in 2005 from the University of Rome "La Sapienza." In 2011 he joined the Sapienza University of Rome, where he is currently assistant professor. His scientific interests focus on low-power digital design and, in particular, in system level techniques for low-power consumption, power modeling, and simulation of digital system platforms.*