EReLA: A Low-Power Reliable Coarse-Grained Reconfigurable Architecture Processor and Its Irradiation Tests

Jun Yao, Mitsutoshi Saito, Shogo Okada, Kazutoshi Kobayashi, and Yasuhiko Nakashima

***Abstract*—In this work, facing pressure from both the increasing vulnerability to single event effects (SEEs) and design constraints of the power consumption, we have proposed a Coarse-Grained Reconfigurable Architecture (CGRA) processor. Our goal is to translate a user programmable redundancy to a guide for balancing energy consumption on the one hand and the reliability requirements on the other. We designed software (SW) and hardware (HW) approaches, coordinating them closely to achieve this purpose. The framework provides several user-assignable patterns of redundancy and the hardware modules to interpret well these patterns. A first version prototype processor, with the name EReLA (Explicit Redundancy Linear Array) has been implemented and manufactured with a 0.18 μm CMOS technology. Stress tests based on alpha particle irradiation were conducted to verify the tradeoff between the robustness and the power efficiency of the proposed schemes.**

***Index Terms*—Data flow computing, fault tolerance, reconfigurable architectures, redundancy.**

I. Introduction

THE continuous shrinking of transistor dimensions has provided the major advances in the functionality of microprocessors. However, the miniaturization of the size, capacitance and working supply voltage of transistors also brings up new problems such as process variations and the increasing vulnerability to cosmic rays, noises and wearouts. Hardwired redundancies, such as rad-hard circuits [1], [2], duplicated executions in IBM z990 [3], [4], and parity in Fujitsu SPARC64 [5], have been well explored to mitigate the single event effects (SEEs) on general purpose processing. These approaches provide a transparent SEE mitigation, which can be used seamlessly to cover all kinds of applications.

Manuscript received July 11, 2014; revised September 24, 2014; accepted November 02, 2014. Date of publication November 25, 2014; date of current version December 11, 2014. This work was supported in part by VLSI Design and Education Center, University of Tokyo, in part by Synopsys, in part by Cadence, in part by Mentor Graphics, and in part by KAKENHI and STARC Programs.

J. Yao, M. Saito, and Y. Nakashima are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma 630-0192, Japan (e-mail: yaojun@is.naist.jp; m-saito@is.naist.jp; nakashim@is.naist.jp).

S. Okada and K. Kobayashi are with the Graduate School of Science and Technology, Kyoto Institute of Technology, Kyoto 606-8585, Japan (e-mail: sokada@vlsi.es.kit.ac.jp; kobayasi@vlsi.es.kit.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2014.2367541

However, the duplicated execution and frequent comparisons to detect an error in the above techniques add visible pressures to the already tightly constrained power consumption. With the exponential growth in the number of transistors in use, processor design is now facing a utilization wall, i.e., due to thermal dissipation issues, not all transistors can be switched simultaneously. Adding a duplicated execution and a comparison becomes increasingly expensive with modern electronic devices, in which the constraint of this wall already dominates.

The redundant execution for reliability is sometimes considered to be over-designed, mainly because it lacks flexibility for individual applications. The redundancy level of most hardwired redundancy cannot be changed after manufacturing. For this reason, all data are unconditionally duplicated and checked, even when entering unimportant data zones. In this paper, we address this lack of flexibility found in most redundancy-based fault-tolerance methods with our processor architecture, named Explicit REdundancy Linear Array (EReLA). EReLA balances power consumption and reliability requirements by effectively exploiting an assignable execution redundancy. More specifically, this EReLA work offers two primary contributions:

1) A Coarse-Grained Reconfigurable Architecture (CGRA) processor is designed to be very compatible with an assignable redundancy level, by providing a hardware/software (HW/SW) cooperative framework. Users can easily indicate the target redundancy at the programming or compiling stages. The HW effectively reflects this explicitly assigned reliability requirement and performs the given fault coverage to address SEEs. In our work, the inflexible hardwired redundancy has been largely replaced by this programmable redundancy.
2) We provide easy-programmable code patterns to help users explicitly assign the importance of data (hereafter, data importance). In addition, our HW/SW cooperative platform is also easily extendable to hardware tuning modules, which analyze the data importance. In this paper, we give several empirical patterns of the data importance of some example program codes. Instructed by the data importance, the processor can optimally reduce the redundancy level for better power efficiency, while not sacrificing a lot of data coverage.

A prototype processor has been implemented to verify these design concepts, with a 0.18 μm CMOS technology. We have conducted stress irradiation verification of the prototype chip. Our results show that by applying simple checks on the control

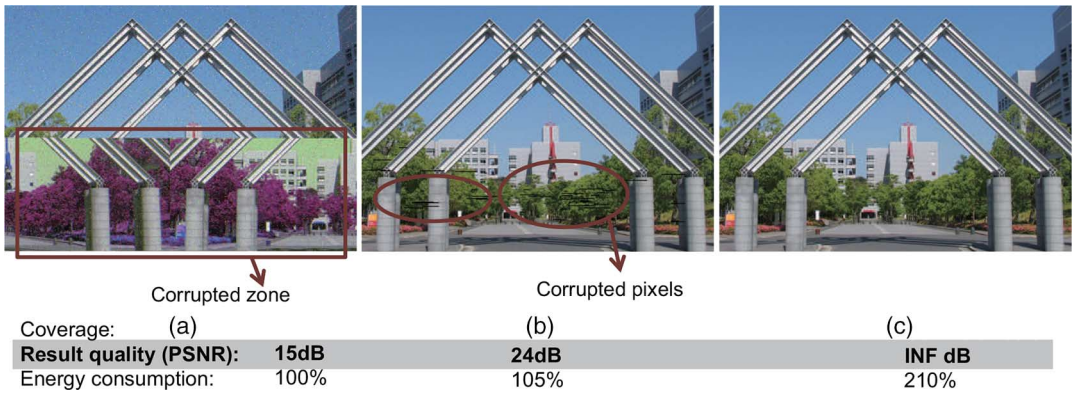

Fig. 1. Tradeoffs between the data coverage by fault-tolerance methods and quality of results. (a) No fault-tolerance. (b) Covers important data. (c) Covers all data.

path, we can avoid unexpected data loss caused by early program abort. These checks incur only 1% of additional power cost. Also, based on the analysis of the program data-flow, we avoid the over-designed excessively frequent data checks, and we place checks only at the final step before generated data are about to interfere with other program blocks. In this way, without sacrificing the robustness, we can save more than 8% in working power in our prototype chip by not having frequent check operations that are commonly performed in most conventional fault-tolerable systems.

II. The Necessity of Data Importance

Many fault-tolerance architecture, such as [6]–[8] have been proposed to guarantee a solid coverage for data. However, the increased chip area and energy consumption due to the full duplication are not preferred in most circumstances: they interfere with power-efficiency, and power efficiency is the most important requirement in some fields. The excessively high energy consumption comes from the multiple instruction execution in the redundancy processor, treating each individual instruction equally important. However, much recent research also indicates that instructions in a program have different weights in the final quality of program outputs. The most representative examples are the image processing applications, where several pixel data corruptions may lead only to an undetectable quality loss.

As an example, Fig. 1 shows the simulated results of heavy fault injections on three redundancy-controlled systems that work on an image processing application. For the purpose of maximizing the fault effect, the SEE rate in these experiments is largely accelerated: over 10^4 bit flips per second in the system. By the redundancy method, these three experiments explore, respectively, 0% data coverage, important data coverage, which is around 2% of the total executions, and full data coverage. The quality of each resulting image and the energy cost are depicted in Fig. 1 from left to right.

The image quality is given by the PSNR (peak signal-to-noise ratio) metric, with respect to the non-error injection result. Human eyes can hardly detect the difference between images with a PSNR larger than 40 dB, and can tolerate the difference between images with a PSNR above 25 dB. As can be expected, the full data coverage has no result different from the correct one, which is achieved at a cost of 210% energy consumption. Conversely, the image quality of the low-cost 0% redundancy (Fig. 1(a)) is visibly low, which shows a 15 dB PSNR to the correct result. Unlike the above two experiments, the method in Fig. 1(b) covers 2% of the total data, the 2% being designated as the important data processed in this image processing application. The noise in the Fig. 1(b) resulting image is already down to a tolerable level, indicated both by its visual quality and its PSNR value. Compared to the full data coverage by redundant execution, the additional energy for this important data coverage is small. Similar findings are also given in [9]. These results show a tradeoff between cost and efficiency in fault tolerance. Consequently, the aim of this work is to carry out a reliable and highly area- and energy-effective platform, which can facilitate the removal of over-designed redundancy according to the data importance.

III. Our Proposal: SW/HW Approaches to Achieve an Instructive Redundancy

A. EReLA: A CGRA Based Baseline Architecture

Basically, to designate well the data importance, a common way is to study the data relationships, i.e. the data flow, of this program. A general-purpose processor, normally, is not specially designed to have sufficient resources for holding a large data flow. We therefore start our work from a special hardware—Coarse-Grained Reconfigurable Architecture (CGRA) [10], [11], which is compatible with, specifically, data-flow based processing. Apart from the baseline architecture, our main proposal in this work is to use explicit redundancy on CGRA to tolerate faults. The CGRA in this research, therefore, is called Explicit REdundancy Linear Array (EReLA).

The baseline architecture of our CGRA processor is given in Fig. 2. The CGRA mainly focuses on accelerating the hottest program blocks, which can usually be abstracted into a loop form or a cascaded loop form. Since the innermost loop takes the most execution time, its data flow graph (DFG) will be the target workload to be handled and accelerated by the CGRA hardware. As shown in Fig. 2, our proposed CGRA processor contains a conventional pipeline, which is composed of instruction fetch, decode, register read, execution, memory access, and commit stages. It is used for non-loop executions. We also linearly add extra execution and memory access stages along the

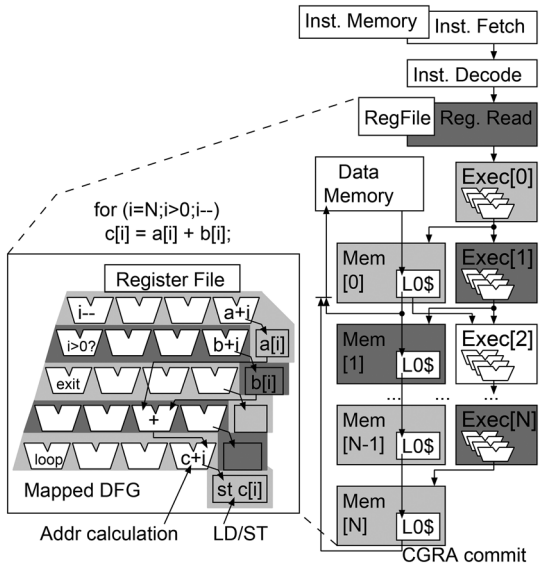

Fig. 2. Baseline architecture of EReLA, a Coarse-Grained Reconfigurable Architecture (CGRA) processor.

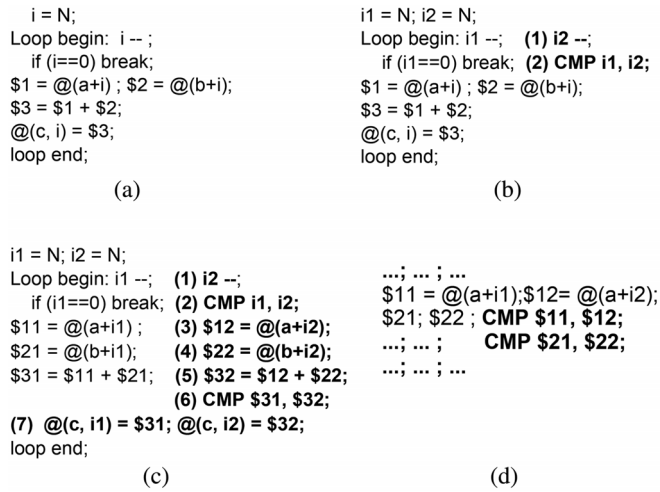

Fig. 3. Different redundancy levels, explicitly assigned at the programming phase. (a) Level 0: no redundancy. (b) Level 1: control redundancy. (c) Level 2: DMR + minimal CMPs. (d) Level 3: Traditional DMR, duplicates and checks all instructions.

vertical direction, whose FUs form a large linear FU array to map loop DFGs. This CGRA processes instructions based on the FR-V instruction set architecture (ISA) [12], which is a VLIW (Very Long Instruction Word) ISA. Each FR-V instruction can take three calculation operations and one load/store operation at maximum. This VLIW ISA also facilitates our fault tolerance. The details are to be introduced in other sections.

The example FOR loop kernel in Fig. 2, which loads two data streams and stores their sum into a third stream, can be mapped into the abstracted FU array by taking five array rows, known as array stages. This architecture can largely fold the execution of different loop iterations into a same cycle for a very high performance. Using the mapping in Fig. 2 as an example, in the first cycle, operations of $\mathtt{i}--$ and $\mathtt{a}+\mathtt{i}$ of the first loop iteration, where $\mathtt{i}=\mathtt{N}$, can be executed in the first array stage. In the second cycle, the execution of $\mathtt{i}>0?$ and $\mathtt{b}+\mathtt{i}$ of the $\mathtt{i}=\mathtt{N}$ iteration can start in the second array stage. Meanwhile, in this second cycle, since the first array stage has completed the $\mathtt{i}--$ and $\mathtt{a}+\mathtt{i}$ operations in the $\mathtt{i}=\mathtt{N}$ iteration in the first cycle, it becomes available and now processes the execution of $\mathtt{i}--$ and $\mathtt{a}+\mathtt{i}$ of the second iteration ($\mathtt{i}=\mathtt{N}-1$). In this way, the throughput of this special hardware is per cycle loop-iteration completion after the FU array is filled, and thus results in a very high performance. Note that data dependence between loop iterations is restricted for this acceleration. The execution detail of a similar CGRA was given in our previous paper [13].

The FU array is adopted in this work because of its high-speed acceleration of loop-based executions by mapping the DFG. Redundant executions can be added into the loop kernel DFG by duplicating the instructions. When the extended DFG can still be held inside the proposed FU array, the top performance, which only relates to the number of loop iterations, can be maintained.

B. Highly-flexible controllable dependability

This section describes our key contribution, called programmable redundancy here, which is offered by SW approaches. The concept is demonstrated in Fig. 3. More specifically, since we translate the data relationships in the program into the DFG, which is mapped into the FU array, we have the following findings, especially for a loop-based DFG.

1) The loop kernel can be divided and abstracted into two parts—the control and data processing. A control data corruption will lead to an uncontrollable number of loop iterations, or an infinite loop execution. Conversely, the corruption of data processing is isolated in individual loop-iterations when there is no data dependence across iterations.
2) A DFG mainly passes data to other DFGs via memory data. In the example of Fig. 3, the array c gathers the processed results in this DFG and provides input data to other parts of the program. Each DFG will converge into a limited number of output data.

Accordingly, if a loop control and stored data are verified, we can suppose that all its iterations are correctly executed. Moreover, the control data have more importance than the other parts of the DFG. We expect that these rules also apply to the program execution because, in general programs, about 90% of the execution time will be consumed by only 10% of the program codes, usually in a loop fashion.

Here, we apply the above findings for the loop DFGs to generate effective redundancy for programs. A compiler module is designed at the final stage of the binary generation to add this effective redundancy. We give two instructive redundancies that can be adaptively used for the SEE mitigation, as shown in Fig. 3. Starting from the C code, the level0 mode gives the non-redundant execution of this program. The level1 mode, which is denoted as the control redundancy, duplicates the control path of the loop kernel by adding instructions (1) and (2), while the data processing is not covered. Level1 is the partial redundancy to balance the energy consumption and reliability. The execution of level1 is expected to finish all the loop iterations while the store-data corruption in the processing leads to silent data corruption (SDC).

Secondly, the level2 mode represents a full duplication of all instructions in the loop kernel, where duplicated instructions from (1) through (6) are added to the original DFG. However, with the full understanding of the loop kernel, we know that the processing in one DFG converges to a limited number of outputs, which are usually in a store-data stream fashion. With this information, we only explicitly add the data check at the final store operation, testing the correctness of its address and its store-data. Though instruction (7) is written as a duplicated store, by hardware supports and extensions it internally performs a data check and an address check to detect possible influence from SEEs.

The level3 mode, with a dual execution and a data check for each instruction, is also given in Fig. 3. It is a traditional DMR mode and is not necessary in the SEE mitigation when we already have a clear understanding of the DFG. We only use it as a comparison mode in our explicitly programmable scheme.

These different redundancies are generated with compiler aids and can be exclusively applied according to the data importance and dynamic error rate sampling. In this way, the task of determining the redundancy has been extracted from HW to a flexible SW layer. This avoids the inflexibility arising from hardwired full duplication. Many possible combinations can be explored even after the processor is manufactured.

C. Compiler Aids to Generate Target VLIW Binaries with Different Redundancies

In this work, we also designed our own tool-chain, which is mainly a compiler module working in the final compiling phase to generate the redundant codes in Fig. 3. Our compiler module uses fully the parallel feature of the VLIW to achieve the redundancy augmentation.

Fig. 4 gives a typical VLIW block with three VLIW instructions. The selected VLIW instruction can carry one LD/ST (load/store) and three ALU operations at maximum, following a specification of a subset of the FR-V ISA [12]. The three VLIW instructions in Fig. 4(a) have read-after-write data dependence between them, which are given as the arrows in Fig. 4(a).

First, the compiler module starts from the instruction duplication, which is referred as VLIW instruction decomposition here. The VLIW instruction is decomposed into individual operations, which are originally in the VLIW slots. The LD/ST operation will then be duplicated along the vertical direction, following the duplication of the B operation in the example in Fig. 4(b). This is to satisfy the specification of VLIW, which exhibits only one LD/ST slot due to the hardware structure, i.e., only a one-port memory is assumed. The other operations will be duplicated along the horizontal slots. As shown in Fig. 4(b), another rule for this duplication is that instructions without data dependence, especially the LD/ST and ALUs, can be merged to save VLIW instructions.

This duplication will have the third ALU slot available for an easy augmentation of the CHECK instruction, as `A == A'`. According to our definition of level2, the data inside loop execution will be checked at the output of this DFG, so that only the store address and data will be checked before store operations. The control data path in level1 and level2, together with other

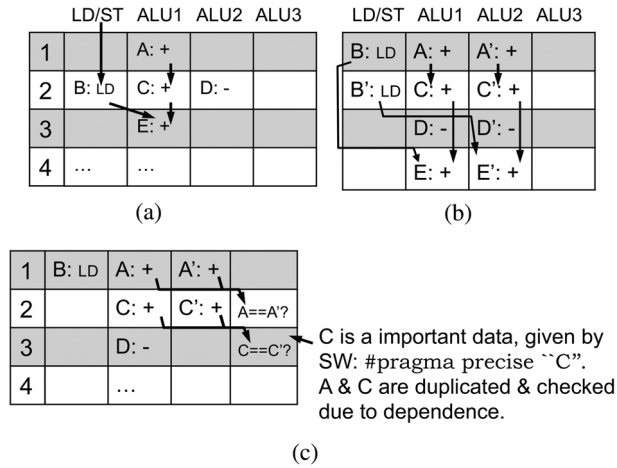

Fig. 4. Compiler aid to generate VLIW binary with explicit redundancy. (a) Original VLW block. VLIW: 1LD/ST + 3ALUs. (b) Duplication. Rules: I. LD/ST: vertically; II. ALU: horizontally; III. move LD duplication forward when possible (example B). (c) Selective Dup. & data check.

important data, will be additionally checked at the middle of the DFG. Fig. 4(c) shows this scenario. First of all, the data importance is indicated either by the automatic recognition of the control data, or by explicit user assignment—`#pragma precise "C"`—in Fig. 4(c). If level1 is assumed, only the related operations, as `A` and `C` will be duplicated. And data check instructions can be added by the compiler aid in the third ALU slots.

In this work, the control data refers to the loop index calculation, which will internally be regarded as an important datum and will thus always be checked. The loop-exit instruction follows a certain pattern like the instruction `bz i, loop_exit;`, which breaks the loop execution when `i` goes to 0. By analyzing the parent instructions to the branch instruction recursively, our compiler module can easily detect the dependent data flow related to i and then cover this flow. Other schemes to recognize data importance needs to understand the program itself. In this work, we provide a style similar to [9], which allows the user to assign the importance via `#pragma precise "C"`-style indications.

After the compiling phase, the binary is generated with the explicit redundancy indication, as shown in Fig. 4(b) and (c). The duplication in Fig. 4(b) will increase DFG to about 2 times in size. More specifically, the size will be dominated by the number LD/ST operations, because LD/ST can only be duplicated vertically. Currently, thanks to the slot utilization rate, which is usually around 30% to 50%, in the conventional VLIW programs due to the data dependence limitation, it is still easy to find empty slots for duplication. The DFG size will be 130% to 200% for most applications. For LD/ST intensive program, the size will reach 250%, in level2. Level1, however, does not have this problem. In level1, the redundancy DFG of Fig. 4(c), modified by our compiler module, will be the similar size of the Fig. 4(a) DFG, since only one or two operations are duplicated.

D. Error Detection and Recovery in Hardware

With the SW providing an instructive redundancy, HW extensions are added to interpret this explicit redundancy so as to detect and tag the erroneous execution accordingly. As in

Fig. 3(b)(c), the control path error is detected at the (2) CMP-instruction before the branch instruction. The data error is detected by the explicitly added comparison instruction, as the (6) CMP instruction in Fig. 3(c) shows. As mentioned above, the data check is also performed internally at the store operation by checking both the data and the address. In this way, we interpret the SW instructive redundancy/comparison and set error marks in the HW. The error marks will then be used to trigger error recovery. If a control error is detected, the loop execution will be instantly terminated and the recovery phase will start. This helps to avoid loosing control of the loop.

Otherwise, in our current first prototype design, if there is no control error in the whole loop execution, a flag will be used to accumulate all the error data inside the loop execution detected before the store operation. Based on the assumption that SEEs still occur infrequently, we predict that the loop will tend to have more successful completions without any error than the erroneous completions. When a SEE happens on the data part of the loop, the recovery will start after the loop is finished.

A control module in EReLA starts the recovery, for both the control-path and the data-path, by taking a rollback to the original states of the loop entry. For most cases, the loop starts from the assignment of the loop counter and base addresses of memory operations. As introduced in Section II, other data in the loop DFG are usually acquired via the load operations and via the child computations based on the load data. Therefore, the re-execution of the loop is easily performed by reassigning the above initial loop counter and the initial base addresses, following a method similar to that in [14].

However, as mentioned in Section III-B, level1 does not have data coverage in the data parts, and this leads to SDCs when SEE attacks. Our previous research gave a method to measure the SEE vulnerability of the next loop body according to the sampled SEE rate [15]. This method can also be used in this work. More specifically, we can assign level1 as the major method to mitigate SEEs on the control path. When the SEE vulnerability increases, level2 can be dynamically invoked to cover all data. Since the compiler gives both level1 and level2 codes, and the HW does the selection of these two modes, the cost for manual indications from users is kept at the lowest level.

IV. Prototype Chip and Irradiation Results

A. Basic parameters of the EReLA chip and the irradiation tests

A prototype of EReLA, shown as EReLAv1 (ver. 1) in Fig. 5, has been implemented to verify the above ideas. The core size of EReLAv1 is 37.5 mm^2 in a 0.18 μm CMOS technology, comprising 3.8M transistors and 10kB of on-chip SRAM. EReLAv1 includes 8 FU array pipeline stages to map DFGs of loop kernels. The floorplan of the 8 FU array stages is also given in Fig. 5(a). Note that the array stage 0 of EReLA has a larger area since the memory units are covered in this stage. Other stages have a similar size. As introduced in Section III-A, EReLA can extend its size vertically by adding more array stages, if a larger chip area is assumed.

Our irradiation test platform is shown in Fig. 5(b), in which the test board is able to supply to the processor a voltage

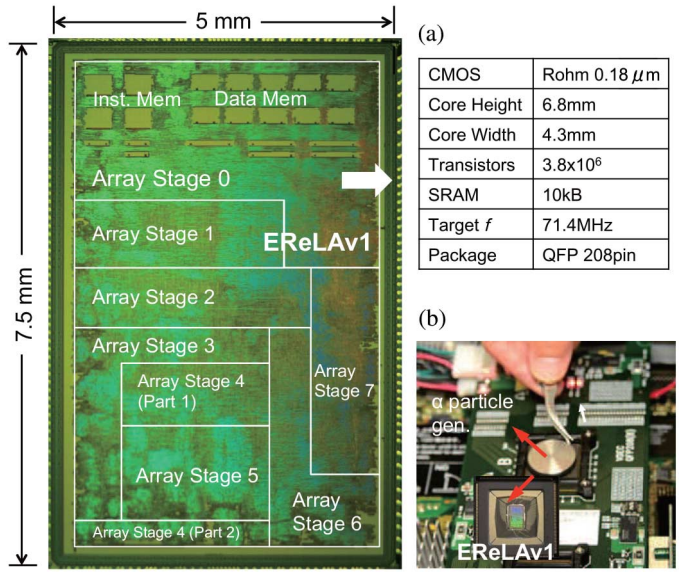

Fig. 5. EReLAv1 chip micrograph.

ranging from 1.25 V to 1.8 V. The irradiation test is performed by an alpha particle source, 3 Mbq 241Am. This test platform in Fig. 5(b) is the same as our previous work [8]. As described in [8], the irradiation generates 7.72×10^{-6} FF flips per bit per second under 1.8 V. This rate increases to 2.24×10^{-4} erroneous flips per second under 1.25 V. In this present study, we apply the 1.25 V supply voltage to the EReLA core to maximize the fault injection rate by our alpha source. Since EReLA has 5,632 unprotected DFFs, which are not covered by device hardening or ECC, these unprotected DFFs will be flipped at a rate of 1.26 FFs/second by the irradiation. Since EReLAv1 was originally designed to work with 1.8 V, defined by the 0.18 μm CMOS library, at a target frequency of 71.4 MHz, we lowered the working frequency to 25 MHz to guarantee that the critical path of the execution unit is not violated under 1.25 V.

B. Fault-Tolerance Tests by Simulation and Irradiation

First, we performed RTL simulations of EReLA, injecting one fault inside the EReLA processing units per each program run. The injected fault may hit randomly the loop control path, the operations in the loop body, and the units that are currently not occupied. Fig. 6(a) presents the results of four experiments of a simple image processing application, whose level2 DFG uses the full 8 stages in EReLAv1. The 4 level0–3 modes, respectively, are applied in these four experiments. Five hundred runs of each mode are performed to reduce the randomness. The results of each mode are given as the number of corrupted pixel data in the final image output.

The data of Fig. 6(a) shows that level0, without any fault-tolerance, leads to 6.11 incorrect pixels in the result. This number of errors is larger than the one fault that we injected. This is a direct proof that faults hitting on the control path terminate the loop execution immediately, leading to large parts of unprocessed data. Level1 can avoid this type of early termination error by merely adding data duplication and check on the control path. The simulated result of level1 shows that 0.25 fault per each injected fault eventually become errors, which is SDC

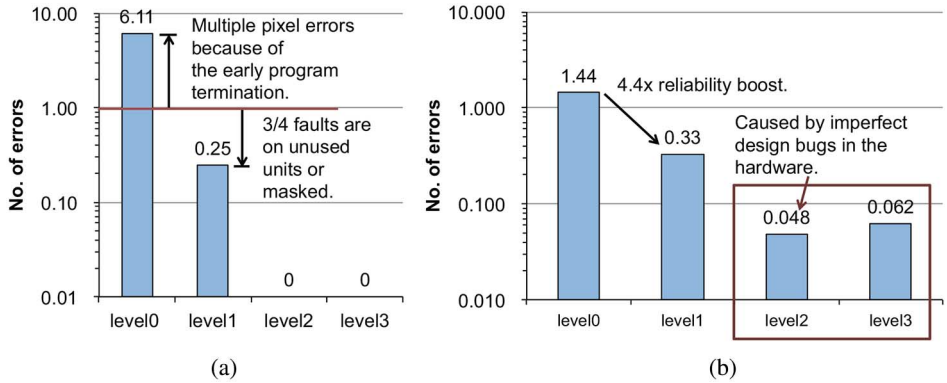

Fig. 6. Simulated error injection and irradiation test results (a) Simulated error injection results. 1 fault injection per each run (b) Radiation test test by α particle gen. Control the radiation time to allow one $\sim$ three faults per run.

error in the loop body execution. Also, this indicates that not all faults will be translated into an SDC. This is because on one hand some faults hit the unoccupied FUs inside the array and on the other hand the check in the control path can cover control errors. If image processing is the target application, the 0.25 SDC per each fault injection has only a negligible QoS drop in the final image data. Level2 and level3 are based on full DMR execution. These modes do not show any data corruption in the result.

The alpha particle irradiation experiments are then carried out to study the real chip-based fault-tolerance results of our SW/HW cooperative EReLAv1. We use the same way to count the number of faults and give the results in the same format, as in Fig. 6(b). Note that in this series of experiments, the number of fault injections cannot be precisely controlled. We managed only the irradiation time of each program execution by monitoring the number of loop iterations. According to our measurement, the alpha source injects faults at a rate of 1.26 FF-flips/second, considering the number of DFFs in EReLAv1. We controlled the run time of each test to inject around two faults, trying to increase the possibility that at least one fault would be injected. The injected faults may possibly range from one to three due to variations. Five hundred runs per each non-redundant or redundant mode were performed and the average results are shown in Fig. 6(b).

The irradiation results in Fig. 6(b) show a resemblance to the simulated fault injection data of Fig. 6(a). More specifically, the non-redundant level0 mode shows a visibly higher rate of errors than that of other fault-tolerant modes. The number of corrupted pixels in level0 is 1.44, averaged from the 500 runs. We have increased the dependability 4.4 times by applying the simple level1 mode, with merely covering the data on the control path. The 0.33 erroneous pixels in level1 experiments are the SDC errors. This 0.33 rate of errors is at a similar level to the simulated result of level1 in Fig. 6(a).

These error rates, around 0.2 to 0.3 erroneous pixels per each fault injection, are connected possibly to the rate of sensitive input/output DFFs, which is known as the utilization ratio of the EReLAv1 during the program runs. Back to the erroneous pixel result in level0, the rate of 1.44 errors per run is higher than the utilization ratio. This proves that under level0, one fault injection may connect to multiple errors in the results.

The difference between the simulation and irradiation in the level0 error numbers may come from the different fault injection schemes. In the simulation, to make the fault more visible, we apply the fault to the outputs FF of FUs. In irradiation, the fault strikes on the input FFs, output FFs and operation type FFs as well. Some of the faults in the irradiation, such as input FF faults, may likely be masked before becoming a visible error. Moreover, the different manufacturing of FFs is also a possible influence to the sensitivity to SEEs.

Also, from Fig. 6(b), we can observe that applying modes of level2 or level3 can lower the number of errors to a 1/7–1/5 level of that of level1. However, these rates are still higher than the error rate we were expecting. Since level2 and level3 apply fully DMR-ed executions to each instruction, all transient faults should be tolerated and the final number of errors should be zero, as the simulated results show. The uncovered modules in the EReLAv1 processor may cause these undetected errors. More specifically, as we re-checked the layout, the configuration DFFs, which are used to assign the operation to each individual FU of EReLAv1, are wrongly designed, not being covered by any error detection techniques. This is also possibly the reason that level3, with DMR and check instructions for every paired DMR-ed execution, demonstrates a higher final error rate than the level2 DMR mode. Level3 uses 50% more configuration bits than level2, so that from simple calculation, its dependability would be 33% worse than level2. The difference between the irradiation results of level2 and level3, which are respectively 0.048 and 0.062 pixel errors per each fault injection, is also around 30%.

Fig. 7 shows the mean-time-to-failure (MTTF) of these irradiation-based experiment data. Fig. 7 shows that level1 extends the MTTF of level0 by 4.4 times, by merely applying the duplication and comparison to the control path. Level2 and level3 further extend MTTF to 30 x and 23 x the MTTF of level0. We also give the estimated MTTF data of an ideal EReLAv1, if all imperfect configuration bits are covered by error correction code (ECC) technology. The correction is done by assuming that all data difference between level2 and level3 in Fig. 6(b) comes from the unprotected configuration bits. The failure shown in Fig. 7 refers to the corrupted pixels. Both of the two lines show a good tendency that adding fault tolerance can extend the MTTF. The MTTF data with the fix of unprotected configuration bits

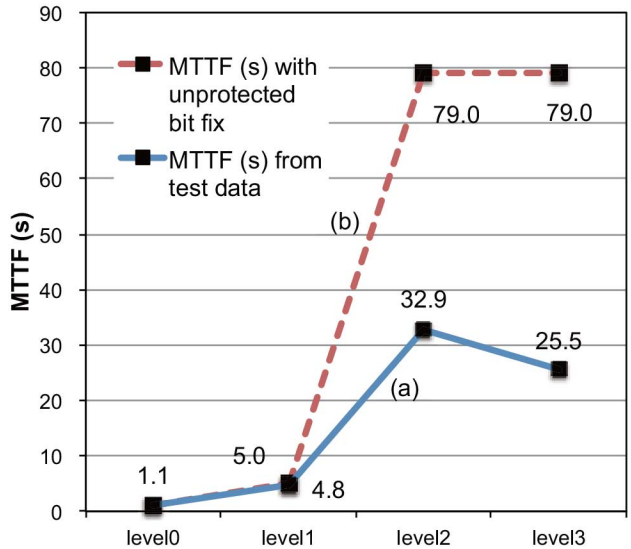

Fig. 7. Mean-time-to-failure (MTTF) results: (a) measured data from EReLAv1 prototype chip. Possible design bug of unhardened configuration FFs makes level2 and level3 MTTFs lower than the expectation. (b) fixed data by estimating the configuration FFs are protected by circuit hardening.

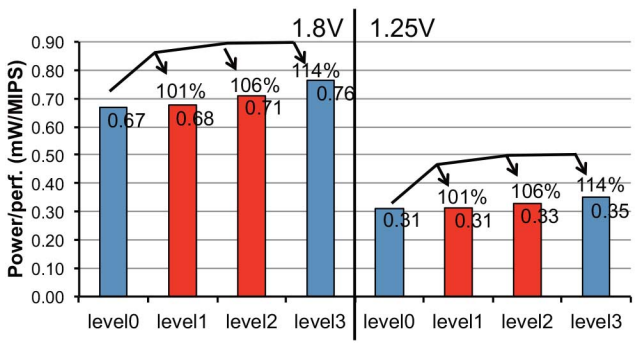

Fig. 8. EReLAv1 W/MIPS results of redundancy modes.

show a much higher MTTF in level2 and level3, even under our stress radiation test. The possible undetected errors in level2 and level3 after data fixing may now in the shared units of the EReLAv1 processor that still escaped protection in our design. Therefore, rad-hard circuits or ECC technique is necessary to cover these units outside the redundancy zone.

In our stress test environment, level2 has a better MTTF than level3, which verifies the point that the reliability is maintained with limited but correctly placed check-operations in level2. Therefore, the over-designed data checks in level3 can be reduced for better energy efficiency without sacrificing the data coverage. Apart from the full data coverage, from the viewpoint of extremely effective redundancy, we can select level1 as the working mode. Note that the 4.8s MTTF of level1 is observed in the accelerated irradiation tests, in the practical environment, level1 may show an optimal tradeoff between robustness and power efficiency. The power consumption data will be given in Section IV-C.

C. Power Consumption Results

The final target of avoiding over-designed redundancy is to achieve better power efficiency. Overall, measurements in Fig. 8 show that, level0 of EReLAv1 achieves 0.47 mW/MIPS with a 25 MHz clock frequency and 1.8 V supply voltage. This power/performance will drop to 0.31 mW/MIPS under the 1.25 V supply voltage.

Also, as shown in Fig. 8, when the redundancy modes are used for fault tolerance, the power/performance, in terms of mW/MIPS, grows according to the redundancy level. Note especially that the traditional level3 DMR gives 14% mW/MIPS increase, while our two proposals—level1 and level2—only give a 1% and 6% mW/MIPS increase, respectively. Also in EReLAv1, to lower the hardware complexity, no power gating technique has been applied. Without power gating, the unmapped FUs in the FU array also partially consume power, so that the power consumption differences from level0 to level3 are not very distinguishable. Overall, in this prototype chip, assuming that level3 represents the conventional DMR method, we can achieve an 8% reduction in power consumption by merely adding the necessary check instructions. Note that this does not sacrifice the data coverage. If some SDCs are allowed, we have a very low-cost fault-tolerant level1, which only increase the working power by 1% of from the non-redundancy mode. All this is achieved by our programmable redundancy in this SW/HW cooperative platform.

V. Conclusion

In this work, we have designed and implemented a prototype chip EReLAv1, which uses architectural redundancy instead of circuit hardening to mitigate SEEs. Programmable redundancies can be flexibly applied by merely changing program software, which largely extends the possible utilization of the designed hardware. The tradeoff between robustness and power efficiency has also been studied with the designed 0.18 μm technology chip. From our irradiation test, the very low-cost level1, which only covers the control data, extends the MTTF of the system by about 4.4 times. Also, our full data coverage level2 removes all unnecessary data checks based on the analysis of program behavior, which avoids over-design in conventional DMR methods. Compared to conventional DMR methods, such as level3, our level2 explicit redundancy has 8% better energy efficiency, without any degradation in the MTTF data, as verified by the stress irradiation test.

References

[1] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *Computer*, vol. 38, no. 2, pp. 43–52, 2005.

[2] R. Yamamoto, C. Hamanaka, J. Furuta, K. Kobayashi, and H. Onodera, "An area-efficient 65 nm radiation-hard dual-modular flip-flop to avoid multiple cell upsets," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 6, pp. 3053–3059, Dec. 2011.

[3] P. J. Meaney, S. B. Swaney, P. N. Sanda, and L. Spainhower, "IBM z990 soft error detection and recovery," *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 419–427, Sep. 2005.

[4] E. Rotenberg, "AR-SMT: A microarchitectural approach to fault tolerance in microprocessors," in *Proc. 29th Annu. Int. Symp. Fault-Tolerant Comput.*, 1999, pp. 84–91.

[5] R. Kan, T. Tanaka, G. Sugizaki, K. Ishizaka, R. Nishiyama, S. Sakabayashi, Y. Koyanagi, R. Iwatsuki, K. Hayasaka, T. Uemura, G. Ito, Y. Ozeki, H. Adachi, K. Furuya, and T. Motokurumada, "The 10th generation 16-core sparc64™ processor for mission critical unix server," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 32–40, Jan. 2014.

[6] T. J. Slegel, R. M. Averill, III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb, "IBM's S/390 G5 microprocessor design," *IEEE Micro*, vol. 19, no. 2, pp. 12–23, 1999.

[7] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith, "Configurable isolation: Building high availability systems with commodity multi-core processors," in *Proc. 34th Annu. Int. Symp. Comput. Arch. (ISCA)*, 2007, pp. 470–481.

[8] J. Yao, S. Okada, M. Masuda, K. Kobayashi, and Y. Nakashima, "DARA: A low-cost reliable architecture based on unhardened devices and its case study of radiation stress test," *IEEE Trans. Nucl. Sci.*, vol. 59, no. 6, pp. 2852–2858, Dec. 2012.

[9] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proc. 17th Int. Conf. Arch. Support for Program. Lang. Operat. Syst.*, Mar. 2012, pp. 301–312.

[10] C. Arbelo, A. Kanstein, S. López, J. F. López, M. Berekovic, R. Sarmiento, and J.-Y. Mignolet, "Mapping control-intensive video kernels onto a coarse-grain reconfigurable architecture: The H.264/AVC deblocking filter," in *Proc. Conf. Design, Autom. Test in Europe (DATE)*, 2007, pp. 177–182.

[11] S. W. Keckler, D. Burger, C. R. Moore, R. Nagarajan, K. Sankaralingam, V. Agarwal, M. S. Hrishikesh, N. Ranganathan, and P. Shivakumar, "A wire-delay scalable microprocessor architecture for high performance systems," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2003, pp. 168–169.

[12] *"FR550 Series Instruction Set Manual Ver.1.1,,"* FUJITSU LIMITED, Japan, 2002.

[13] K. Yoshimura, T. Iwakami, T. Nakada, J. Yao, H. Shimada, and Y. Nakashima, "An instruction mapping scheme for FU array accelerator," *IEICE Trans. Inf. Syst.*, vol. E94-D, no. 2, pp. 286–297, Feb. 2011.

[14] M. de Kruijf and K. Sankaralingam, "Idempotent processor architecture," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarch. (MICRO-44)*, Dec. 2011, pp. 140–151.

[15] T. Ahmed, J. Yao, and Y. Nakashima, "A two-order increase in robustness of partial redundancy under a radiation stress test by using SDC prediction," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 4, pp. 1567–1574, Aug. 2014.