A Self-Adaptive SEU Mitigation Scheme for Embedded Systems in Extreme Radiation Environments

Yufan Lu, Xiaojun Zhai, *Senior Member, IEEE*, Sangeet Saha, Shoaib Ehsan, *Senior Member, IEEE*, and Klaus D. McDonald-Maier, *Senior Member, IEEE*

***Abstract*—When electronic systems are working in radiation environments, transient errors, and permanent errors may occur. Static random-access memory (SRAM) has been the one of most significant parts in various semiconductor chips for its high performance and high logic density features. However, because of their dedicated electronic circuits, SRAMs are sensitive to radiation effects. In this article, a portable scheme combined with error correcting code (ECC) and refreshing techniques is proposed to correct errors and mitigate error accumulation in extreme radiation environments. Since the proposed scheme is small and transparent to other modules and no additional latency is introduced, it therefore can be easily applied to the system where the hardware modules are designed with fixed reading and writing latency. We evaluated this design by simulation in a hardware fault injection platform and radiation experiments in the neutron radiation facility. The results obtained in the neutron experiment, where the flux of neutron particles is 5×10^6 cm^2. s^{-1}, show that the number of bit-flips in 32 kB self-refresh ECC RAM on the Xilinx Artix-7 FPGA remains zero, while the number of bit-flips in unhardened RAM rose to 32 in 1.5 h.**

***Index Terms*—Error correcting codes (ECCs), neutron radiation, SEU mitigation, static random access memory (SRAM).**

I. Introduction

The natural radiation environment consists of electrons, protons, and a very small fraction of heavier nuclei, which are trapped by Earth magnetic field or produced in solar events and cosmic rays [1]. The interaction of these particles with other materials can further generate a cascade of secondary particles, including neutrons, protons, and electrons. Electronic systems based on semiconductor materials can also be affected in radiation environments. The interaction of these particles with electronics can lead to transient, permanent, or intermittent faults [2].

Commonly, there are two major types of radiation effects on integrated circuits: 1) cumulative effects and 2) single event effects (SEEs). Cumulative effects are long-term effects that can change the parameters of semiconductor materials, and these can be divided into two categories: 1) total ionizing dose [3] and 2) displacement damage [4]. In contrast, SEE is caused by a single ionizing particle, when the particle penetrates sensitive nodes within electronic devices. These can also be divided into a number of effect categories. The basic effects are given in Table I. Among them, SET and SEU are most common effects causing soft errors (recoverable or transient errors) [5].

TABLE I
Basic Categories of SEEs [6]

SEU	Single event upset	Temporary change of memory or control bit
SET	Single event transient	Transient introduced by single event
SEL	Single event latch up	Device latches in high current state
SES	Single event snap back	Regenerative current mode in NMOS
SEB	Single event burn out	Device draws high current and burns out
SEGR	Single event gate rupture	Gate destroyed in power MOSFETs
SEFI	Single event functional interrupt	Control path corrupted by an upset
MBU	Multi-bit upset	Several bits upset by the same event

Static random access memory (SRAM) is one of the most significant parts in various semiconductor chips, including CPU and field programmable gate array (FPGA). Due to a special feedback mechanism facilitating the memories data retention, it is also the most sensitive to the radiation. Therefore, with systems designed for radiation environments, it is necessary to harden the RAM to achieve higher reliability. In order to mitigate errors in the memory systems, a series of error mitigation strategies have been taken into consideration, including triple modular redundancy (TMR) technology, error correcting codes (ECCs), and scrubbing technology [7]–[9].

The TMR technology is a well-known fault tolerant technology [10]. It is an effective way to harden the systems. However, considering the large size of RAMs, it would require a large amount of hardware resources as consequence of the triple redundancy [11]. ECCs are widely known as another anti-interference encoding strategy, which requires less hardware resource, to enhance the reliability of memory devices and communication systems [12]–[14]. However, in radiation

Manuscript received August 17, 2020; revised September 6, 2021, December 20, 2021, and January 10, 2022; accepted January 13, 2022. This work was supported by the U.K. Engineering and Physical Sciences Research Council under Grant EP/R02572X/1, Grant EP/P017487/1, and Grant EP/V034111/1. *(Corresponding author: Xiaojun Zhai.)*

The authors are with the School of Computer Science and Electronic Engineering, University of Essex CO4 3SQ, U.K. (e-mail: yl19888@essex.ac.uk; xzhai@essex.ac.uk; sangeet.saha@essex.ac.uk; sehsan@essex.ac.uk; kdm@essex.ac.uk).

Digital Object Identifier 10.1109/JSYST.2022.3144019

environments, the number of error bits may accumulate and exceed the correcting capability. Scrubbing is an effective error mitigation technology for memory devices to resolve the accumulation of errors [15]–[17]. By keep rewriting the correct bits to RAMs, the error bits could be corrected in the early stage of the error accumulation.

Scrubbing method is suitable for hardening RAMs with static data (e.g., configuration RAMs in FPGAs). However, when it is applied to the RAMs with dynamic data [e.g., block RAMs (BRAMs)], it could be a problem for the scrubber and user modules to access RAMs at the same time. Normally, there are two options to address this issue. First method uses extra RAM ports. However, it could be very limited, there is no free ports in RAMs. Second method is to modify the timing sequence of user module to spare some bandwidth for scrubbing. However, the modifications in user modules are timing consuming and inconvenient, especially for dedicated user modules.

The proposed system is designed to support the computer system in radiation environment, one of the common features of such systems is that the clock frequency of processor is relatively lower than normal system. For example, the frequency of LEON3/LEON3-FT IP processor [18] (i.e., it is a widely used IP core in space applications) is designed under 100 MHz, this frequency is much lower than most of RAMs' frequencies. It is possible to unlock the potential of RAMs by increasing the frequency. Therefore, we proposed a self-refresh RAM design by doubling the frequency of RAMs.

In the proposed scheme, the scrubber requires no extra RAM access port in RAMs or modifications in user modules. Because the frequency of RAMs are doubled, there is extra bandwidth for scrubbing, and extra cycles for processing errors. For the user modules, the proposed design works just like a normal RAMs, hence it could be easily applied in various systems. In addition, considering that bandwidth for scrubber is equal to the user module, the scrubber can scan RAMs with high rates, which provides better error mitigation performance in radiation environments.

In this article, we use an Xilinx Virtex-5 XC5VLX110 T FPGA to build a hardware platform for simulation and an Artix-7 XC7A15T-1CPG236 C FPGAs to build a prototype for radiation experiments. The simulation platform consisting of a hardware and a software part is proposed to provide SEU injection and performance verification in real time. The hardware and software co-simulation shows that the proposed design can handle more than 99.9% and 99.97% of errors, while the SEU rates are 1×10^4 and 6.25×10^4 bit/s, respectively. The size of the prototype is 4 kB and the RAM frequency is 100 MHz. In the experiment, the observed error rates in an unhardened RAM is $1.2\ \text{bit}/(\text{kB} \cdot h)$. The errors rates in conventional ECC ram are approximately $4.3 \times 10^{-4}\ \text{bit}/(\text{kB}\cdot h)$, while the self-scrubbing RAM is less than $8.7 \times 10^{-5}\ \text{bit}/(\text{kB}\cdot h)$.

The main contributions of this article are stated as follows.

1) The proposed design is highly flexible. Compared to conventional external scrubbers [19]–[21], the proposed work will not occupy additional RAM access ports. The refresh controller can share the same ports with the original user modules, hence it can be adapted to various systems.

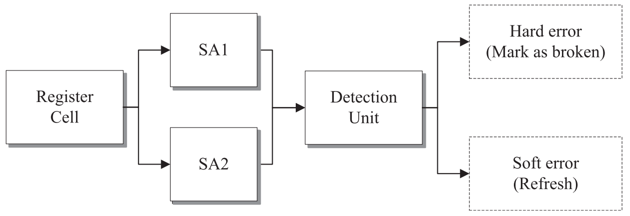

Fig. 1. Adding refresh circuit in memory cell (SA indicates voltage-sense amplifiers) [22].

2) The proposed design will not affect the timing sequence of the original user modules. When it is applied in RAMs system, there is no need to modify to the original user modules. Hence, it is very convenient to implement this scheme, especially when the user modules are dedicated circuits [e.g., finite-state machine (FSM)] that cannot be modify easily.
3) The proposed SEU mitigation design can achieve high SEU correction rates in various conditions. The results of the simulation and the radiation environment test follow the same trend. In simulation, the proposed design can correct more than 99.97% of SEU's errors at the SEU injection rate 6.25×10^4 bit/s. In the neutron radiation experiment, the SEU correction rate achieves 100%, when the flux of neutron radiation is $5 \times 10^6\ \text{cm}^2\text{s}^{-1}$.

The rest of this article is organized as follows. Section II provides a brief overview of related works, including some scrubbing designs. Section III presents the architecture of the self-refresh ECC RAM. Section IV presents the design of it's state machine and the scanning strategy. Section V presents the results of the hardware simulation platform. Section VI presents the results of the neutron radiation experiment designs. Section VII concludes this article.

II. Related Works

There is a plethora of error mitigation techniques for RAM. For example, researchers have employed redundancy, correcting code, and scrubbing to detect and correct errors. Those designs can be divided into three categories.

1) Redesigning the RAM cell circuits so that there is no separate controller.
2) Internal scrubbers for refreshing data, which are normally based on the built-in dedicated circuits.
3) External scrubbers, which access memory as ordinary modules with reading and writing operations.

A. Refresh Memory Cells

Tosson *et al.* [22] presented a refresh circuit for resolve the soft-error failures for memory cell. Two voltage-sense amplifiers are added to detect errors in bits lines. When errors occur, voltage-sense amplifiers can trigger data refreshing operations.

Fig. 1 shows the architecture of refreshing memory cell. The refresh circuit consists of SA1 (1-input), SA2 (2-input), and error detection unit. SA1 and SA2 are the voltage-sense amplifiers used to detect the RAM state of register cell. By comparing the

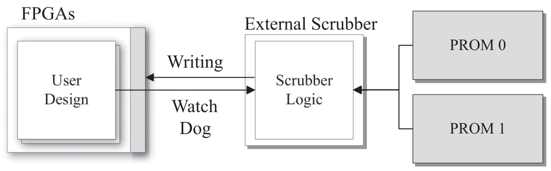

Fig. 2. Refreshing the configuration memory in FPGAs by using external scrubber and PROM.

voltage of different locations, different states of the memory device can be recognized.

Compared to separate scrubber, the RAM circuits are redesigned in this scheme, which brings lower detection time and performance loss. It takes less than 100 pico second to detect errors with less than 10% increase in the memory power consumption. However, the usage of this method is limited due to the higher prices and lower flexibility.

B. External Scrubber

Kumar *et al.* [19]–[21], [23] presented the external scrubber schemes where the external scrubber was independent of target devices. For example, in [19], a separate FPGA was programmed as the external scrubber for Geostationary Mission. In this article, there is a PROPM used to save initial program. By overwriting the contents of configuration memory on periodic basis, it can prevent system failure due to error accumulation.

It's a postconfiguration write operation in the configuration memory of Xilinx FPGA without disturbing the system operation. Basic block diagram of the system is shown in Fig. 2; there are two FPGAs in this scheme: 1) a targeted SRAM FPGA needed to be hardened and 2) an external scrubber implemented in a separate FPGA.

During power stage, the configuration parameters of Xilinx FPGA are first initialized to their default value. Configuration happens after the proper initialization of the configuration parameters. Read-back test is then performed immediately post-successful configuration to make sure that there is no hard error in target devices. Scrubbing operation on the FPGA resources is done only after successful read-back test. The contents of configuration memory are refreshed every 6 min on periodic basis.

C. Internal Scrubber

Another method is to internal scrubber in the scrubbing scheme. Zhang *et al.* [24], [25] proposed scrubbing methods for FPGA configuration RAM. They adopted the internal scrubbing method by using internal configuration access port (ICAP) to read and write configuration RAMs.

A basic architecture [24] of scrubbing platform is shown in Fig. 3. The FRAME_ECC logic calculates the syndrome value according to the bits in one frame including the ECC bits by reading frames from the configuration RAMs. The majority voter designed to detect the unexpected outputs in the user design. Once the errors are detected, the FSM in the srcubber are triggered to refresh the configuration RAMs. This method are suitable for configuration RAMs. In the configuration RAMs,

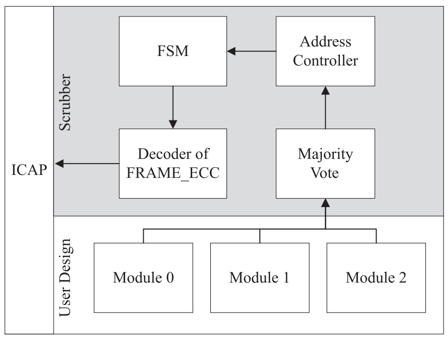

Fig. 3. Internal refresh scheme by using ICAP to access the configuration memory [24].

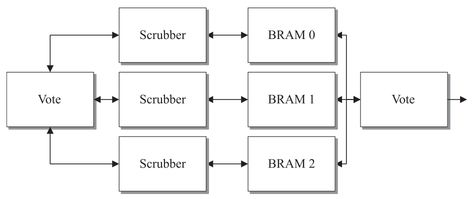

Fig. 4. Using dual-port RAMs to implement scrubbers in BRAM systems [26].

bits are static, so there is no additional read or write (R/W) operations to occupy the access ports.

However, scrubbing methods can be an challenging in BRAMs, if all available ports are in use. Keller and Wirthlin [26] proposed a scheme combining with TMR and scrubbing methods as shown in Fig. 4. In this article, dual-port BRAM modules are used for hardening the RAM in LEON3 processor. The processor use only one port or the BRAM modules, which leaves the other port left for scrubbing operations. In this scheme, BRAM and scrubber are tripled and the correct value is determined by voting between the redundant copies. However, this method will still occupy the BRAM access ports. In order to apply this method, single-port BRAMs are replaced by dual-port BRAMs. If the dual-port BRAM are already in use, the method in [26] will be limited.

In our case, there are a number of predesigned hardware modules using BRAMs in FPGAs. We want to update the RAM parts to improve the stability of the system. However, there are some difficulties to apply scrubbing methods in the system. First, due to the predesigned FSM in the hardware modules, modifications in time sequence of the hardware modules will be difficult, which means that we cannot just add the correction programs. Second, considering the usage of the dual-port BRAMs, there will not be enough RAM access ports for us to connect scrubbers. Third, because the TMR methods will need triple resources to work, the available RAM space will be limited.

To address these problems, we propose a self-refresh architecture with ECC techniques to enhance the RAM. The proposed scheme falls in the category of internal scrubbers, so that the design can be implemented in the FPGAs, without redesigning memory cells or using external chips. Compared to other methods, the proposed scheme requires no additional RAMs, so that it can be applied in a wide range of hardware systems.

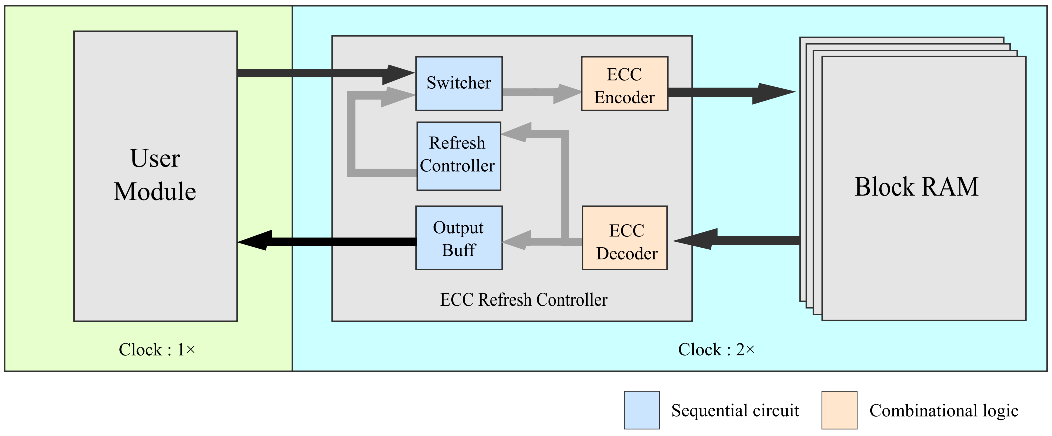

Fig. 5. Scheme of the external scrubber platform. User module will run at 1×clock speed, while the controller and RAMs will run at 2×clock speed.

III. Architecture of the Self-Refresh ECC RAM

Fig. 5 shows the system block diagram of self-refresh ECC RAM. Unlike the scrubbing scheme with dual-port RAMs where scrubbers use separate ports, the self-refresh controller and user modules share the same RAM ports in proposed system. In this scheme, the self-refresh controller is not only a scrubber to “clean” errors in RAMs, but also a transmitter for the passing of data from the user module. In addition, in order to minimize the effect on the read and write timing sequence of user module. The controller and RAMs operate at double frequency of the user clock, which means that the ECC refresh controller runs faster than user modules, and is thus, able to utilize extra clock cycles to perform additional tasks (e.g., fault detection and error correction) without interrupting the normal operations.

A. Switcher for Operations

As mentioned, the purposes of the controller are: 1) transmitting operations from user modules and 2) scrubbing RAMs. When the proposed scheme is applied in the system, the user module will access the controller with original read and write operations. For user modules, the controller will work just as a simple configured BRAM. The switcher will pass all controller signals from the user module to RAMs. By using classic RAM control circuits, the multiple features (e.g., enable signal and mask function) can be easily implemented for various hardware systems.

To achieve this purpose, the operations from user modules and the scrubber should be carefully arranged to make sure the timing sequence unchanged. Assuming that the outputs are ready in the next user clock cycle in the original timing sequence, the controller should also follow the same timing sequence. In the controller, a switcher module is designed to rearrange the sequence of operations from two directions: 1) user module and 2) refresh controller. As shown in Fig. 6, the external operations from user module and the refreshing operations are interlaced by the switcher. Considering that the controller is actually working at the double frequency of user clock. There will be one clock cycle left (in 2×clock domain) to transmit and return data.

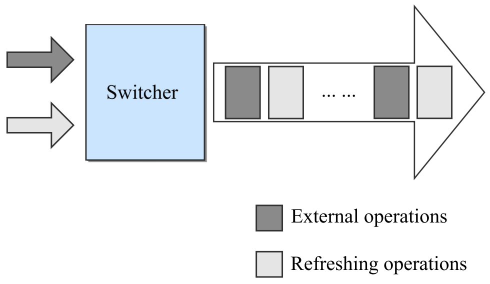

Fig. 6. Sequence of input operations stream. Operations from the user module and refresh controller will be placed one by one.

However, one clock cycle is still very limited to conduct transmission and ECC coding. Therefore, in this scheme, we use Hamming code as the correcting code. Because of its simple calculations, the decoder and coder can be designed fully based on logic gates. In this way, the coding processes will not require additional cycles.

B. Refresh Controller

The refresh controller is a core module of the proposed self-refresh ECC RAM. It has two operating modes: 1) the scan mode and 2) the refresh mode. When the system is working, the refresh controller continuously generates reading commands to read all memory units in the RAM periodically. Simultaneously, it checks the outputs of the ECC decoder, which is a combinational logic module to decode the outputs of the RAM. When SEUs occur, the ECC decoder can detect and fix the errors per byte. An error flag will also be asserted to indicate the occurrences of SEU. Then, the refresh controller will switch to refreshing mode and generate a writing command to refresh memory units.

C. Output Buffer

In order to resolve the timing problems, a buffer module is set between the ECC decoder and output port. Because the operations sent to the RAM are interlaced, the output data stream is also interlaced. To ensure that the external module will not be affected, the sequence of the output data stream needs to be

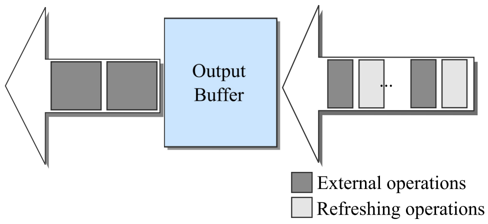

Fig. 7. Sequence of output data stream. The read data of the user module will be buffered, while the read data of the refresh controller will be blocked.

rearranged accordingly. As shown in Fig. 7, the output buffer module will block the outputs of refreshing operations. In other words, the output buffer allows only the outputs of external operations to go through in order to prevent the external module access to the data of refreshing operations.

In this architecture, the external operations (e.g., reading and writing) and internal operations (e.g., refreshing) execute in parallel, hence the performance of the systems will therefore not be affected.

IV. Hardware Implementation

As mentioned in the last section, as all operations from external modules (i.e., CPU) and the refresh controller are interlaced, the controller must arrange the sequence of the operation stream carefully. We will discuss the details of the FSM that enables this and how to handle the conflicts between external modules and internal modules.

A. Design of FSM

In order to distinguish from user module operations, in this article, "R/W operation" indicates read and write operations from user modules, while "scrubbing operation" indicates operations from the refresh controller. Considering the different clock domains of user module and the refresh controller, the operation sequence needs to be carefully arranged. In this article, the cycles (2×clock domain) used by the refresh controller are called "refresh cycles," while the rest cycles are called "user cycles". The refresh operations will be distributed into the "refresh cycles".

The state machine diagram of the refresh controller is shown in Fig. 8. The progress of refreshing starts from the rising edge of the user clock (1×clock domain), in this way, the output sequence can be synchronized with the user clock domain.

The refresh progresses start from S0, indicating that it is in refresh cycles. In S0, the RAM input port is occupied by refresh controller for reading data from the target address (address A). Then, in the next cycle (S1), output data from address A (DA) will be ready at the RAM output port. Because the decoder is combinational logic, the correcting results is also ready in the same clock cycle. If there is no error in the current address, the address controller will just move to next address and the FSM will move to the next round and return to S0. If the decoding results show the data from the target address is incorrect (DW), the FSM will switch to the error processing state (S2). In this state, corrected data will be written back to the target address.

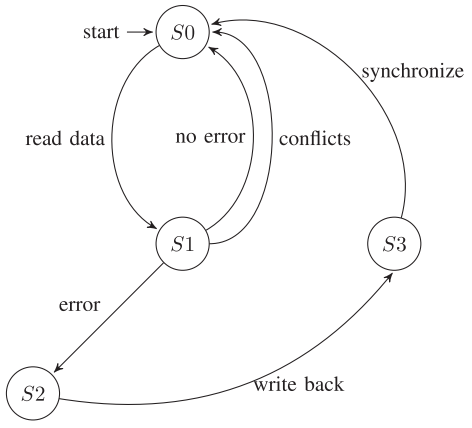

Fig. 8. There are four states in this FSM: state 0 (S0) (start, next), S1 (check), S2 (refresh), and S3 (synchronize). The R/W operations from refresh controller can only be conducted in S0 and S2, because the RAM access port is occupied by user module operations in S1 and S3.

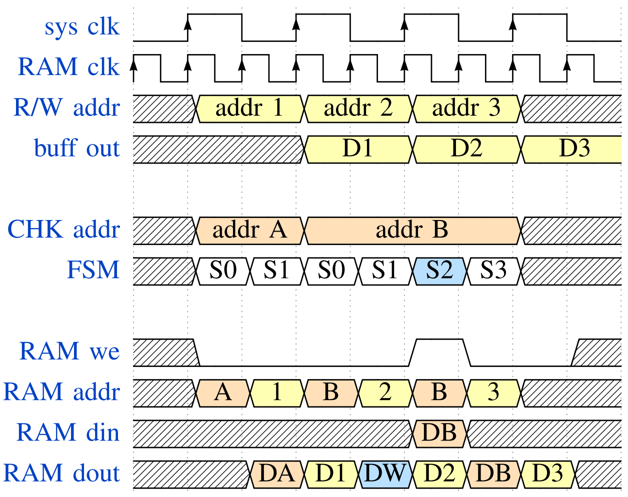

Fig. 9. Example of timing diagram without address conflicts. The addresses 1, 2, and 3 are different with addresses A and B. The refresh processes will not affect user operations.

After writing back, the target memory unit will have been refreshed. Finally, in state 3, the controller will be synchronized to make sure that S0 start from the next rising edge of the user clock.

Fig. 9 shows examples of the timing sequence of refreshing. The CHK addr is the checking address, which is read by refresh controller. The R/W addr represents the address operated by the user module. RAM addr represents the actual RAM address in operation. The CHK addresses and the R/W addresses are represented by alphabet and numbers, respectively, to indicate that the user module and refresh controller are accessing the different addresses.

In this figure, addr A represents an address with correct data, while addr B represent an address with incorrect data. FSM start from S0 to read the addr A. The output data and checking result are ready in the next state S1. Because there is no error, FSM move back to the S0 to read address B. If there is an error in

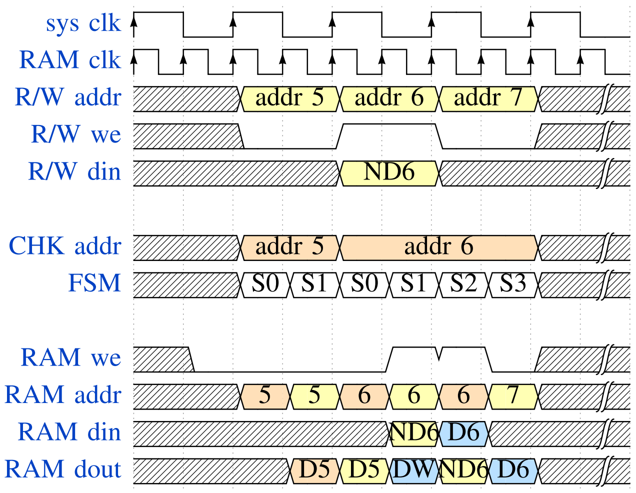

Fig. 10. Errors caused writing address conflicts. The user module is writing ND to addr 6, while the refresh controller is writing the "corrected" data (DW is corrected to be D6) to the same address. The ND is overwritten by D6.

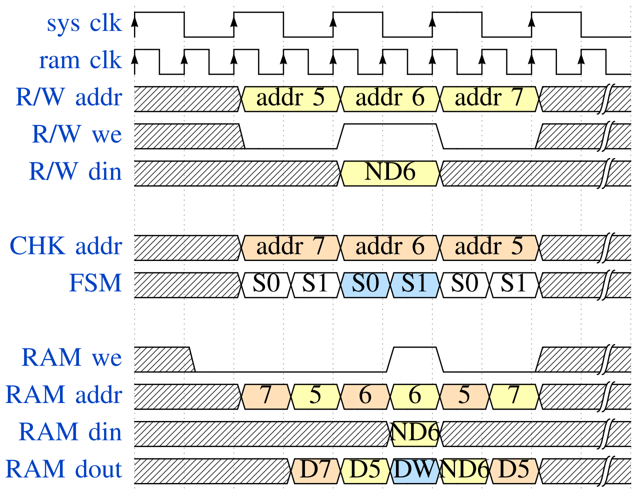

Fig. 11. Actual timing diagram of the proposed scheme. In address conflicts, the refresh controller gives up overwrite operations. Error bits are still corrected by the ECC decoder. Scan order of the address is also reversed to reduce address conflicts.

address B, FSM will move to S2 to overwrite corrected data to address B. Finally, FSM is synchronized in S3 to make sure the S0 start from refresh clock cycles.

On the other hand, the buffer module will block the RAM outputs used by refresh controller. The outputs of the buffer (D1, D2, and D3) will correspond to the sequence of user module operations (addresses 1, 2, and 3). All the refreshing operations are invisible to the external modules. Following the shown timing sequence, the user R/W operations will not be interrupted by either reading or rewriting processes. Hence, for user modules, the self-refresh ECC RAM works just like a normal single port RAM.

B. *Conflicts of Operations and Strategy of Scanning*

Typically, the FSM works as shown in Fig. 9, the refreshing progress without errors lasts for two RAM cycles and the refreshing progress with errors lasts for four RAM cycles. It is possible that user module writes new data (ND) to the same address that refresh controller are rewriting. In this case, the ND may be covered by the out-of-date "corrected" data. In this article, the cases that user module and refresh controller access the same address are called "address conflict".

Fig. 10 shows the time sequence of the address conflict in the refreshing processes. In this figure, the user module and refresh controller are accessing the addresses 5 and 6 at the same time. In the shown case, the data in address 6 is incorrect. The user module is writing ND to address 6 (ND6), while the refresh controller is reading the old data from address 6 (D6). In the S1 for address 6, ND6 is written to address 6, however, out-of-data D6 is written subsequently in S2 due to the previous checking results.

In order to solve this problem, the refresh controller has to monitor the address of user operations to ensure that there is no address conflict. In the proposed design, if the refresh controller accesses the address that is being written to by user module, it will give up the current operation and move to the next address directly, whether there is an error or not. In this way, refresh controller will not write out-of-date data back to the address in address conflicts by the costs of two cycles. However, considering that most external modules will access memory devices by the order of address, there is a good chance that the refresh controller and the user module will continue accessing the same address. Thus, in order to lower the probability of the address conflict, the refresh controller accesses the memory units by reverse order.

An example of actual timing sequence of the proposed scheme is shown in Fig. 11. The user module accesses memory units by order of addresses 5, 6, and 7, while the refresh controller accesses memory units by order of addresses 7, 6, and 5. When there is an address conflict, the refresh controller will skip the current address (address 6) and check the next address (address 5), while the user module will access a different address (address 7) in the next user clock cycles. In this way, there will be no continuous address conflicts.

C. *Parallel Architecture*

Due to the difference between error refreshing and no error refreshing processes, the time of scanning all memory units is not constant. In this system, scanning time is based on the memory size, clock frequency, and number of detected errors. In a RAM where N memory units are under detection, the frequency of the controller working clock is f and the scanning time T, can be represented by

$$\boldsymbol{T} = \frac{2n_1 + 4n_2}{f} = \frac{2(N + n_2)}{f} \tag{1}$$

where n_1 represents the number of memory units without errors and n_2 represents the number of memory units with errors.

In order to correct errors before the occurrence of the sequential error, the scanning time should be less than the error generation time. Therefore, the maximal scanning time (T_{max})

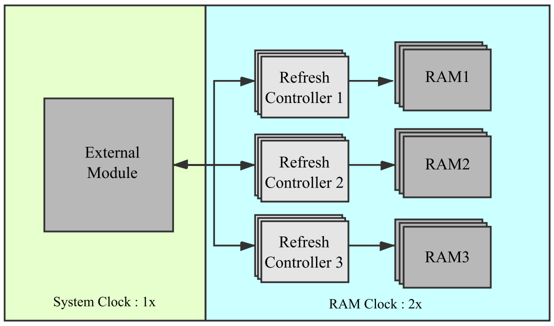

Fig. 12. Architecture of multirefresh controllers.

for stable executing can be represented by

$$\boldsymbol{T}_{\max} = \frac{1}{NR} \tag{2}$$

where R represents the generation rate of bit flips and N represents the number of memory units under detection. For example, if the given generation rate of bit flips in an environment is 1×10^{-3} bit/(N·h), the scanning time for 1 MB RAM should be less than 1×10^{-3} h.

As aforementioned, the scanning time depends on the frequency of the RAM clock and the RAM size under detection. Normally, the frequency of RAM is fixed during the running time. Therefore, the scanning time can be easily calculated by the size of RAMs.

Similar to the ordinary RAMs, the refresh controllers can also be connected in parallel by allocating different addresses to different controllers. Therefore, we can divide a large RAM into a set of smaller RAMs. Each refresh controller only needs to check a smaller memory size. Hence, the scanning time for each small RAM can be represented by

$$\boldsymbol{T} = \frac{2(N + n_2)}{mf} \tag{3}$$

where m represents the number of refresh controllers, n_2 represents the number of memory units with errors, and N represents the number of total memory units under detection.

In this way, we can change the number of units under detection to change the scanning time. By using such multirefresh controllers, we can lower the time of scanning significantly, which decides the system's performance of SEU mitigation.

The architecture of multirefresh controllers is shown in Fig. 12, the large RAM is divided into a set of smaller RAMs, the external module can access each small RAM by different addresses. Hence, we can use different architecture according to different environments. In high-radiation environments, we can use additional refresh controllers to compensate and achieve higher performance. On the other hand, we can also reduce the number of refresh controllers in order to save hardware resources at the cost of lower performance.

In addition, using small RAMs rather than a large RAM means that the size of the refresh controller can also be small. Considering that RAMs run at twice the frequency of the system clock, it can also help to resolve the setup time problems.

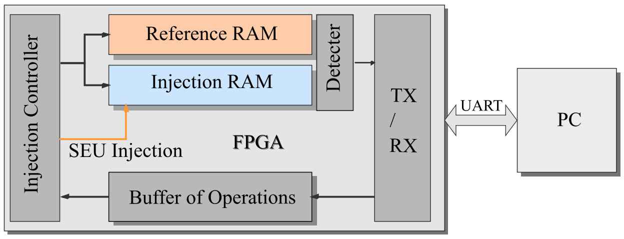

Fig. 13. Architecture of hardware fault injection platform.

V. Fault Injection Platform and Hardware Simulation

In order to verify the performance of the self-refresh ECC memory technology and to conduct functional tests, a hardware simulation platform was built to carry out hardware SEU fault injection. The self-refresh ECC RAM was implemented in this platform to evaluate the performance.

A. *Design of Hardware Fault Injection Platform*

This platform includes an FPGA part, performing SEU hardware simulation, and a PC software part for data analysis and human–computer interaction. Those two parts communicate via UART. The PC client part is mainly responsible for the operation control of SEU simulation platform and the display of error correction results. The FPGA part is designed to implement SEU fault injection and ECC verification.

The main architecture of the hardware platform is shown in Fig. 13. The operations communicated from the PC are stored in a buffer. There are two RAMs under test: 1) a reference RAM, which is a RAM without injection and 2) a injection RAM, where we carry out the SEU injection. In this article, both reference RAM and the injection RAM will be replaced by the proposed design.

When a simulation test starts, the reference RAM and the injection RAM will be read or written simultaneously according to the pregenerated operations. At the same time, the injection controller will inject the error bits into the memory units of the injection RAM to simulate the occurrence of SEU. The platform can also simulate the intensity of radiation by adjusting the probability of the occurrence of the single particle effect and the frequency of SEU injection. This allows it to evaluate the effects of different factors and performance of hardening design in different situations. By using the equivalent circuit without the injection, the simulation platform can simulate the state of the module in both radiation and nonradiation environments.

In this article, the errors that may affect the system are called functional errors. In the unhardened RAMs, all the errors read by the system are functional errors, while in the proposed systems, functional errors are the errors not corrected by the refresh controller. In the simulation, different outputs between the injection RAM and the reference RAM suggest functional errors. We evaluate the SEE mitigation performance by comparing the functional error rates. All data generated in the RAM operation is exported to the PC to analyze the capability of SEU mitigation of the tested module in real time.

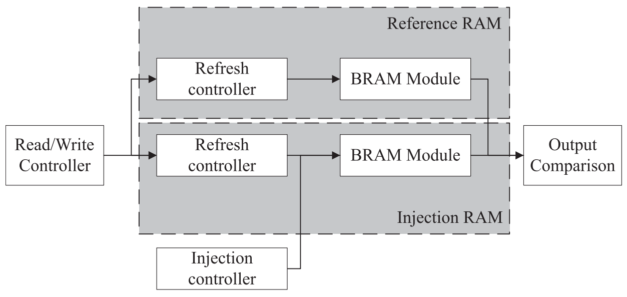

Fig. 14. Injection controller will bypass the refresh controller and write error bits into RAMs directly. The BRAM module is modified to accept data from both refresh controller and injection controller.

B. *Fault Injection in Hardware Platform*

Unlike the software simulation, the hardware simulation cannot simply simulate the occurrence of SEU by just specifying bit flip. The hardware simulation platform has incorrect bits written into the target memory units to simulate SEUs. To ensure that the RAM logic function will not change during execution, we adopted the idea of self-refresh ECC RAM and double the system clock to make the injection controller operate at a higher frequency. Hence, the proposed simulation system can make use of extra clock cycles to perform SEU injections.

Also, to detect the effects of SEUs on different instruction sequences in RAM, another RAM is used to store instructions. By reading or writing the RAM with a specific operation order, this platform can also simulate the code execution of different software programs. Hence, we can also evaluate the capability of SEU mitigation of different programs.

Fig. 14 shows the block diagram of the injection modules for evaluate the proposed design. In the simulation the injection controller is connected to the BRAM module directly. BRAM modules are not just BRAMs. It is a module with BRAMs inside and can conduct operations from refresh controller and injection controller. By using the similar method (e.g., double frequency), the operations refreshing controller will not be affected.

During hardware simulations, the refresh controller should keep sending R/W operations to BRAM module. The read/write controller will work as the user module and keep sending operation to refresh controller. In the meantime, the injection controller writes error bits into BRAMs from time to time. Inside the injection controller, there is random number generator. After each injection, it will generate a random interval time for the next injection. The random seed, average injection time, and floating range can be set by users. The actual interval will float randomly within the range around the average time.

C. *Results of Simulations*

Table II gives the main technical specifications of the SEU hardware simulation platform, which is built on an Xilinx Virtex-5 XC5VLX110 T FPGA. The system works at a frequency of 100 MHz and has ability of SEU injection at maximal frequency of 50 MHz. The executor programming simulated by the hardware platform works at frequency of 25 MHz, which equals the frequency of R/W operations. Hence, according to (1), the scanning time is approximately 160 μs and the SEU rate is 6.25×10^4 bit/s.

There are following three simulation modes to satisfy different requirements.

1) Single mode means performing all saving R/W operations for one round, which is designed to test certain programmes.
2) Loop mode means repeating R/W operations for specified times rounds, which is designed to test the performance of the systems in a specified time.
3) Unlimited mode means continuously R/W operations, until it is stopped by users, which is designed to evaluate the error rate.

Fig. 15 shows the performance of the self-refresh ECC RAM and an unhardened RAM in hardware simulation. The injection time represents the average interval time between two SEUs. In this simulation, all the operations are generated by PC in a random generated order. Because the number of operations is much higher than the number of memory units, the injection controller may access the same unit multiple times, before the errors in this memory unit are covered or refreshed by ND. In other words, the number of detected functional errors may be greater than the number of SEU injections. As we can see, after using the self-refresh ECC technique, the number of functional errors is reduced to almost 0, which validates that the proposed design effectively avoids functional errors caused by SEU. Error rates of different average injection time are shown in Fig. 16. When the average injection time is more than 100 μs, which

TABLE II
SPECIFICATIONS OF HARDWARE FAULT INJECTION PLATFORM

Description	Specifications
RAM size for SEU injection	4096
Size of command buffer	4096
Maximum number of R/W operations	4096 $\times$ 256
System clock	100 MHz
Frequency of Read or Write operation	25 MHz
Clock of refresh controller	50 MHz
Max frequency of SEU injection	50 MHz
Simulation mode	Single/Loop/Unlimited

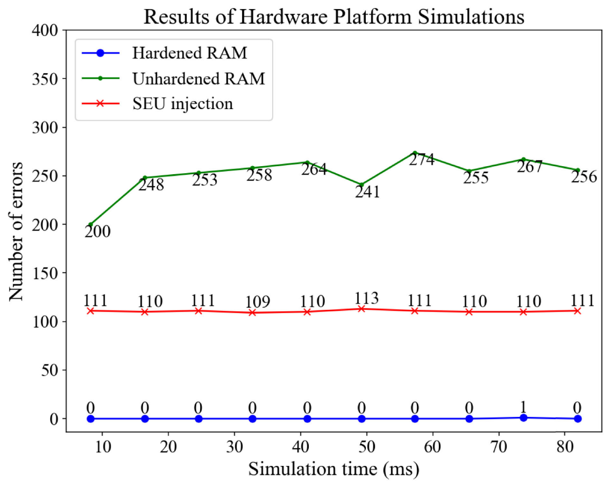

Fig. 15. Results of simulations with 80-μs injection time.

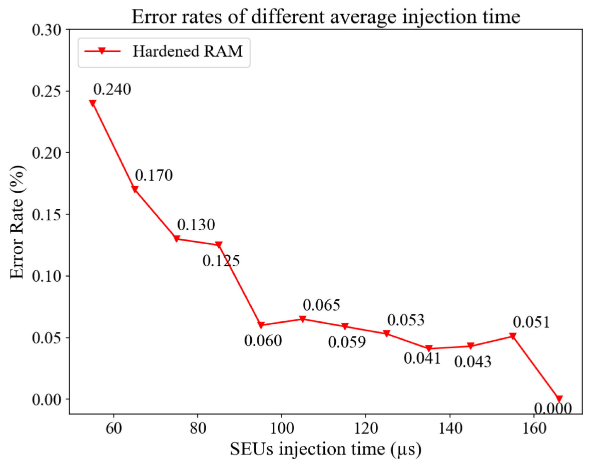

Fig. 16. Error rates of different average injection time.

means that the SEU rate is 1×10^4 bit/s, the self-refresh ECC RAM can handle 99.9% of errors. When the injection time is equal to the scanning time which is 160 μs, the self-refresh controller can handle 99.97% of errors.

The injection RAM and the reference RAM are changeable in this platform. By replacing the injection RAM and the reference RAM with other hardened RAMs, this platform can also be used to evaluate the performance of other SEU reinforcement design. By this platform, we verify the reliability, functionality, and effectiveness of self-refresh ECC RAM. However, hardware simulation is not equivalent to test with real radiation. Therefore, it is necessary to test the proposed systems in real radiation environments.

VI. Neutron Radiation Experiments

In order to evaluate the real world performance of the system, we conducted experiments with neutron radiation. Neutron radiation was used to create extreme environment to evaluate the SEU mitigation performance of self-refresh ECC RAM [27]–[29].

A. Setup of the Neutron Experiment

Radiation experiments were conducted at the ChipIr facility at ISIS, Didcot, U.K. [30]. ChipIr provides a neutron spectrum, which is suitable to emulate the effects of terrestrial neutrons in electronic devices and systems. The ChipIr neutron flux (with $E_n > 10$ MeV) has been measured to be approximately 5×10^6 cm^{2}s^{-1}. The neutron flux at ChipIr is about eight to nine orders of magnitude higher than the terrestrial flux at sea level. Calculated by the scientists from ISIS, the radiation dose on water (human body) is about 20 000 mSv/h in our experiments.

We design two experiments to evaluate the proposed design: 1) comparison between unhardened RAMs and the self-refresh RAMs and 2) comparison between the conventional ECC RAMs and the self-refresh RAMs. In the experiments, the systems are placed under the neutron beams for several hours, that amounts to a neutron yield of 6.5×10^{10} per hour considering a 7×7 cm beam. This is equivalent to more than half a million years of natural exposure.

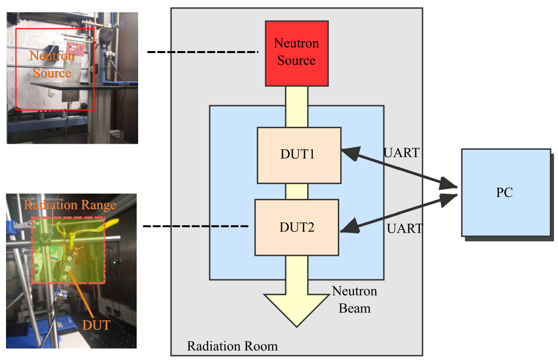

Fig. 17. Setup of neutron radiation experiment.

TABLE III
Specifications of Hardware Implementation

Description	Specifications	
	Unhardened RAM	self-refresh ECCRAM
Available RAM size	32 KByte	32 KByte
BRAM*	16.0	21.50
LUT	562(5.4%)	681(6.5%)
LUTRAM	8	8
FF	467	575

*Number of the 36-kB BRAM used in the FPGA.

The setup of the neutron experiment is shown in Fig. 17. The devices under test (DUT) are placed in the radiation room. Because of the strong penetration of neutron beam, it can penetrate all test boards. The DUTs in the radiation room are connected to the PC in control room by long USB cables. In order to reduce the impact from other electronic components, the communication modules and power supplies used in the experiment were kept outside of the radiation range.

B. Hardware Implementation of the Self-Refresh ECC RAM

In the radiation experiment, the self-refresh ECC RAM is implemented on Digilent Cmod A7-15 T, which is a low-price entry-level FPGA development board. The chip on this board is an Artix-7 XC7A15T-1CPG236 C FPGA with 112.5-kB block RAM inside.

The basic architecture of DUT is similar to the abovementioned hardware injection platform but no injection controller and reference RAMs. The design of experiment circuit includes two parts: 1) the UART controller and 2) the target RAM. When the boards are working, the PC client sends compressed write or read commands to the FPGA part via UART periodically. Subsequently, the UART controller operates target RAMs according to those commands. All outputs of the read operations will be sent to the PC immediately, to avoid the impacts of radiation.

The specification of the hardware of the whole design is given in Table III. Both the reference RAM and the target RAM have the same available memory size, which is 32 kB and the same available bandwidth for external modules. Hence, the self-refresh ECC RAM consumes slightly more memory and requires a higher frequency of the RAM clock. The unhardened

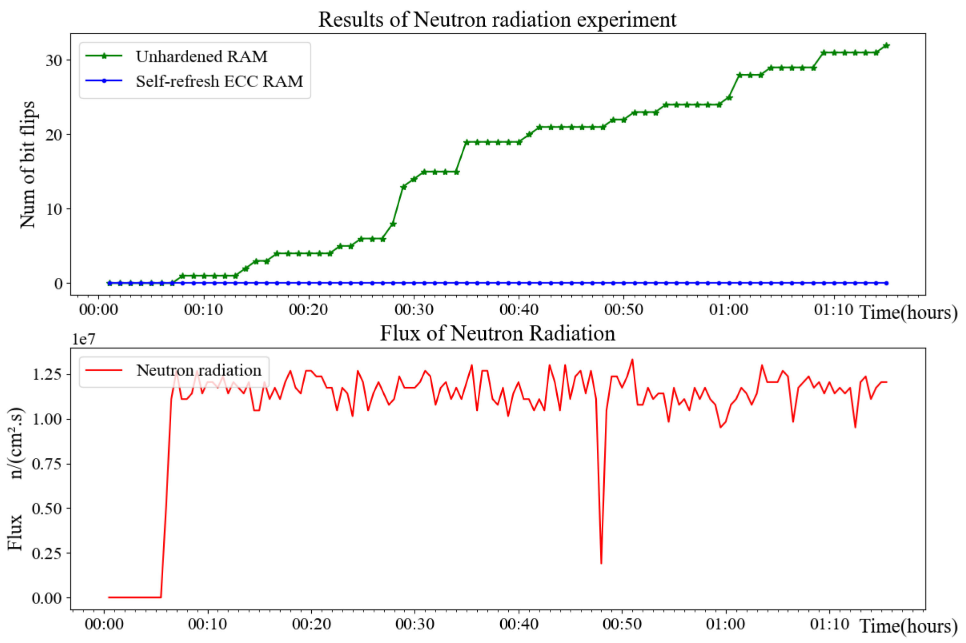

Fig. 18. Scheme of the external scrubber platform.

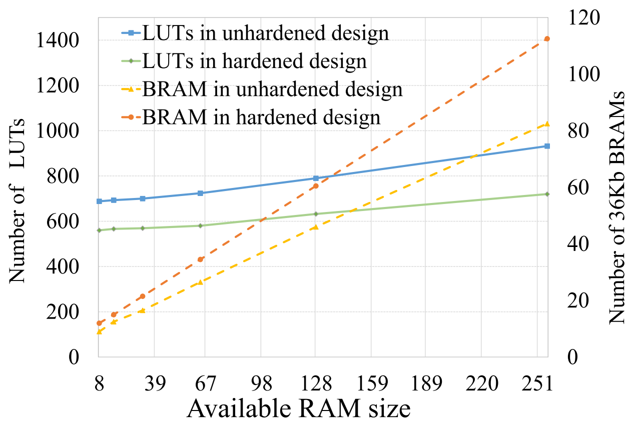

Fig. 19. Hardware costs with the scale of the design.

design and hardened design consume 16 and 21.5 BRAM units, respectively. The operating frequency of the unhardened RAM and the self-refresh ECC RAM are 50 and 100 MHz, respectively. The number of the LUTs for the entire design included UART and operation parts, the unhardened design and hardened design use 562 and 681 LUTs, respectively. The additional 121 LUTs are used to build the self-refresh controller.

The scalability of the proposed design is shown in Fig. 19. It shows the trends of the utilization of LUTs and BRAM with available RAM size. When the available RAM size is 8 kB, the unhardened and hardened designs consume 560 and 688 LUTs, respectively. When the available RAM size was 256 kB, the unhardened and hardened designs consume 720 and 932 LUTs, respectively. The additional utilization of the LUTs scales up slightly for wider bandwidths. The refreshing controller itself does not scale up. The utilization of BRAM grows linearly with the available RAM size.

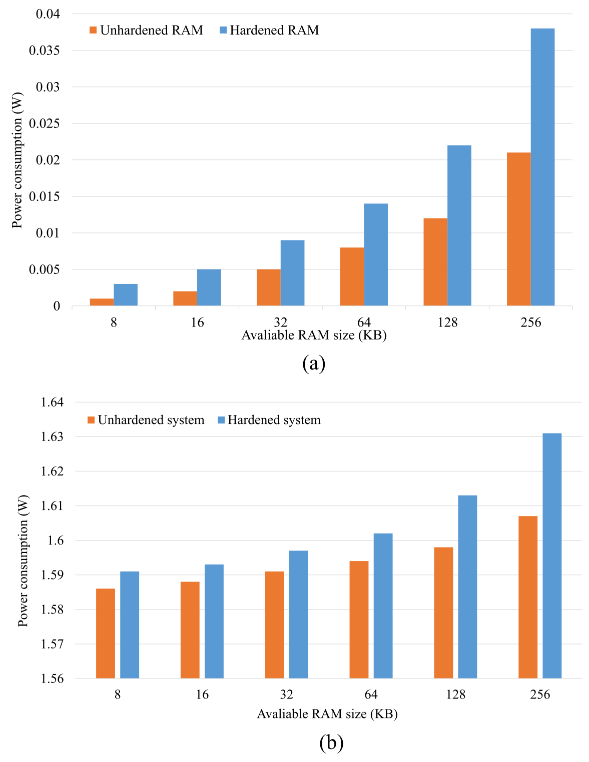

Fig. 20. Power consumption with the scale of the design. (a) Power consumption of the RAM part. (b) Power consumption of the system.

The power analysis generated by the Vivado analysis tool is shown in the Fig. 20. Compared to the unhardened RAM, the power consumption of the hardened systems also slightly scale up with the RAM size. There are two parts to the additional power consumption. First, the dynamic power consumption of the self-refresh ECC RAM increases for operating at higher frequency. Second, hardened design consumes more power for the additional control circuits.

When the available RAM size is 8 kB, the unhardened RAM and the hardened RAM consume 0.001 and 0.003 W, respectively. When the available RAM size is 256 kB, the unhardened RAM and the hardened RAM consume 0.02 and 0.038 W, respectively. However, the RAMs consume much less power than the processor in the designs. The total chip power consumption does not increase significantly. When the RAM size is 256 kB, the hardened system consumes 1.4% more power than unhardened system.

C. *Analysis of the Return Data*

In the radiation experiments, the entire FPGA board was placed in the radiation room, hence the communication and control modules implemented in FPGAs were also be irradiated. It is possible that the detected error bits are false positive results instead of the actual errors in BRAMs. Fortunately, it is possible to identify the errors types in the systems. In the experiments, the PC will read back all the raw data from the FPGAs in every 5 s. We can compare the current read back data with the next data to see if there are some changes. According to our experiment, we find that the error bits can be categorized into following three types: 1) bit flips that can be read back continuously, 2) transient bit flips show up once, and 3) a large number of unexpected error bits in line. In the proposed design, if there are too many multiple error bits in a byte unit for controller to fix, the refresh controller will leave it alone and rewrite the correcting codes to avoid repeating detection. Hence, if there are errors in BRAMs that refreshing cannot correct, behaviors of errors should fall in the first category. Because the communication modules keep receiving and send data. It is more likely that errors in communication modules are refreshed by the ND instead of showing up in the location of the read back data continuously. Finally, if there are a large number of unexpected errors, it is most likely that a failure happened in the controller systems. Therefore, the errors in the proposed design can be identified if the errors are repeatedly presented in the read back data.

D. *Comparison Between Unhardened RAMs and the Self-Refresh RAMs*

The results of the neutron radiation experiments of the comparison between unhardened RAMs and the self-refresh RAMs are shown in Fig. 18. The number of bit flips in self-refresh ECC RAM remains at zero during the entire experiment, while the number of bit flips in unhardened RAM rises to 32 in the initial 1.5 h. As both RAMs are working in the same radiation environment, it proves that the design of self-refresh ECC RAM is effective for SEU mitigation.

As aforementioned, the size of the unhardened RAM was 32 kB, therefore, the generation rate of bit flips in this device in this radiation environment was about 1 bit/(kB $\cdot h$) or 1×10^{-3} bit/(N$\cdot h$), where N represents the number of memory units.

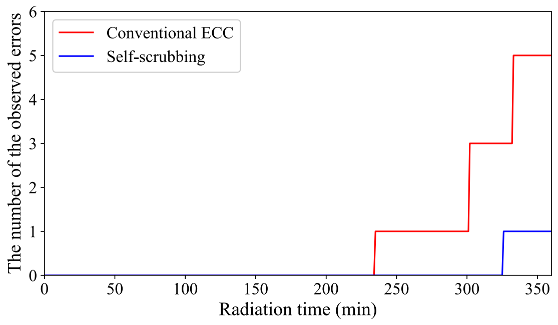

Fig. 21. Comparison between the conventional ECC RAMs and the self-refresh ECC RAMs.

TABLE IV
SEU Cross Sections of SRAM on Artix-7 FPGAs in Neutron Radiation Environments

Radiation facilities	Integrated flux (MeV)	Hardening	Cross section $(cm^2 \cdot bit^{-1})$
ChipIr (this paper)	10	None	2.22×10^{-14}
ChipIr (this paper)	10	ECC	5.79×10^{-16}
ChipIr (this paper)	10	SR*	1.16×10^{-16}
CERN [32]	20	None	3.18×10^{-14}
GENEPI2 [33]	14.2	None	1.2×10^{-14}
TTEA [34]	1	None	7.6×10^{-15}

*SR indicates self-refresh.

E. *Comparison Between the Conventional ECC RAMs and the Self-Refresh RAMs*

The comparison results between conventional ECC RAM and self-refresh RAM is shown in the Fig. 21. In this experiment, the conventional ECC RAM is the RAM hardened by Xilinx official ECC modules [31], which is used as the reference RAM. After the 360 min radiation experiment, the total number of observed errors in the conventional ECC RAM is five, whilst the number of the errors in the self-refresh RAM is only one.

Table IV gives the SEU cross section of BRAMs on Artix-7 FPGAs in different neutron radiation experiments. Despite the experiments were performed in different radiation environments, the SEU cross section of unhardened RAMs is about the same order of magnitude. Compared to the unhardened RAMs and conventional ECC RAMs, the self-refresh RAMs achieve better error mitigation performance in neutron radiation environments. In our experiment, the error rates of the self-refresh RAMs are 8.7×10^{-5} bit $\cdot$ (kB$\cdot$h)$^{-1}$ and the calculated SEU cross section is 1.16×10^{-16}cm$^2 \cdot$ bit^{-1}, which is one-fifth of the error rate of the ECC RAM.

VII. Conclusion

This article proposed a scheme that combines ECC and refreshing methods to mitigate SEUs for the devices that are supposed to work in extreme radiation environments. Compared with the conventional refreshing method, it can refresh memory units separately with high frequency without interrupt operations from user modules. The proposed scheme requires no additional RAM ports, so that it can be applied in a wide range of hardware systems. In addition, by modifying parallel architecture according to the density of radiation, this design can achieve a balance between performance and hardware costs.

The experiments were conducted in the neutron radiation environments. It is shown that the error rates remain robust irrespective of the RAM size. The comparison of the radiation experiments also shows that the self-refresh scheme is an effective strategy for hardening embedded system and the error rate of the self-scrubbing RAM is one-fifth of the conventional ECC RAM.

References

[1] R. H. Maurer, M. E. Fraeman, M. N. Martin, and D. R. Roth, "Harsh environments: Space radiation," *Johns Hopkins APL Tech. Dig.*, vol. 28, no. 1, pp. 17–29, 2008.

[2] R. Trivedi and U. S. Mehta, "A survey of radiation hardening by design (RHBD) techniques for electronic systems for space application," *Int. J. Electron. Commun. Eng. Technol.*, vol. 7, no. 1, pp. 75–86, 2016.

[3] L. Gonella *et al.*, "Total ionizing dose effects in 130-nm commercial CMOS technologies for HEP experiments," *Nucl. Instrum. Methods Phys. Res., Sect. A: Accel., Spectrometers, Detectors Assoc. Equip.*, vol. 582, no. 3, pp. 750–754, Dec. 2007.

[4] Y. Liu *et al.*, "Analysis of displacement damage effects on bipolar transistors irradiated by spallation neutrons," *Chin. Phys. B*, vol. 28, no. 6, 2019, Art. no. 67302.

[5] S. Saha, S. Ehsan, A. Stoica, R. Stolkin, and K. McDonald-Maier, "Real-time application processing for FPGA-Based resilient embedded systems in harsh environments," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2018, pp. 299–304.

[6] E. Petersen, *Single Event Effects in Aerospace*. Hoboken, NJ, USA: Wiley, 2011.

[7] E. Petritoli and F. Leccese, "Reliability andSEE mitigation in memories for space applications," in *Proc. IEEE Metrol. Aerosp. (MetroAeroSpace).*, 2016, pp. 136–140.

[8] S. Bianchi, R. Paggi, G. L. Mariotti, and F. Leccese, "Why and when must the preventive maintenance be performed?," in *Proc. IEEE Metrol. Aerosp.*, 2014, pp. 221–226.

[9] M. R. Rohanipoor, B. Ghavami, and M. Raji, "Improving combinational circuit reliability against multiple event transients via a partition and restructuring approach," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1059–1072, May 2020.

[10] J. Xiong, Z. Cheng, and Z. You, "On board computer subsystem design for the Tsinghua nanosatellite," in *Proc. 20th AIAA Int. Commun. Satell. Syst. Conf. Exhibit.*, 2002.

[11] J. Yao, W. Shaojun, M. Ning, and P. Yu, "A SEU test and simulation method for Zynq BRAM and flip-flops," in *Proc. 13th IEEE Int. Conf. Electron. Meas. Instrum.*, Oct. 2017, pp. 1–5.

[12] M. Franklin and K. Saluja, "Pattern sensitive fault testing of RAMs with built-in ECC," in *Proc. Dig. Papers Fault-Tolerant Comput.: 21st Int. Symp.*, 1991, pp. 385–392.

[13] J. Hong, J. Kim, S. Han, and E.-Y. Chung, "A locality-aware compression scheme for highly reliable embedded systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 3, pp. 453–465, Mar. 2018.

[14] P. Papavramidou and M. Nicolaidis, "Iterative diagnosis approach for ECC-based memory repair," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 2, pp. 464–477, Feb. 2020.

[15] T. Jiang, P. Huang, and K. Zhou, "Scrub unleveling: Achieving high data reliability at low scrubbing cost," in *Proc. Des., Automat. Test Europe Conf. Exhib.,* 2019, pp. 1403–1408.

[16] N. Kim and K. Choi, "A design guideline for volatile STT-RAM with ECC and scrubbing," in *Proc. ISOCC 2015 - Int. SoC Des. Conf.: SoC Internet Everything.*, 2016, pp. 29–30.

[17] G. Mayuga, Y. Sato, and M. Inoue, "Highly reliable memory architecture using adaptive combination of proactive aging-aware in-field self-repair and ECC," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 2, pp. 464–477, Feb. 2020.

[18] Microprocessors in ESA projects, Sep. 2020. [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Microprocessors

[19] M. Kumar, D. Digdarsini, N. Misra, and T. V. Ram, "SEU mitigation of rad-tolerant Xilinx FPGA using external scrubbing for geostationary mission," in *Proc. 4th Int. Conf. Signal Process. Integr. Netw.*, 2017, pp. 414–418.

[20] J.-Y. Lee *et al.*, "Heterogeneous configuration memory scrubbing for soft error mitigation in FPGAs," in *Proc. Int. Conf. Field-Programmable Technol.*, 2012, pp. 23–28.

[21] R. Giordano, S. Perrella, V. Izzo, G. Milluzzo, and A. Aloisio, "Redundant-configuration scrubbing of SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 9, pp. 2497–2504, Sep. 2017.

[22] A. M. Tosson, M. Anis, and L. Wei, "RRAM refresh circuit," in *Proc. 26th Ed. Great Lakes Symp.*, 2016, pp. 227–232.

[23] R. Glein, B. Schmidt, F. Rittner, J. Teich, and D. Ziener, "A self-adaptive SEU mitigation system for FPGAs with an internal block ram radiation particle sensor," in *Proc. IEEE 22nd Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2014, pp. 251–258.

[24] R.-S. Zhang, L.-Y. Xiao, X.-B. Cao, J. Li, J.-Q. Li, and L.-Z. Li, "A fast scrubbing method based on triple modular redundancy for SRAM-Based FPGAs," in *Proc. 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol.,* 2018, pp. 1–3.

[25] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 965–970, Aug. 2007.

[26] A. M. Keller and M. J. Wirthlin, "Benefits of complementary SEU mitigation for the LEON3 soft processor on SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 1, pp. 519–528, Jan. 2016.

[27] B. Aslam *et al.*, "Degradation measurement of commercial camera sensors under fast neutron beamline," in *Proc. SEE MAPLD*, 2019, pp. 1–16.

[28] Z. Khanam *et al.*, "Degradation measurement of kinect sensor under fast neutron beamline," in *Proc. Radiat. Effect Data Workshop*, 2019, pp. 1–5.

[29] L. Obermueller, C. Cazzaniga, S. Kulmiya, and C. Frost, "A fast neutron monitor based on single event effects in SRAMs using commercial off-the-shelf components," in *Proc. 18th Eur. Conf. Radiat. Effects Compon. Syst.,* 2018, pp. 1–5.

[30] C. Cazzaniga and C. D. Frost, "Progress of the scientific commissioning of a fast neutron beamline for chip irradiation," in *Proc. J. Phys.: Conf. Ser.*, 2018, pp. 1–4.

[31] 7 series FPGAs memory resources, [EB/OL], Jul. 4, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf

[32] G. Tsiligiannis *et al.*, "Radiation effects on deep submicrometer SRAM-based FPGAs under the CERN mixed-field radiation environment," *IEEE Trans. Nucl. Sci.*, vol. 65, no. 8, pp. 1511–1518, Aug. 2018.

[33] J. C. Fabero *et al.*, "Single event upsets under 14-MeV neutrons in a 28-nm SRAM-based FPGA in static mode," *IEEE Trans. Nucl. Sci.*, vol. 67, no. 7, pp. 1461–1469, Jul. 2020.

[34] Y. Nakazawa *et al.*, "Radiation study of FPGAs with neutron beam for comet phase-I," *Nucl. Instrum. Methods Phys. Res. Sect. A: Accel., Spectrometers, Detectors Assoc. Equip.*, vol. 936, pp. 351–352, 2019.