PAPER

An Error Correction Scheme through Time Redundancy for Enhancing Persistent Soft-Error Tolerance of CGRAs

Takashi IMAGAWA[†a)], *Nonmember*, **Masayuki HIROMOTO**[†], **Hiroyuki OCHI**[††], *and* **Takashi SATO**[†], *Members*

SUMMARY Time redundancy is sometimes an only option for enhancing circuit reliability when the circuit area is severely restricted. In this paper, a time-redundant error-correction scheme, which is particularly suitable for coarse-grained reconfigurable arrays (CGRAs), is proposed. It judges the correctness of the executions by comparing the results of two identical runs. Once a mismatch is found, the second run is terminated immediately to start the third run, under the assumption that the errors tend to persist in many applications, for selecting the correct result in the three runs. The circuit area and reliability of the proposed method is compared with a straightforward implementation of time-redundancy and a selective triple modular redundancy (TMR). A case study on a CGRA revealed that the area of the proposed method is 1% larger than that of the implementation for the selective TMR. The study also shows the proposed scheme is up to 2.6x more reliable than the full-TMR when the persistent error is predominant.

key words: *coarse-grained reconfigurable architecture, reliability, triple modular redundancy, immediate termination, error-critical period*

1. Introduction

As CMOS process technologies enter into the range of a few tens of nanometers, various phenomena that disturb the normal operation of LSI systems have become prominent. In particular, soft errors induced by high-energy particles, such as single-event upset (SEU) and single-event transient (SET) phenomena, have been receiving increasing attention. The impact of soft error is expected to become even larger in further scaled devices. The consideration of soft-error vulnerability will soon become a common practice even for consumer-oriented system designs, where the trade-off between cost (*e.g.*, chip area, power consumption) and quality (*e.g.*, performance, reliability) are critically important.

Coarse-grained reconfigurable arrays (CGRAs) are suitable for cost-effective implementation of reliability-aware LSI systems [1]. The reconfigurability significantly reduces the non-recurring engineering cost for designing specific chips, that is, ASICs to meet various reliability requirements. The reconfigurability also extends the lifetime of LSI systems because the reconfiguration makes it possible to avoid the use of known or developed faulty units in the array [2]. As has been studied, CGRAs are superior to their fine-grained counterparts, i.e., FPGAs, in terms of performance and energy efficiency [3], because of the word-wise operation and routing. The large configuration granularity reduces the size of configuration SRAM which is vulnerable to the soft errors. Therefore, the chance of soft errors in CGRAs is expected to be smaller than that in FPGAs. Our preliminary experiments show that the amount of configuration information in FPGAs is 10 to 100 times as large as that in CGRAs to implement application circuits. This result suggests that the circuits implemented on FPGAs are 10 to 100 times more susceptible to soft error than those on CGRAs. Therefore, adopting CGRAs instead of FPGAs can improve the soft-error resilience of the circuits. This advantage becomes more notable when the target applications are hardware accelerators that mostly execute word-wise operations.

One of the well-known methods to enhance the reliability is the triple modular redundancy (TMR). The TMR enhances fault tolerance at the cost of chip area. Recently, a reliability-aware CGRA was proposed in [4], which adopts TMR selectively to the part of a circuit. In an actual CGRA that implements a practical application, sufficient room for applying TMR may not be available. In order to maximize the reliability under such situations, a method that gives ordering of the circuit blocks in terms of the effectiveness for triplication was proposed [5].

Another approach to improve the reliability of LSI with lower area-overhead is to utilize time-redundancy [6], [7]. In previous time-redundancy techniques such as [8], the output of combinational circuit is latched at three different timing points to mask SET pluses. However, applying this method to datapaths of reconfigurable device does not ensure sufficient reliability because they use configuration SRAMs which are susceptible to soft errors. In another time-redundancy techniques, the same computation is repeated using the same computational resources after a certain period of time [9]. Reliability enhancement can be achieved even for large applications that occupy most of the computational resources. There are a lot of previous works that study time-redundancy technique on FPGAs such as [10]–[12]. Although these methods can improve the soft-error resilience drastically, some area overhead is still required to implement them because each reconfigurable component has to include the additional circuits.

In this paper, a novel error correction scheme that utilizes time-redundancy is proposed. It is particularly suit-

Manuscript received June 9, 2014.
Manuscript revised December 7, 2014.

[†]The authors are with the Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Kyoto-shi, 606–8501 Japan.

[††]The author is with the Department of Computer Science, College of Information Science and Engineering, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.

a) E-mail: paper@easter.kuee.kyoto-u.ac.jp

DOI: 10.1587/transele.E98.C.741

able to enhance the reliability of the hardware accelerators, such as stream processing circuit, which are the main application domain of CGRAs. Upon the observation that the soft errors tend to be persistent in the reconfigurable arrays, the proposed scheme immediately restarts execution as soon as an error is detected. The immediate termination of the running process lowers the probability of a subsequent fault which makes the error correction impossible, resulting in improved reliability of the circuit and the small area overhead. In this paper, a CGRA example, which is our main target architecture for implementing the proposed scheme, is used to explain the operations and evaluations, although the proposed scheme can be applied for other architectures such as FPGAs.

Circuit area and reliability comparisons are made with a straightforward implementation of time-redundancy, and a space-redundancy techniques. The area evaluation results show that the additional circuit that realizes the proposed scheme can be much smaller than other components of the CGRA circuits, and thus the proposed scheme is usable when there is a severe area limitation. The reliability evaluation shows that which redundancy strategy should be selected under given throughput and area constraints and target application circuits. Even in case triplicating all the circuits (full-TMR) is acceptable, the proposed scheme can be the best solution to enhance soft-error reliability when the persistent error is predominant.

The remainder of this paper is organized as follows. In Sect. 2, soft errors are classified based on error-continuity. Section 3 explains the implementations of the conventional and the proposed time-redundancy schemes. Sections 4 and 5 present area and reliability evaluations, respectively, using a CGRA example. Section 6 concludes this paper.

2. Soft Error

In this paper, soft errors are classified into either *transient* or *persistent* errors.

Transient error is an error that appears in the circuit only within the clock cycle when the fault occurs. The transient error does not produce error in the subsequent clock cycles, and the outputs of the circuit for those cycles contain no error.

Persistent error is an error that remains in the circuit or that produces other errors over multiple clock cycles. Once the persistent error occurs, the output of the circuit remains erroneous until the states of the circuit are reset or reloaded.

The SEU and SET can cause either *transient* or *persistent* error depending on where they occur. When an SEU occurs in a configuration memory of CGRA, its effect remains like a hard-error. Once the correct information is reloaded, such as by scrubbing [13], the error will be eliminated. An SET in a cyclic datapath of an application circuit also triggers *persistent* errors over several clock cycles [14]. The SEU and SET in these cases are *persistent*. On the other hand, *transient* error can be observed only in limited situations. The SET in an acyclic datapath is a representative example of *transient* errors.

The incidence ratio of *transient* and *persistent* errors in a circuit are determined by two factors. One is the incidence rates of SEU and SET themselves, and the other is the area ratio of circuit elements which induce *transient* and *persistent* errors. The former is analyzed and measured in previous works [15], [16], so that the soft-error rates in flip-flops and combinational circuits can be regarded to have the equivalent order of magnitude in the advanced process technologies. Therefore, in reconfigurable arrays, such as FPGA and CGRA, the area of the configuration memory and that of the acyclic data path defines the ratio between *persistent* and *transient* errors, respectively. The occupation area of configuration memory in CGRAs is still large, although it is smaller than that in FPGAs. When loops are formed in the data path, the area where *persistent* errors occur becomes even larger. Therefore, the soft errors in CGRAs tends to be *persistent*. Later in the evaluation section, we will quantitatively evaluate the area ratios using an example CGRA and its applications.

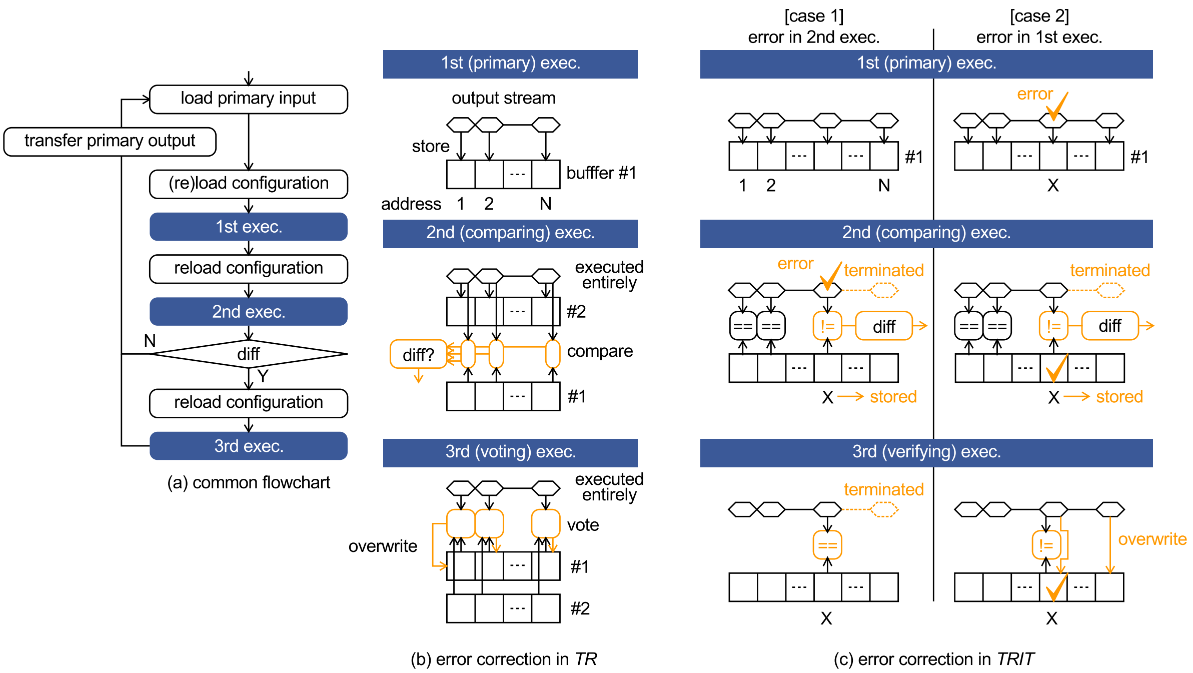
3. Error Correction Scheme Utilizing Time-Redundancy

Conventionally, reliability enhancement through time-redundancy is realized by repeating the same operations multiple times using the same hardware resource. An example implementation for a processor can be found in [9]. In general, time-redundancy requires almost no extra hardware resource, but throughput will be severely degraded.

3.1 Time-Redundancy Methods in CGRAs

In a general purpose processor with a time-redundancy method, a series of instructions between two checkpoints is executed for multiple times [17]. That concept can be applied to a CGRA that processes stream data by repeating a series of computational process multiple times. In this paper, a set of processes that is a unit for the time-redundancy repetition is called as *exec*. This repetition granularity in the proposed method is larger than those in the previous works that apply time redundancy to small portions of an entire process one by one.

To apply a time-redundancy method to a CGRA for stream processing, the CGRA should have a feature to reload configuration data to implement *scrubbing* [13] to prevent a persistent error in an *exec* from disturbing succeeding *exec*s. When the configuration data is reloaded, the internal state of the application circuit is also initialized to prevent a persistent error. In other words, an internal state after an *exec* is not stored to eliminate memory overheads. Then, the time-redundancy method can not be applied for control circuits and glue logics, because their primary outputs are decided not only by primary inputs but also by internal states of previous *exec*s. On the other hand, in some



(a) common flowchart

(b) error correction in *TR*

(c) error correction in *TRIT*

Fig. 1 Proposed *TR*/*TRIT* time-redundancy technique on CGRA.

```
while do
    load primary input
    load configuration infomation
    for $i = 1; i \leq N; i++$ do /* primary exec */
        Buf[1][$i$] = Exec($i$)
    end for
    load configuration infomation
    diff = false
    for $i = 1; i \leq N; i++$ do /* comparing exec */
        Buf[2][$i$] = Exec($i$)
        diff | = (Buf[1][$i$] != Buf[2][$i$])
    end for
    if diff then
        load configuration infomation
        for $i = 1; i \leq N; i++$ do /* voting exec */
            Buf[1][$i$] = vote(Buf[1][$i$], Buf[2][$i$], Exec($i$))
        end for
    end if
    send Buf[1] to the external system
end while
```

Fig. 2 Conventional time redundancy scheme. Buf[1] and Buf[2] correspond to buffer #1/#2 in Fig. 1, respectively. Exec(i) is the execution whose result will be stored in address i of the output memory.

hardware accelerators, the primary outputs are not affected by internal states of previous *exec*s. An example is an application in which divided image blocks are processed individually. Therefore, the following time-redundancy methods suit for CGRAs whose main targets are hardware accelerators rather than other architectures.

3.2 Conventional Time-Redundancy

Conventional time-redundancy (*TR*) technique works as in Fig. 1 (a), (b) and Fig. 2. The overall flowchart of the conventional time-redundancy (*TR*) technique is shown in Fig. 1 (a), and its formal description is given in Fig. 2. Figure 1 (b) illustrates how the error is detected and corrected in *TR*. In *TR*, the third run is conducted only when at least one mismatch is found between the first and the second runs. In *TR*, each run is executed entirely so that all errors are corrected by the voting unless the errors occur at the same address in two or more *exec*s out of three. Let the throughput of a non-redundant circuit and a selective TMR be 1.0, then that of *TR* method is 0.5 when all the data of *primary exec* and *comparing exec* matches, or 0.33 when there is a mismatch. When the error rate is very small, the average throughput becomes close to 0.5. As illustrated in Fig. 1 (b), as compared to non-redundant implementation, twice the amount of buffer is required in this method to store the all processing results of *primary exec* and *comparing exec*.

3.3 Time-Redundancy with Immediate Termination

If it is assumed that the errors in a circuit are mainly the *persistent* ones, we may be able to reduce the required buffer, to shorten the execution time for *comparing*/*voting exec*, and to enhance reliability for the *persistent* error. Let a mismatch due to error is found during the *comparing exec* at a certain address, say *X*, most of the outputs from *primary exec* and *comparing exec* do not match beyond address *X*. We cannot determine which output is correct unless the third run (*verifying exec*) is conducted. Hence, we propose to stop the *comparing exec* as soon as we find a mismatch, and immediately start the third run.

Here, we have an additional assumption that the error rate is sufficiently small such that we do not observe two error incidents during the three *exec*s. With this, it is possible to determine either the *primary exec* or the *comparing exec*

```
while  do
  load primary input
  load configuration infomation
  for $i = 1; i \leq N; i++$ do /* primary exec */
    Buf[1][$i$] = Exec($i$)
  end for
  load configuration infomation
  $X = -1$
  for $i = 1; i \leq N; i++$ do /* comparing exec */
    if Buf[1][$i$] != Exec($i$) then
      $X = i$
      break
    end if
  end for
  if $X \geq 0$ then /* verifying exec */
    load configuration infomation
    for $i = 1; i < X; i++$ do
      Exec($i$)
    end for
    tmp = Exec($X$)
    if Buf[1][$X$] == tmp then
      break
    else
      Buf[1][$X$] = tmp
      for $i = X + 1; i \leq N; i++$ do
        Buf[1][$i$] = Exec($i$)
      end for
    end if
  end if
  send Buf[1] to the external system
end while
```

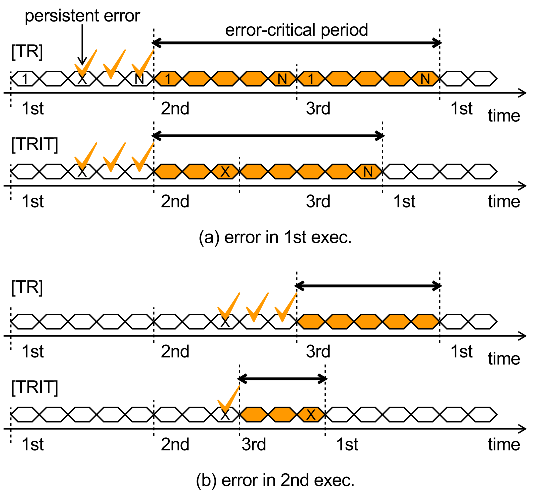
Fig. 3 Execution control for time-redundancy with immediate termination.

is correct by a single comparison on the data of the first mismatch. Figure 1 (c) illustrates how the proposed method, time-redundancy with immediate termination (*TRIT*), detects and corrects the error. At the point of error in the third *exec*, when it is found that the *primary exec* is correct, we stop the *verifying exec* and use the result of *primary exec* stored in the buffer. When the *comparing exec* is correct, then the results of the third run has to replace the contents of the buffer beyond address *X*. As could be understood in the figure, only one set of buffer is required in the proposed method as opposed to the conventional one that requires two sets of buffer. The procedure of the proposed method is summarized in Fig. 3.

The average throughput of this method becomes approximately 0.5 because both *primary exec* and *comparing exec* are fully performed in most *exec*s. In case an error is found, its throughput varies from 0.33 to 1.0, depending on where the mismatch occurs.

One may think the outputs of the proposed scheme is not reliable because the outputs beyond the error address is not be validated. However, opposed to this intuition, the proposed method is more reliable than to the simple time-redundancy when *persistent* error is dominant. This will be understood by considering the 'critical' period.

Figure 4 illustrates the reason why the *TRIT* is expected to be more reliable than the *TR* for the *persistent* error. The error-critical period illustrated in Fig. 4 is the interval when a soft error makes it impossible to correct the erroneous values by the time-redundancy techniques. When an initial *persistent* error occurs in the 1st *exec*, the 2nd and the 3rd *exec*s of both *TR* and *TRIT* cannot allow another *persistent* error. The total processing time of the *TRIT* for the 2nd and the 3rd *exec*s, i.e., the error-critical period, is shorter in the proposed scheme than that of the *TR* because the 2nd *exec* in the *TRIT* is immediately terminated upon a mismatch. In the case an initial error occurs in the 2nd *exec*, the 3rd *exec* in *TRIT* is also terminated, so that the error-critical period of the *TRIT* is shorter than that of *TR*.



(a) error in 1st exec.

(b) error in 2nd exec.

Fig. 4 Error-critical period in the time-redundancy techniques.

4. Circuit Area Evaluation

In the following two sections, the area overhead and reliability enhancement of the proposed scheme will be evaluated. The CGRA prototypes that use either *TR* or *TRIT* methods are implemented and synthesized using Verilog HDL for this purpose. A commercial tool and a 65-nm commercial library is used for the CGRA design. In addition, a CGRA that utilizes selective TMR is also synthesized. In these implementations, the configuration memory and the buffer in Fig. 1 are realized by a register file and an SRAM, respectively. The circuit area is evaluated by the number of equivalent 2-input NAND gates. It is assumed that the area of 1-bit SRAM cell is equal to that of a 2-input NAND gate in calculating SRAM area.

4.1 CGRA Architecture

The basic structure of the CGRA used for the evaluations is illustrated in Fig. 5. It consists of a two-dimensional cell array, data memory, and an array controller. Each cell is composed of a processing element (PE), a wiring resource, and a configuration memory that programs the functionality of the PE and the wiring connections. The PE executes arithmetic and logical operations. Data memories store the primary input and output of the application circuit implemented on the CGRA. The memory controller mainly defines a scheme for

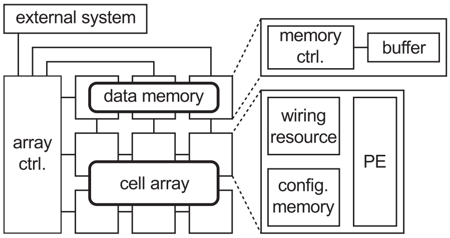


Fig. 5 Architecture overview of time-redundancy CGRA. This architecture is similar to many other CGRAs, but the array controller and memory controller include some operators for the time-redundancy methods.

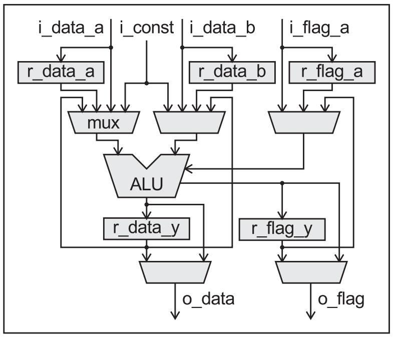


Fig. 6 PE architecture of the ALU cell.

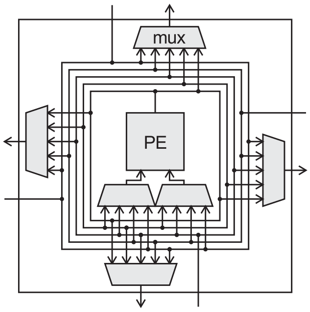


Fig. 7 Wiring resource architecture. This figure illustrates only wires and multiplexers for i_data_{a,b} in Fig. 6.

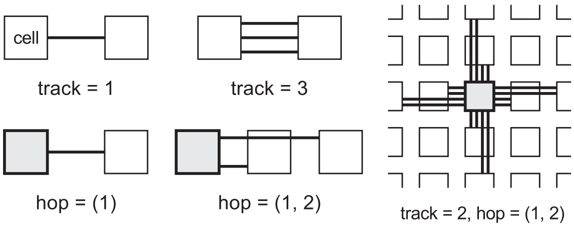


Fig. 8 The parameters of routing resources: *hop* and *track*.

Table 1 NAND2-equivalent area of ALU and MULT cells. The flag width is fixed to 1.

Condition			Cell area	
data width	track	hop	ALU	MULT
8	1	(1, 2)	2047.75	1488.75
8	2	(1, 2, 3)	4916.00	4341.75
16	1	(1, 2)	3483.75	2303.50
16	2	(1, 2, 3)	8195.00	6985.25

Table 2 NAND2-equivalent area of "array ctrl" in Fig. 5.

Condition		Area
	data width	array ctrl.
selective TMR	8	584.00
	16	1130.25
TR	8	620.75
	16	1164.75
TRIT	8	621.75
	16	1165.25

time-redundancy. The array controller serves as an interface to an external system and a manager of the process repetition in the time-redundancy.

Figure 6 illustrates the PE architecture of the ALU cell to help understand the area-impact introduced by the redundancy enhancement. There are two kinds of PE cells: ALU and MULT. The ALU executes arithmetic and logical operations with two data (i_data_{a,b}) and one flag (i_flag_a) inputs. In the MULT cell, a multiplier replaces the ALU. These PE cells include registers to store their input (r_data_{a,b}, r_flag_a) and output (r_{data,flag}_y). The ALU and multiplier can use a constant value (i_const) as an operand stored in a configuration memory.

As illustrated in Fig. 7, the wiring resource is composed of six word-width and five flag-width multiplexers (mux). The routing resource of the CGRA is defined by two parameters: *hop* and *track* (Fig. 8). The *hop* refers to the length of wires in the unit of the cell dimensions. The *track* is the number of wires for each hop. When *track* = 2 and *hop* = (1, 2), a cell is directly connected to eight nearby cells illustrated in the right of Fig. 8. These parameters have impacts on not just the routability but the reliability because large *hop* and *track* increase the number of inputs for the wiring resource and make its configuration memory large, and thus error susceptible area will become large.

4.2 Area Impact for Control Circuit

When a time-redundancy scheme is applied, execution control becomes more complex than the selective TMR implementations. Hence, the area overhead of the "array ctrl" and "memory ctrl" circuits in the time redundancy techniques has to be evaluated.

Tables 2 and 3 list the areas of the "array ctrl," "memory ctrl," and buffers in the different reliability enhancement schemes. Table 1 shows the total areas of an ALU and MULT cells. For the selective TMR, two memory control circuits are designed: one is to triplicate only the array-cells, and the other is to triplicate both data memories and the array-cells which is denoted as "with triplicated memory." When multiple buffers are required, such as in the case of *TR* implementation, the area values in Table 3 are multiplied values of a single buffer.

The area of control circuits of the time-redundancy techniques are always larger than those of the selective TMR for both "array ctrl" and "memory ctrl." However, the in-

Table 3 NAND2-equivalent area of “memory ctrl” in Fig. 5.

Condition			Area	
	data width	buffer words	memory ctrl.	buffer
selective TMR	8	1024	0.0	8192.0
	8	65536	0.0	524288.0
	16	1024	0.0	16384.0
	16	65536	0.0	1048576.0
selective TMR with triplicated memory	8	1024	43.03	24576.0
	8	65536	43.03	1572864.0
	16	1024	83.53	49152.0
	16	65536	83.53	3145728.0
TR	8	1024	102.75	16384.0
	8	65536	102.75	1048576.0
	16	1024	185.50	32768.0
	16	65536	185.50	2097152.0
TRIT	8	1024	144.25	8192.0
	8	65536	204.50	524288.0
	16	1024	166.50	16384.0
	16	65536	226.75	1048576.0

creased area is much smaller than the areas of an array-cell or an SRAM. For example, the circuit area of *TRIT* is 1% larger than that of the selective TMR, when the data width is 16 bit, the memory size is 1024, the *track* is 1, the *hop* is (1, 2) and the cell array is 4x4. The additional circuits to enhance reliability of the CGRAs with the time-redundancy techniques are negligibly small.

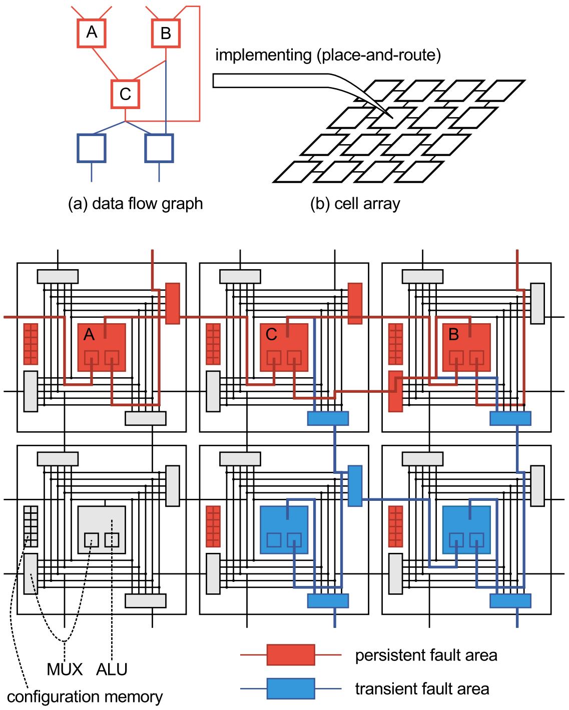
4.3 Ratio between Persistent and Transient Errors

The ratio between *persistent* and *transient* errors is important because it determines the effectiveness of the proposed *TRIT* scheme. The *TRIT* is particularly preferable when the *persistent* error dominates the *transient* error.

The circuit regions that cause *persistent* and *transient* errors are defined as *persistent error regions* and *transient error regions*, respectively. Using an example data flow graph in Fig. 9, we will quickly explain how the *persistent error regions* and *transient error regions* are defined. In the graph, nodes B and C form a cyclic datapath, so all regions in the layout that correspond to these nodes are considered as *persistent error region*. When an error occurs at node A, the input of C may contain error with a high probability. The error in the input of a node results in *persistent* error at the output, even if the original error is *transient*. Hence, the area that corresponds to node A is classified as *persistent error region*. The other areas are classified as *transient error region* except for the configuration memories which is a *persistent error region* as stated in the earlier section. Figure 9 (c) shows a region assignment for the example data flow graph.

As described in the last paragraph of Sect. 2, the ratio of occurrence probability of *persistent* and *transient* errors are determined by the area ratio of *persistent error region* and *transient error region*. Hereafter, the area ratio is denoted as $S_{\rm pt}$ which are defined by the area of *persistent error region* divided by that of *transient error region*.

The $S_{\rm pt}$ values are calculated for five example applications based on the results of automated place-and-route to the CGRA described above. The applications are: a 1024-point FFT, an 8-tap FIR filter, a color invert filter, a horizontal-differential filter, and an edge detection filter. In this evaluation, the area of wire is assumed to be negligible but its composition influences the amount of configuration memory used in the circuit. The $S_{\rm pt}$ value become very high for the FFT circuit, in which entire array is repeatedly used. It is expected that there are many applications for which the proposed reliability enhancement by *TRIT* is effective.



(a) data flow graph

(b) cell array

(c) classification of persistent/transient fault area

Fig. 9 Example of *transient* and *persistent error region*.

Table 4 $S_{\rm pt}$ values for sample application circuits on the CGRA. If the value is larger than 1, *persistent* error is dominant in the circuits. “—” means that a circuit can not be implemented on a target CGRA because of routing resource shortage.

	routing resource parameter (track, hop)			
application	1, (1, 2)	2, (1)	2, (1, 2)	2, (1, 2, 3)
color invert filter	2.91	3.09	3.18	2.69
horizontal-differential filter	1.91	1.63	1.83	1.64
edge detection filter	—	—	1.21	0.90
8-tap FIR	0.96	0.81	0.89	0.90
1024-FFT	—	—	239.65	158.52

As Table 4 shows, the relationship between the routing resource parameters and $S_{\rm pt}$ is not straightforward. Hence, the place-and-route and the area ratio calculation should be performed to know which error mode is dominant, to determine which time-redundancy technique, *TR* or *TRIT*, should be adopted for the target routing parameter and the application circuit.

5. Reliability Evaluation

5.1 Evaluation Setup

The circuit reliability is quantitatively evaluated by using Monte Carlo simulations. In order to compare reliability of the circuits that use different schemes, we use the amount of successfully processed data until the first failure of the circuit (hereafter, “ASPD metric” in short) is used as a reliability metric. Mean time to failure (MTTF) is not appropriate in this evaluation because the throughput of the time-redundancy is less than that of the normal implementations, such as space-redundancy techniques.

Besides the *transient* and *persistent* errors defined in Sect. 2, the *unrecoverable* mode error which represents hard-errors such as time dependent dielectric breakdown (TDDB) is also taken into account to evaluate the reliability in the field. The *unrecoverable* error cannot be recovered even by a configuration reloading. The *transient*, *persistent*, and *unrecoverable* errors follow Poisson models, and their incidence probabilities per a unit of time are denoted as λ_t, λ_p and λ_u, respectively.

The duration of one *exec*, which is equal to the reconfiguration interval in this evaluation, is denoted as N. In the following evaluations, the application circuits are assumed to output the processing result in every clock cycle, hence the total number of output is equal to N.

5.2 Reliability Improvements

First, the ASPD metrics of *TR*, *TRIT*, and selective TMR schemes for each error mode are evaluated and compared for $10^{-16} \leq \lambda_{t,p,u} \leq 10^{-3}$ and $10^2 \leq N \leq 10^5$. For the selective TMR, the range of the circuit triplication is varied from applying no-redundancy to full-TMR.

Figure 10 shows the ASPD metrics when $N = 10^3$ and $10^{-6} \leq \lambda_{t,p,u} \leq 10^{-4}$. An average of 100 trials are shown. The horizontal axis shows the circuit area overhead, where that of the circuit without redundancy is 1.0, and that of the fully-triplicated circuit is 3.0. According to the results in the previous section, the overhead of the time-redundancy techniques is slightly greater than but very close to 1.0. In terms of the ASPD metric, a partial triplication is not so effective unless the circuit is triplicated almost entirely. On the other hand, the time-redundancy techniques achieve improvements with a slight area overhead regardless of $\lambda_{t,p}$.

When the reliability of these circuits is compared, their area overhead and throughput should be equivalent. When the parallelization to make a clone of the target circuit is acceptable, the area overhead and the throughput of the circuits with the time-redundancy techniques are equal to those of the half-triplicated circuits.

The parallelization keeps the reliability of the time-redundancy techniques because their MTTF are half. Therefore, the time-redundancy techniques are more reliable for *transient* and *persistent* errors than the selective TMR when their area overhead and throughput are equivalent. This trend is more prominent when $\lambda_{t,u}$ is small. In contrast, the reliability of circuits with the time-redundancy techniques for the *unrecoverable* error becomes worse than that of no-redundancy circuits. This is because their throughputs are less than 1.0 and they can not mask even one *unrecoverable* error. Comparing the two time-redundancy techniques, the *TRIT* is less reliable than *TR* for the *transient* error as expected. In contrast, the *TRIT* is 1.4x more reliable than *TR* for the *persistent* error regardless of N and λ_p. This is explained by the error-critical period, which is shorter for *TRIT* than *TR* (Fig. 4).

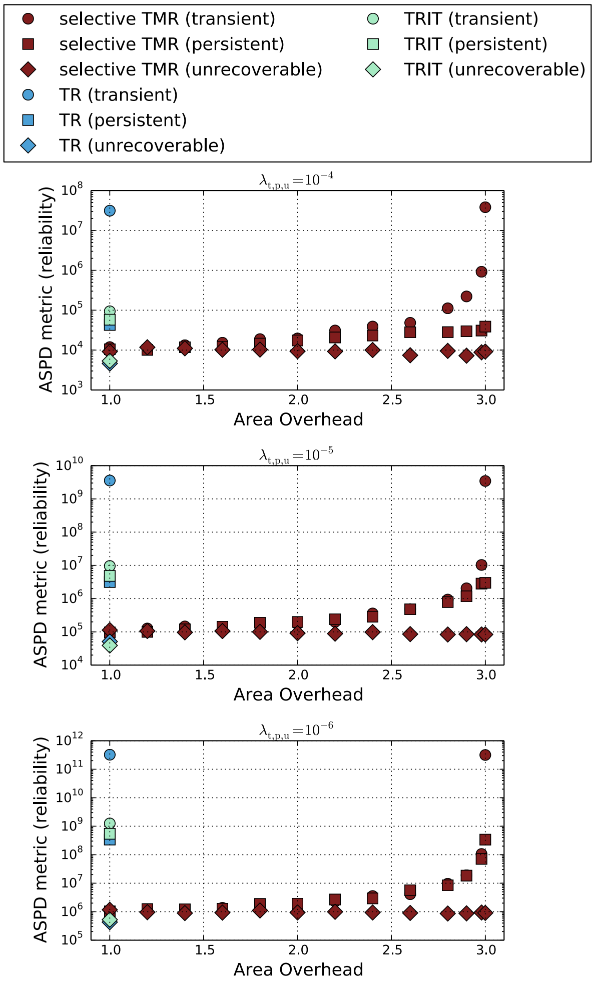


Fig. 10 Reliability as functions of area overhead for each error mode.

The *TR* can be expanded to repeat an *exec* more than three times, and it is expected to be more reliable than the *TRIT* when enough number of repetition times are acceptable. Figure 11 shows the reliability of *TRIT* and *TR* whose upper limits of repetition (deadline) are 3, 4, and 5. The *TR* can be more reliable than the *TRIT* only when five times and more repetitions and the buffer overhead are acceptable. When the reliability itself is highly important, for example in mission-critical applications, the *TR* is more appropriate than the *TRIT*. On the other hand, when the trade-off be-

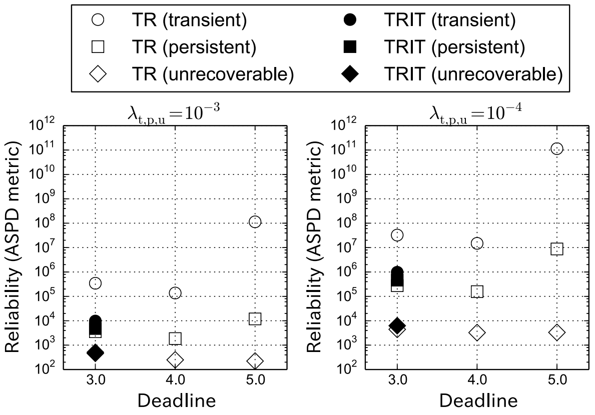


Fig. 11 Reliability comparison between *TR* and *TRIT* as functions of repetition limit (deadline) (N = 10^3).

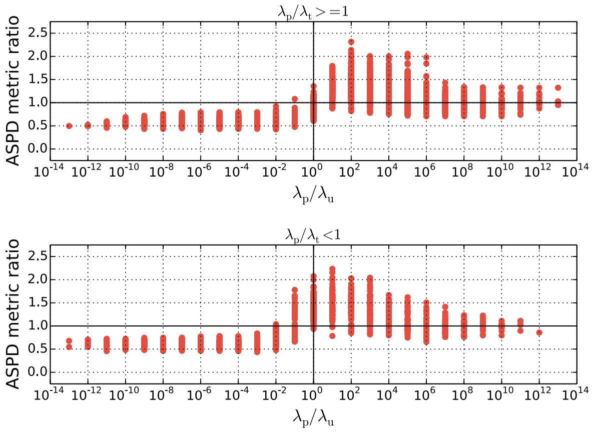


Fig. 12 ASPD metric ratio between *TR* and full-TMR. If a value of the vertical axis is larger than 1.0, *TR* is more reliable than full-TMR. When the value of horizontal axis λ_p/λ_u is larger than 1, *persistent* error is more frequent than *unrecoverable*. Note that this value rarely become less than 1 in practical cases. The upper graph shows the results when the *persistent* error dominates the *transient*.

tween cost and reliability is more important than reliability itself, the *TRIT* is an attractive option to enhance reliability efficiently because its area overhead is negligibly small.

Next, reliability of the redundancy techniques are compared considering all types of errors, *i.e.*, *transient*, *persistent* and *unrecoverable*. The ratio of *persistent* error to *transient* error (λ_p/λ_t) has been changed from 10^{-2} to 10^3. The upper bound is determined by the results of example implementations in Table 4. The lower bound is based on the area ratio between (configuration memory) and (wiring resource + PE) in Fig. 5, at which the target application includes no cyclic datapath and the *persistent error region* becomes the smallest in the assumed CGRA.

Figures 12, 13 and 14 show the ASPD metric ratio between full-TMR, *TR*, and *TRIT*. The reliability of *TR* and *TRIT* tend to be higher as the λ_u is lower, and they are up to 2.31x and 2.60x larger than that of the full-TMR, respectively.

Figure 12 shows that the *TR* is equally or more reliable than the full-TMR unless the *unrecoverable* error (hard error) is more frequent than the other error modes (soft error). As Figs. 13 and 14 shows, when the *persistent* error is dominant ($\lambda_p/\lambda_t \geq 1$ and $\lambda_p/\lambda_u \geq 1$), the *TRIT* achieves higher reliability than the full-TMR implementation, and is equally or more reliable than the *TR* with less amount of the buffer. Therefore, the *TRIT* is the best way to enhance soft-error reliability when the *persistent* error dominates other error modes.

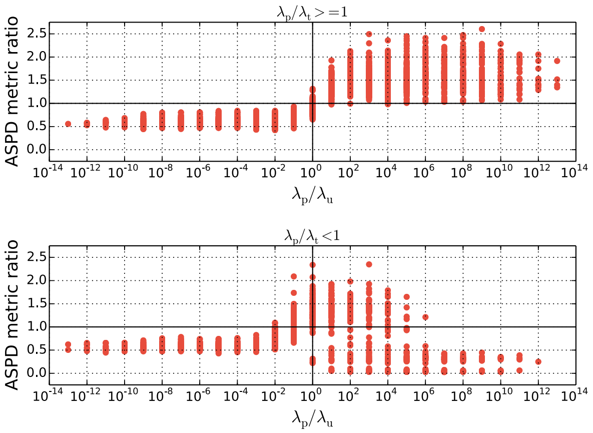


Fig. 13 ASPD metric ratio between *TRIT* and full-TMR. If a value of the vertical axis is larger than 1.0, *TRIT* is more reliable than full-TMR.

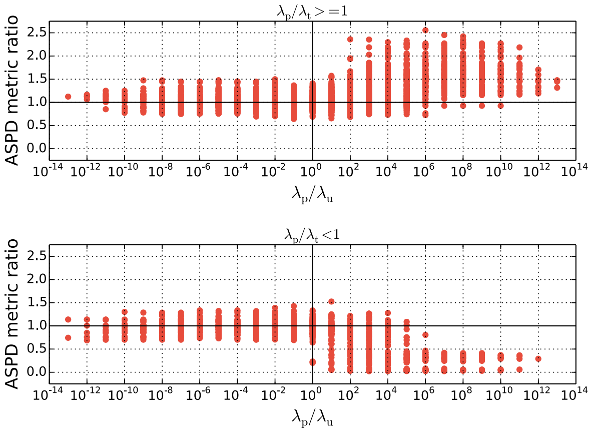


Fig. 14 ASPD metric ratio between *TRIT* and *TR*. A value of vertical axis is a reliability ratio of the former to the latter. If the value is larger than 1.0, *TRIT* is more reliable than *TR*.

5.3 Strategy for Selecting Reliability-Enhancement Method

Based on the above results and the given design constraints, it is possible to determine which redundancy techniques should be utilized for a CGRA circuit.

When the parallelization of the target application is unacceptable because of the area constraints, either the time-redundancy techniques or the selective TMR should be used to satisfy a throughput constraint. For example, the time-redundancy techniques, that is, *TR* and *TRIT* should be adopted if the performance degradation is acceptable. Conversely, when the throughput is as important as reliability, the circuit should be partly triplicated. In contrast, when the parallelization is permitted, either of the time-redundancy techniques, *TR* and *TRIT* should be used based on S_{pt}, i.e., whether *transient* or *persistent* is dominant. The area ratio can be calculated by the result of place-and-route. In both techniques, an application circuit can be implemented with the same place-and-route result. Therefore, a designer can decide which technique is applied after the place-and-route and calculating the area ratio. For example, in the sample applications in Sect. 4, *TRIT* is suitable for the all circuits except the 8-tap FIR filter and the edge detection filter when *track* is 2 and *hop* is (1, 2, 3).

6. Conclusion

This paper proposes the *TRIT* method which efficiently corrects persistent soft errors using the framework of a time-redundancy. In this method, the running process is immediately terminated when a mismatch is found between the results of two identical runs. The immediate termination improves the reliability of the circuit by shortening the error-critical period, during which time the circuit becomes vulnerable for the persistent soft-error that is a predominant error in reconfigurable architectures. In an example application, the soft-error reliability of the proposed method become 2.6x better than that of full-TMR with negligibly small area overhead.

Acknowledgements

This work is partly supported by Grant-in-Aid for JSPS Fellows 22·6265, JSPS KAKENHI Grant-in-Aid for Scientific Research (B) Grant number 26280014, and by VDEC, the University of Tokyo in collaboration with Synopsys, Inc. and Mentor Graphics, Inc.

References

[1] T. Imagawa, M. Hiromoto, H. Ochi, and T. Sato, “Reliability evaluation environment for exploring design space of coarse-grained reconfigurable architectures,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E93-A, no.12, pp.2524–2532, Dec. 2010.

[2] Z.E. Rakosi, M. Hiromoto, H. Ochi, and Y. Nakamura, “Hot-swapping architecture extension for mitigation of permanent functional unit faults,” Proc. International Conference on Field Programmable Logic and Applications (FPL), pp.578–581, Aug. 2009.

[3] Zain-ul-Abdin and B. Svensson, “Evolution in architectures and programming methodologies of coarse-grained reconfigurable computing,” Microprocessors and Microsystems, vol.33, no.3, pp.161–178, May 2009.

[4] D. Alnajjar, Y. Ko, T. Imagawa, H. Konoura, M. Hiromoto, Y. Mitsuyama, M. Hashimoto, H. Ochi, and T. Onoye, “Coarse-grained dynamically reconfigurable architecture with flexible reliability,” Proc. International Conference on Field Programmable Logic and Applications (FPL), pp.186–192, Aug. 2009.

[5] T. Imagawa, H. Tsutsui, H. Ochi, and T. Sato, “A cost-effective selective TMR for heterogeneous coarse-grained reconfigurable architectures based on DFG-level vulnerability analysis,” Proc. Design, Automation and Test in Europe (DATE), pp.701–706, March 2013.

[6] B.W. Johnson, Design and Analysis of Fault Tolerant Digital Systems, Addison-Wesley Longman Publishing, 1988.

[7] D.K. Pradhan, Fault-Tolerant Computer System Design, Prentice-Hall, 1996.

[8] M. Nicolaidis, “Time redundancy based soft-error tolerance to rescue nanometer technologies,” Proc. 17th IEEE VLSI Test Symposium 1999, pp.86–94, April 1999.

[9] A. Ejlali, B.M. Al-Hashimi, M.T. Schmitz, P. Rosinger, and S.G. Miremadi, “Combined time and information redundancy for SEU–tolerance in energy-efficient real-time systems,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.14, no.4, pp.323–335, April 2006.

[10] K. Nakahara, S. Kouyama, T. Izumi, H. Ochi, and Y. Nakamura, “Fault tolerant dynamic reconfigurable device based on EDAC with rollback,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E89-A, no.12, pp.3652–3658, Dec. 2006.

[11] D.G. Mavis and P.H. Eaton, “SEU and SET mitigation techniques for FPGA circuit and configuration bit storage design,” Proc. 3rd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD), Sept. 2000.

[12] F. Lima, L. Carro, and R. Reis, “Designing fault tolerant systems into SRAM-based FPGAs,” Proc. IEEE/ACM Design Automation Conference (DAC), pp.650–655, June 2003.

[13] C. Carmichael, M. Caffrey, and A. Salazar, Correcting Single-Event Upsets Through Virtex Partial Configuration, Xilinx Corporation, Tech. Rep. XAPP216 v1.0, June 2000.

[14] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M.J. Wirthlin, “Improving FPGA design robustness with partial TMR,” Proc. 2006 IEEE International Reliability Physics Symposium (IRPS), pp.226–232, March 2006.

[15] J. Yao, Z. Ye, M. Li, Y. Li, R. Schrimpf, D. Fleetwood, and Y. Wang, “Statistical analysis of soft error rate in digital logic design including process variations,” IEEE Trans. Nucl. Sci., vol.59, no.6, pp.2811–2817, Dec. 2012.

[16] S. Jagannathan, T. Loveless, B. Bhuva, N. Gaspard, N. Mahatme, T. Assis, S.-J. Wen, R. Wong, and L. Massengill, “Frequency dependence of alpha-particle induced soft error rates of flip-flops in 40-nm CMOS technology,” IEEE Trans. Nucl. Sci., vol.59, no.6, pp.2796–2802, Dec. 2012.

[17] R. Melhem, D. Mossé, and E. Elnozahy, “The interplay of power management and fault recovery in real-time systems,” IEEE Trans. Comput., vol.53, no.2, pp.217–231, Feb. 2004.

Takashi Imagawa received his B.E. degree in Electrical and Electronic Engineering, his master degree in Communications and Computer Engineering, from Kyoto University in 2008 and 2010. Presently, he is a doctor course student at Department of Communications and Computer Engineering, Kyoto University. He is a student member of IPSJ and IEEE.

Masayuki Hiromoto received B.E. degree in Electrical and Electronic Engineering and M.Sc. and Ph.D. degrees in Communications and Computer Engineering from Kyoto University in 2006, 2007, and 2009 respectively. He was a JSPS research fellow from 2009 to 2010, and with Panasonic Corp. from 2010 to 2013. In 2013, he joined the Graduate School of Informatics, Kyoto University, where he is currently an assistant professor. His research interests include VLSI design methodology, image processing and pattern recognition. He is a member of IEEE and IPSJ.

Hiroyuki Ochi received his B.E., M.E., and Ph.D. degrees from Kyoto University in 1989, 1991, and 1994, respectively, all in Engineering. From 1994 to 2004, he was an Associate Professor with Hiroshima City University, and from 2004 to 2013, he was an Associate Professor with Kyoto University. In 2013, he joined Ritsumeikan University as a Professor. His research interests include low-power/reliability-aware VLSI design and reconfigurable architectures. He is a member of IPSJ, IEEE, and ACM.

Takashi Sato received B.E. and M.E. degrees from Waseda University, Tokyo, Japan, and a Ph.D. degree from Kyoto University, Kyoto, Japan. He was with Hitachi, Ltd., Tokyo, Japan, from 1991 to 2003, with Renesas Technology Corp., Tokyo, Japan, from 2003 to 2006, and with the Tokyo Institute of Technology, Yokohama, Japan. In 2009, he joined the Graduate School of Informatics, Kyoto University, Kyoto, Japan, where he is currently a professor. He was a visiting industrial fellow at the University of California, Berkeley, from 1998 to 1999. His research interests include CAD for nanometer-scale LSI design, fabrication-aware design methodology, and performance optimization for variation tolerance. Dr. Sato is a member of the IEEE and the Institute of Electronics, Information and Communication Engineers (IEICE). He received the Beatrice Winner Award at ISSCC 2000 and the Best Paper Award at ISQED 2003.