Double Moduler Redundancy Design of LSI Controller for Soft Error Tolerance

Katsutoshi Otsuka Kazuhito Ito

Graduate School of Science and Engineering
Saitama University
Saitama 338-8570, Japan

Abstract— A soft error in LSI is a temporary malfunction in which stored data or signals are flipped. Redundancy is used to correct soft errors. Double modular redundancy performs computation execution and data recording in duplicate, detects soft errors through comparison, and corrects errors by re-executing the computation. It is preferable in terms of LSI area and power consumption compared to triple modular redundancy. While many studies have been conducted on redundancy in LSI datapaths, there have been few reports on double modular redundancy in LSI control units. In this paper, a double redundancy design for LSI controllers is proposed.

I. Introduction

When neutrons caused by cosmic rays enter a large scale integrated circuit (LSI), the signal value in the circuit is reversed if the energy exceeds a certain threshold. This is called a soft error [1, 2]. Due to the miniaturization and lower voltage of semiconductor devices, the signal energy in LSIs is reduced, and the threshold value is lowered accordingly. Furthermore, as the number of devices increases due to larger scale circuits, the probability of soft errors occurring in LSIs increases, and the probability of LSI malfunctions due to soft errors is getting higher.

Redundancy is known as a soft error countermeasure. Triple modular redundancy (TMR) uses three systems of modules to perform the same calculation and store data (calculation results) in triplicate, and takes majority vote [3]. Even if there is an error in either the calculation or the data, the error can be corrected by majority vote and subsequent calculations can be continued correctly. However, TMR has the disadvantage that it requires three times the circuit size and power consumption.

Double modular redundancy (DMR) detects errors by performing the same calculations on two modules and comparing the results. If the results do not match, an error has occurred, and the error is corrected by re-executing the operation that may contain the error, and then the subsequent operations are continued [4, 5, 6]. Although there is a delay time required to re-execute operations for error correction, it has the advantage of smaller circuit size and power consumption than TMR.

Generally, a digital system is divided into a data processing section (datapath) and a control section (controller). Since the controller is also implemented using LSI, soft errors may occur in the controller as well. While many studies have been conducted on datapath soft error countermeasures, including TMR and DMR [7, 8, 9], there are few reports on controller soft error countermeasures. In this study, we propose a countermeasure against soft errors in the controller using DMR.

The remainder of the paper is organized as follows. The soft-error model and redundancy for the error detection and correction are reviewed in Sect. 2. The proposed method for DMR controller is presented in Sect. 3. Experimental results are presented in Sect. 4 and Sect. 5 concludes the work.

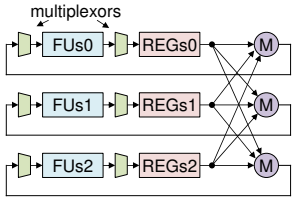

Fig. 1. Triple modular redundancy (TMR).

II. Error collection and register usage in DMR

A. Error model

Based on the principle and probability of soft error occurrence, we assume the following model for soft errors in LSI.

- Only one soft error occurs within one LSI at a time.
- Soft errors occur uniformly regardless of whether it is a combinational logic circuit or a flip-flop.
- The impact of soft errors is limited to one module (functional unit or register).
- Once a soft error occurs, it will not occur for a sufficiently long time in the same LSI.
- Soft errors in combinational circuits do not persist beyond the trigger of the clock signal.

B. TMR

The datapath with triple module redundancy [3] is illustrated in Fig. 1. The results of operations in functional units (FUs) are selected by multiplexers (MUXs) and stored in registers (REGs). Data is read from the REGs and selected by multiplexers (MUXs) to be used as input data for the FUs. FUs, REGs, and MUXs are tripled, and a tripled majority voter (M) is inserted between the REGs output and the input MUXs of the FUs. An error in FUs, REGs, or MUXs causes one of the REGs to have an incorrect value. However, errors are corrected by selecting non-error values by M and providing them to the FUs.

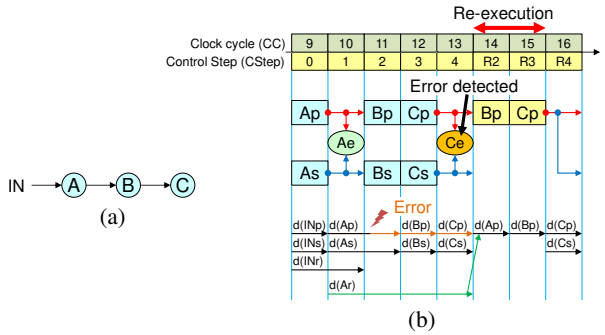

Fig. 2. An example for error correction by replay in DMR. (a) a DFG. (b) replay the operations when an error is detected.

C. *Error Correction in DMR by Replay*

Figure 2(a) shows an example data-flow graph (DFG), where a node represents an operation and an edge represents data dependency between operations. With DMR, each operation is executed twice and these are called respectively the *primary* and *secondary* executions of the operation. They are denoted as 'Ap' and 'As' for operation A. Figure 2(b) shows the schedule of operation executions and comparisons. Let d(G*m*) denote the result produced by G*m* for operation G ($m = $ p or $m = $ s). For example, Ap and As start in control step (CStep) 0 at clock cycle (CC) 9, both Ap and As take one CC, and d(Ap) and d(As) are stored in registers at the end of CC 9. d(Ap) and d(As) are read from the registers and compared their equality by the comparison 'Ae' in CStep 1 at CC 10.

When Ce detects an error as illustrated in Fig. 2(b), the error exists either in the execution of Bp, Bs, Cp or Cs, or in the data d(Ap), d(As), d(Bp), or d(Bs) (in this example, the error occurred in d(Ap) in CC 2). The error is corrected by executing the necessary operations again [10]. This is called *replay*. Replay of Bp requires the error-free input data of A. Thus another copy of the result of A denoted as d(Ar) is kept in a register not affected by the error in d(Ap) or d(As). Due to the error model of at most one operation or one data contains an error at a time, when an error is detected by Ce, d(Ar) is ensured to be error-free. Let such data for replay be called *replay input*. The replay input d(Ar) is copied to the primary register at the end of CC 13, and Bp is replayed using the data in CStep R2 at CC 14. There is another way to use the replay input and it is discussed in Sect. 3. Cp is then replayed in Cstep R3 at CC 15 using the replayed d(Bp) as shown in Fig. 2(b). At the end of the replay, d(Cp) is copied as d(Cs), and the normal DMR operation resumes.

The execution of operations are delayed by the replay when an error is detected. It is called *delay penalty* and denoted as P_d. In real time applications, an upper tolerance limit is imposed on the delay penalty.

D. *Redundant design of controller*

Generally, a circuit consists of a datapath that performs calculations and a controller that controls the datapath. Just like the datapath, the controller can also experience soft errors. Figure 3 shows an outline of the (non-redundant) circuit. The controller consists of a register CStep and combinational circuits

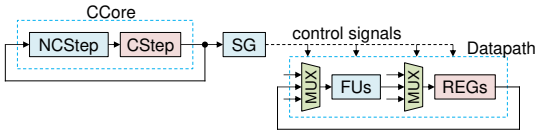

Fig. 3. Processing system consisting of controller and datapath.

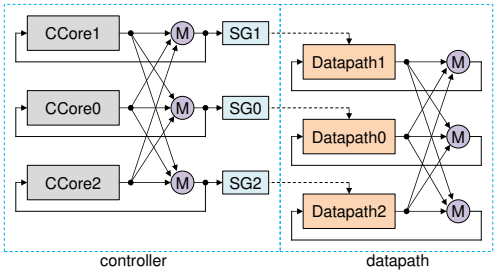

Fig. 4. Full TMR configuration of controller and datapath.

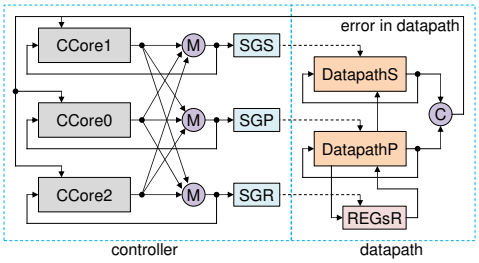

Fig. 5. The configuration of TMR controller and DMR datapath.

NCStep and SG. CStep stores the current control step in the schedule of processing execution. NCStep calculates the next control step based on the current CStep value. The SG is a combinational circuit that generates control signals for the datapath in order to cause the datapath to perform operations determined by the processing execution schedule in each control step. CStep and NCStep are called controller core (CCore).

The datapath consists of functional units (FU), register (REG), and multiplexers (MUX).

D.1. *Full TMR*

Figure 4 shows a full TMR for controllers and datapaths. CCore, SG, and datapaths are all tripled, and the CStep output in the controller and the REG output in the datapath are determined by the majority voter M. Compared to the non-redundant case, the area of the controller and datapath is three times larger, and the area of the majority voters is also added.

D.2. *TMR controller and DMR datapath*

Figure 5 shows a method of configure the datapath in DMR to reduce the area of the datapath. The duplicated datapaths are DatapathP for primary execution and DatapashS for secondary execution. Comparator C compares the values of the primary and secondary registers, and if they do not match, an error exists in the datapath. Register REGsR is used for replay input. The control signals for DatapathP, DatapathS, and REGsR are generated by the combinational circuits SGP, SGS, and SGR, respectively. Since the controller needs to control replay execution in addition to normal execution, the complexity of the

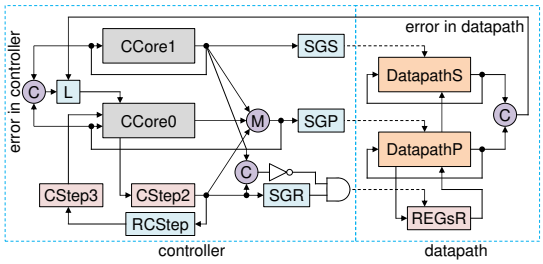

Fig. 6. DMR configuration of controller and datapath.

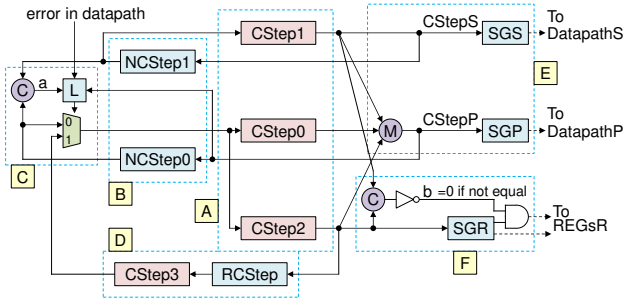

Fig. 7. The proposed architecture of DMR controller.

controller increases compared to the full TMR case, and the area increases accordingly. The effect of increased controller area is tripled in TMR controller.

III. The Proposed Method

A. DMR controller

Figure 6 shows a proposed configuration of redundancy, where not only the datapath but also the controller is configured with DMR. The detail of the proposed DMR controller is shown in Fig. 7. The registers CStep are tripled, and are called CStep0, CStep1, and CStep2. The combinational circuit NCStep that calculates the next CStep value is duplicated and is called NCStep0 and NCStep1. Replay of the datapaths is controlled using CStep0 and NCStep0. Generally, the number of bits of CStep0 is larger than CStep1 and CStep2 because it represents CStep for replay. Furthermore, NCStep0 not only calculates the CStep value for normal processing but also calculates the CStep value for replay, so the circuit scale of NCStep0 is larger than that of NCStep1. In normal operation, the output of NCStep0 is used as the next value of CStep0 and CSep2, and the output of NCStep1 is used as the next value of CStep1. The majority vote of CStep0, CStep1, and CStep2 is duplicated and used as inputs of NCStep0 and NCstep1, respectively. The majority vote results are CStepP and CStepS, and control signals for the datapaths, DatapathP and DatapathS, are generated by combinational circuits SGP and SGS, respectively. The value of CStep2 is input to the combinational circuit SGR, which generates control signals for the replay input registers REGsR.

CStep3 is a register that records the value of replay start CStep, and the value is obtained by the combinational circuit RCStep. When the result of comparing the outputs of NCStep0 and NCStep1 (marked as ‘a’ in the figure) shows a mismatch, in order to start replay from the next CC, the multiplexer is controlled to assign the value of register CStep3 to CStep0 and CStep2. As a result, CStep0 and CStep2 will have the same value, and CStep1 will have a different value, but CStepP is equal to CStep0 by the majority vote. Hence during replay execution, a control signal for replay is generated by SGP and given to DatapathP based on the same value as CStep0. In the case of a datapath error, replay is started in the same way as in the case of an error in NCStep0 or NCStep1. Therefore, the multiplexer selection signal is determined by the combinational circuit L based on the comparison result ’a’, the error detection signal of the datapath, and the value of CStep0 (to distinguish whether normal operation or replay is in progress).

B. Responding to controller errors

The response to soft errors in each area of the controller will be explained according to Fig. 7. Note that based on the error model, if an error occurs in any area, it is guaranteed that there is no error in other areas.

Area A: An error in CStep0 is corrected by majority vote. An error in CStep1 is detected by the comparison of the outputs of NCStep1 and NCStep0, and replay is activated. If there is an error in CStep2, CStepP and CStepS are correct and the datapath operates normally. REGsR may contain erroneous data but the values are not used and will be overwritten by correct value before a next error occurs.

Area B: If there is an error in NCStep0 or NCStep1, the comparison result ‘a’ indicates a mismatch. Then the multiplexer transfers the value of CStep3 to CStep0 and CStep2, and replay is started. When an error occurs in NCStep0 or NCStep1, it means that there is no error in the CStep values and the correct operation is executing. Nevertheless, replay is intentionally started before CSteps receive incorrect values, thereby eliminating a circuit for correcting errors in the NCStep circuits.

Area C: If an error occurs, unnecessary replay will be started (false error detection). Since the replay itself is performed correctly, there is no real harm other than a delay penalty. If there is an error in the multiplexor, CStep0 and CStep2 receive incorrect value, and an error is caused in the datapaths and replay starts. Since CStep1 and CStep2 are not equal, the load signals to REGsR are disabled. Thus the replay input values are maintained and used in replay.

Area D: If an error occurs, CStep3 is erroneous, but replay does not start, and the value is not used. CStep3 will be overwritten by correct value before a next error occurs.

Area E: If the majority voter M is incorrect, CStepP and CStepS are different, and the error is detected through NCStep0 and NCStep1. Further, an error in any of M, SGP, or SGS results in erroneous control signals to the datapaths. Then, DatapathP and DatapathS perform different processing, resulting in a mismatch in the results, and an error is detected. In both cases, replay will start and correct the error.

area F: It is possible for REGsR to hold an incorrect replay input value due to an error in the comparator C or SGR, but the replay will not start and REGsR will be overwritten by the correct replay input data before a next error occurs.

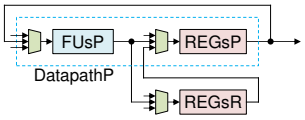

Fig. 8. The configuration of DatapathP and REGsR for registered input (A1).

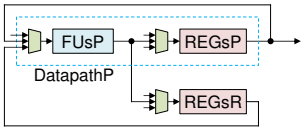

Fig. 9. The configuration of DatapathP and REGsR for direct input (A2).

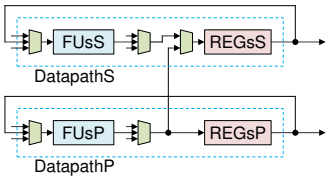

Fig. 10. Copying the replay results from DatapathP to DatapathS.

C. How to use replay input data

When executing replay, the data stored in REGsR is used as replay input data to perform the operations necessary for error correction. Replay is performed by the primary execution datapath DatapathP (functional unit FUsP and register REGsP). There are two possible ways to use replay input data.

The first method copies the replay input data from REGsR to the appropriate register in REGsP, and FUsP reads the data from REGsP and performs the replay operation. This is called the registered input method (Method A1). In this method, in order to transfer the data of REGsR to REGsP, the output of REGsR is connected to the input of REGsP through a multiplexer, as shown in Fig. 8. One CStep is required to transfer the replay input data before executing the replay operation.

The second method immediately inputs the replay input data read from REGsR to the appropriate FUsP and executes the replay operation. This is called the direct input method (Method A2). In this method, the output of REGsR is directly connected to the input of FUsP through a multiplexer, as shown in Fig. 9. Calculations for error correction can be executed from the first CStep of replay.

D. Replay control step assignment

The controller manages CStep not only for normal processing execution but also for replay execution. Normal execution of a certain process consists of a total of N CSteps, and each CStep of normal execution is expressed as 'C_k' ($0 \le k \le N-1$). When the integer k is expressed as a binary number, the number of bits of CStep in normal execution is $B = \lceil \log_2 N - 1 \rceil$. Note that $\lceil x \rceil$ is the smallest integer not smaller than x. 'R_k' represents the CStep executed during replay corresponding to 'C_k' in normal execution.

In DMR, multiple error detection times are set in consideration of the upper limit of the delay penalty, and different replays are executed depending on which time an error is detected. Now, there are P replays, and the number of CSteps executed by the ith replay is expressed as L_i ($i = 1, 2, \ldots, P$).

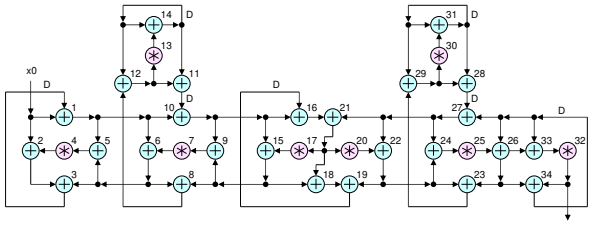

Fig. 11. The DFG of 5th order wave elliptic filter (WEF).

Consider the following two methods in assigning CSteps for replay. The first method is to sequentially assign CSteps after N to replay. This method is called a sequential assign method (B1). The replay input data shall be used according to A1 described in the previous section. Let $C_0, C_1, \ldots, C_{L_1-1}$ be the CSteps targeted for the first replay execution. In the sequential assign method, the replay input data is copied in CStep N, R_0 is executed in CStep $N+1$, R_1 in CStep $N+2$, and R_{L_1-1} in CStep $N+L_1$. The CSteps of normal execution targeted for the second replay are $C_{L_1}, C_{L_1+1}, \ldots, C_{L_1+L_2-1}$. Copying replay input data is executed in CStep $N+L_1+1$, R_{L_1} is executed in CStep $N+L_1+2$, R_{L_1+2} in CStep $N+L_1+3$, and $R_{L_1+L_2-1}$ in CStep $N+L_1+L_2+1$. The same shall apply hereinafter. In this way, the bit patterns of the CStep values of C_k and R_k are generally different. C_k and R_k perform the same processing in DatapathP, but since the bit patterns of the CStep values are different, the complexity of the control signal generation circuit SGP increases and the circuit area increases accordingly.

Method A2 described in the previous section does not require CStep to copy replay input data. The second method of assigning CStep to replay assumes the method A2 and assigns CStep $N_N + k$ to R_k. This method is called a copy assign method (B2). Note that N_N is a power of 2 that is not smaller than N, and $N_N = 2^B$. Similar to method B2, the least significant B bits of the CStep assigned to C_k and R_k have the same bit pattern. In the first CStep of replay, REGsR is selected as the FUsP input, and it is different from normal execution where REGsP is selected. Thus the control signals are not exactly the same for the normal execution and the replay, but for other CSteps, the control signals are identical. Therefore, it is expected that an increase in the area of the control signal generation circuit SGP can be suppressed.

In either method, in the last CStep of replay, the results executed in DatapathP are copied to REGsS in DatapathS in preparation for resuming normal execution. Hence wiring and a multiplexer are required to select the output of FUsP and input it to REGsS, as shown in Fig. 10.

IV. Experimental Results

We evaluate the area of DMR LSI using the proposed DMR controller. Targeted processing is 5th order wave elliptic filter (WEF) (26 additions, 8 multiplications), 8 input 8 output processing (8x8) (24 adds, 12 muls), 16 input 16 output processing (16x16) (64 adds, 32 muls) and the DFGs are shown in Figs. 11 to 13, respectively. In the DFGs, '+' represents an addition and

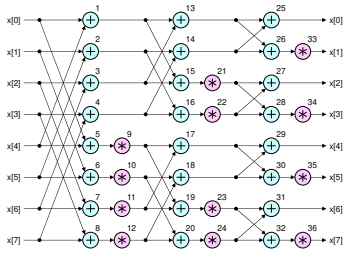

Fig. 12. The DFG of 8-input, 8-output processing (8x8).

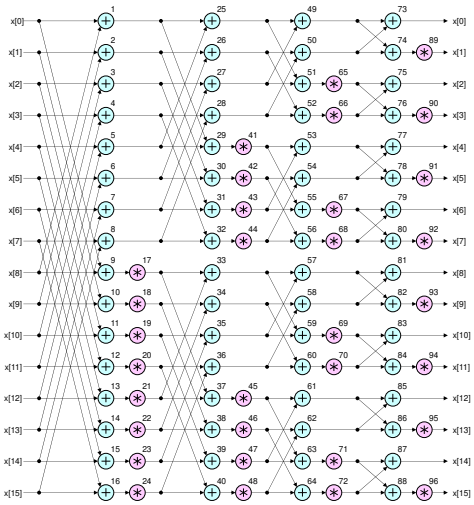

Fig. 13. The DFG of 16-input, 16-output processing (16x16).

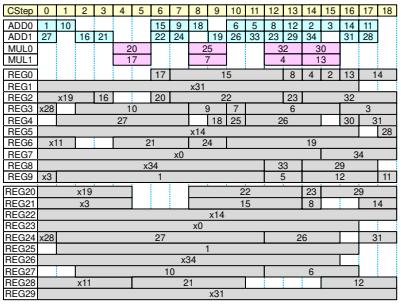

Fig. 14. The schedule and binding of WEF.

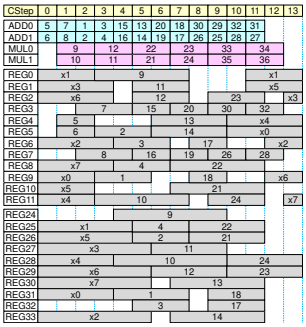

Fig. 15. The schedule and binding of 8x8.

'$*$' represents a multiplication. The schedule of operations, the binding between operations and FUs, and the binding between data and registers were assumed to be given as shown in Figs. 14 to 16. For example, 2 adders (ADD0, ADD1), 2 multipliers (MUL0, MUL1), and 10 registers (REG0 to REG9) were used for DatapathP in WEF. The schedule and binding for primary and secondary operations were identical. Hence additional 2 adders (ADD2, ADD3), 2 multipliers (MUL2, MUL3), and 10 registers (REG10 to REG19) were used for DatapathS in WEF although these are not shown in Fig. 14.

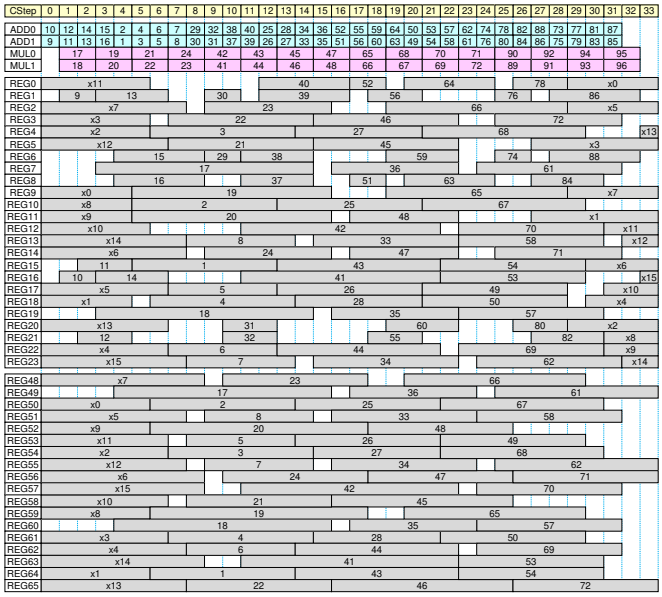

Fig. 16. The schedule and binding of 16x16.

TABLE I
RESOURCES (16 BITS)

	Area [μm^2]
Adder	258
Multiplier	3973
Register	297
Comparator	115
Majority voter	120
2-to-1 Multiplexor	92

The comparisons to detect soft-error in datapaths were inserted so that the delay penalty for each replay does not exceed 6 CCs. Then the necessary replay input data were derived and the binding between the replay input data and REGsR registers were given as shown in Figs. 14 to 16. For example, as REGsR, 10 registers (REG20 to REG29) were used to store replay input in WEF. Consequently, 4 adders, 4 multipliers, and 30 registers as well as necessary multiplexors are used in the datapath for WEF. Similarly, 4 adders, 4 multipliers, and 34 registers were used for 8x8, and 4 adders, 4 multipliers, and 66 registers for 16x16. Note that Figs. 14 to 16 show the binding of REGsR for replay input usage method A2. In the case of A1, the end time of each replay input is 1 CC shorter than A2 but the number of REGsRs is the same as A2.

The LSI areas of 16-bit resources are shown in Table I. These areas were determined by logic synthesis targeting a CMOS 90 nm process and the CC of 1 ns. An addition takes one CC. A multiplication takes two CCs and is not pipelined. The controller is described using HDL and its area is obtained by logic synthesis in the same way as the resources.

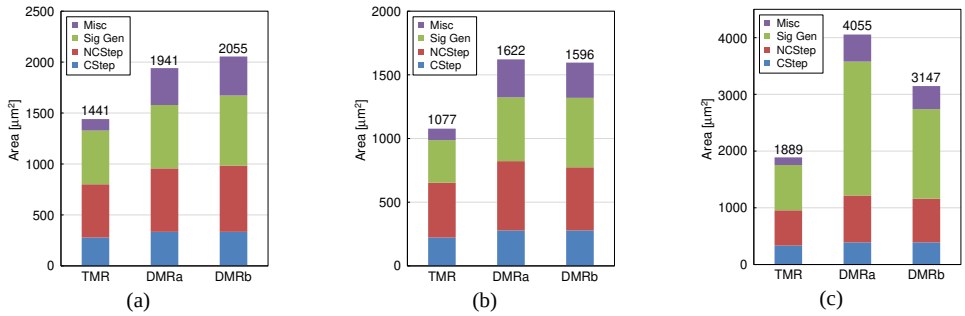

Fig. 17. The area of the redundant controllers. (a) WEF, (b) 8x8, (c) 16x16.

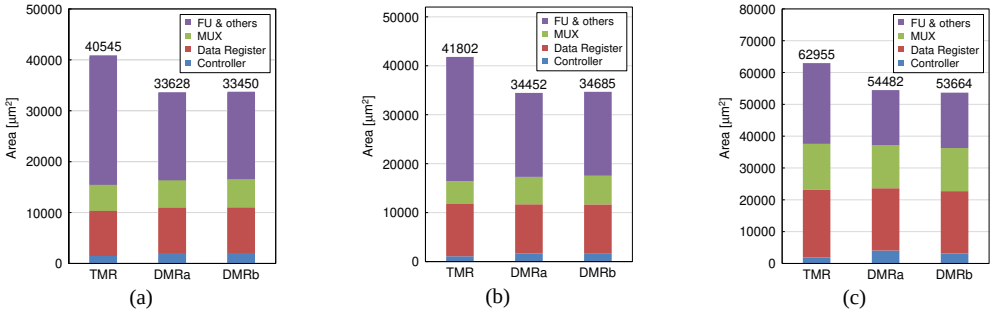

Fig. 18. The total area of the redundant processing systems. (a) WEF, (b) 8x8, (c) 16x16.

The experimented configurations of the replay input usage method and the replay control step assignment method are DMRa with A1 and B1 and DMRb with A2 and B2. The breakdown of the area of the controller is shown in Fig. 17. The comparison result of LSI area among the conventional full-TMR configuration (TMR) and the proposed DMR configuration is show in Fig. 18.

Regarding replay control step assignment, DMRb uses methods B2, where the least significant B bits of CStep assigned to CStep C_k in normal execution and CStep R_k in replay execution are identical. From Fig. 17, it can be seen that the area of the circuit signal generation circuit of DMRb tends to be smaller than that of DMRa.

DMR requires data registers REGsR for replay input data, and the number of registers is almost the same as REGsP and REGsS. Therefore, it can be seen from Fig. 18 that the register areas for DMR are almost the same as TMR, which have tripled data registers. The number of FUs in the datapath for DMR is reduced to two-thirds of TMR. Since the area of a multiplier is large compared to other resources, the reduction in the number of multipliers results in a smaller area of DMR than TMR as shown in Fig. 18. Comparing TMR and DMRb, the achieved area reduction is 17% for WEF, 17% for 8x8, and 14% for 16x16.

V. Conclusions

In this paper, an architecture of the DMR controller was proposed to implement the controller as well as the datapath in DMR. Total area of the LSI with the DMR configuration can be reduced up to 17% from full TMR configuration. Minimizing the number of registers for replay input data, optimizing the bindings of FUs and registers for minimizing the area of multiplexors, and the consideration of error check locations in the schedule for further optimization remain as future work.

Acknowledgements

This work was supported through the activities of VDEC, The University of Tokyo, in collaboration with NIHON SYNOPSYS G.K.

References

[1] R. Baumann, “Soft errors in advanced computer systems,” IEEE Design & Test of Computers, vol.22, no.3, pp.258–266, 2005.

[2] F. Wang and V.D. Agrawal, “Single event upset: An embedded tutorial,” Proc. Int. Conf. VLSI Design, pp.429–434, 2008.

[3] R.E. Lyons and W. Vanderkulk, “The use of triple-modular redundancy to improve computer reliability,” IBM Journal of Research and Development, vol.6, no.2, pp.200–209, 1962.

[4] S. Matsuzaka and K. Inoue, “A dependable processor architecture with data-path partitioning,” IPSJ Tech. Report, vol.2004-SLDM-117, pp.7–11, 2004.

[5] S. Mitra, M. Zhang, S. Waqas, N. Seifert, B. Gill, and K.S. Kim, “Combinational logic soft error correction,” Proc. IEEE Int. Test Conf., pp.824–832, 2006.

[6] Y. Suda and K. Ito, “A method of power supply voltage assignment and scheduling of operations to reduce energy consumption of error detectable computations,” Proc. The 17th Workshop on Synthesis And System Integration of Mixed Information Technologies, pp.420–424, 2012.

[7] J. Oh and M. Kaneko, “Area-efficient soft-error tolerant datapath synthesis based on speculative resource sharing,” IEICE Trans. Fund., vol.E99-A, no.7, pp.1311–1322, 2016.

[8] J. Oh and M. Kaneko, “Latency-aware selection of check variables for soft-error tolerant datapath synthesis,” IEICE Trans. Fund., vol.E100-A, no.7, pp.1506–1510, 2017.

[9] K. Ito, Y. Ishihara, and S. Nishizawa, “Minimization of vote operations for soft error detection in dmr design with error correction by operation re-execution,” IEICE Trans. Fund., vol.E101-A, no.12, pp.2271–2279, 2018.

[10] Y. Kitazawa and K. Ito, “Register minimization and its application in schedule exploration for area minimization for double modular redundancy lsi design,” IEICE Trans. Fund., vol.E105-A, no.3, pp.530–539, 2022.