DOI:10.1587/transfun.2024VLP0014

Publicized:2024/09/05

PAPER

Double Modular Redundancy Design of LSI Controller for Soft Error Tolerance

Katsutoshi OTSUKA[†], ***Nonmember and*** **Kazuhito ITO**[†a)], ***Member***

SUMMARY A soft error in LSI is a temporary malfunction in which signals in combinational circuits or data stored in registers are flipped. Double modular redundancy performs computation execution and data storing 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. In this paper, a double modular redundancy design for LSI controllers is proposed. The register for the control step and the combinational circuit to compute the control step value are doubled to check an error in the controller. Re-execution of operations necessary to correct an error in datapath and controller is controlled using one bit signal which is also doubled for error detection and correction. The area of the proposed controller is reduced to about half of that of the conventional triple modular redundant controller.
key words: *DMR, Controller, Soft error, LSI design*

1. Introduction

When neutrons caused by cosmic rays enter a large scale integrated circuit (LSI), a nuclear reaction with a neutron occurs, and carriers produced by the secondary ions are collected, the signal value in the circuit is reversed if the collected charge 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]–[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.

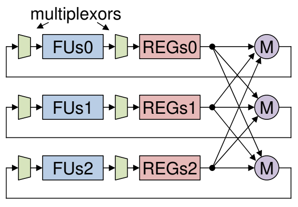


Fig. 1 Triple modular redundancy (TMR).

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]–[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.

2. Error collection and register usage in DMR

2.1 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 a soft error is limited to one module (combinational circuit block including majority voter and multiplexor, 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.

Final manuscript received January 1, 2003.
[†]The authors are with the Graduate School of Science and Engineering, Saitama University, Saitama-shi, 338-8570, Japan.
a) E-mail: kazuhito@mail.saitama-u.ac.jp

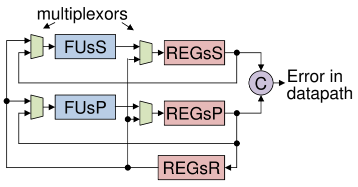


Fig. 2 Double modular redundancy (DMR).

2.2 TMR

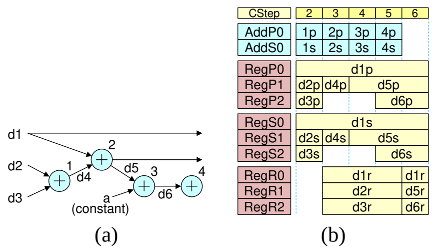
The datapath with triple modular redundancy [3] is illustrated in Fig. 1. The results of operations in functional units (FUs) are selected by multiplexers (MUXs) and stored in registers (REGs). Data is read from the REGs and selected by multiplexers (MUXs) to be used as input data for the FUs. FUs, REGs, 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.

2.3 DMR

Figure 2 shows the datapath for DMR. There are two FUs, one less than TMR, and they are called FUsP and FUsS, respectively. Each FU performs the same operation once, for a total of two operations. The two execution results are stored in registers REGsP and REGsS, respectively, and the values read from these registers are compared by a comparator (C). If they match, a correct result was obtained; if they do not match, an error has occurred. Then the error is corrected by re-executing the necessary operations [10]. Re-execution requires input data that is guaranteed to be error-free. Therefore, a third register, REGsR, is prepared to store the necessary data. When an error occurs in either REGsP or REGsS, the data in REGsR is error-free, and the data is read from REGsR and used as the input for re-execution.

2.4 Error Correction in DMR by Replay

Figure 3(a) shows an example data-flow graph (DFG), where a node represents an operation and an edge represents data dependency between operations. For example, operation 1 uses two data d2 and d3, and outputs the result d4. Then d4 as well as d1 are used by operation 2. With DMR, each operation is executed twice and these are called respectively the *primary* and *secondary* executions of the operation. They are denoted as ‘1p’ and ‘1s’ for operation 1. Figure 3(b) shows the schedule of operation executions and storing data. An FU AddP0 executes primary operations and another FU AddS0 executes secondary operations. RegP0 to RegP2 store the primary data and RegS0 to RegS2 store the secondary data. RegR0 to RegR2 store the input data for re-execution. In Fig. 3(b), 1p and 1s are executed



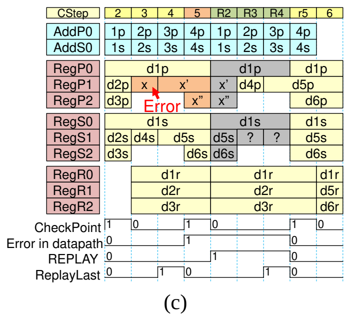


Fig. 3 An example for error correction by replay in DMR. (a) a DFG. (b) an operation and data schedule. (c) replay the operations when an error is detected.

in control step (CStep) 2, both 1p and 1s take one clock cycle (CC). The input data d2p and d3p (d2s and d3s) are read from RegP1 and RegP2 (RegS1 and RegS2), and used by 1p (1s). Then the results d4p and d4s are stored in RegP1 and RegS1, respectively, at the end of CStep 2.

A checkpoint [11] is a CStep where the correctness of data is checked and the data necessary as the input for re-execution are stored. Assume that CStep 2 is a checkpoint where d1p and d1s, d2p and d2s, and d3p and d3s are respectively compared, and if there is no error, d1p, d2p, and d3p are respectively stored to RegR0 to RegR2 as shown in Fig. 3(b). If an error occurred in RegP1 at CStep 3, the data d4p is incorrect and the succeeding data d5p and d6p becomes erroneous. Let CStep 5 be another checkpoint and the data d6p and d6s are compared by the comparator C in Fig. 2. Then the output of C becomes 1 and the error is detected. Since there exists an error at CStep 5, the results of operations executed at CStep 5 are erroneous and hence the result of operation 4 in this case is discarded. The re-execution starts at the CStep after CStep 5. Let the re-execution be called *replay*. In this case, CSteps 2, 3, and 4 are re-executed in order to correct the data d4p, d5p, and d6p. These CSteps are denoted as R2, R3, and R4 as shown in Fig. 3(c). The replay requires the replay input data d1r, d2r, and d3r. Hence the data have been stored in RegR0 to RegR2 at CStep 2. When the error is detected at CStep 5, the data in RegRs are maintained from CStep R2 to R4. Operation 1 executes using the input data d2r and d3r from RegR1 and RegR2, and the result d4p is stored in RegP1. Operation 2 executes using the input data d1r from RegR0 and d4p from RegP1, the result d5p is stored in RegP1. That is, replay is executed using either error-free data from RegRs or re-executed data from RegPs. At the last CStep of replay, R4, RegPs receive error-free data from RegRs if the data

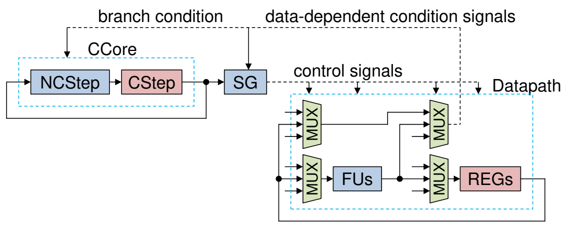


Fig. 4 Processing system consisting of controller and datapath.

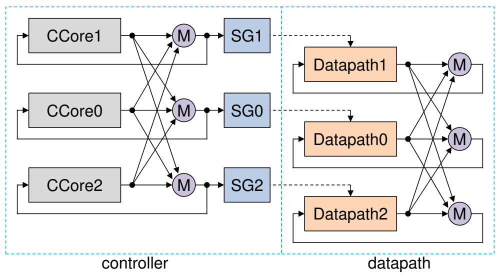


Fig. 5 Full TMR configuration of controller and datapath.

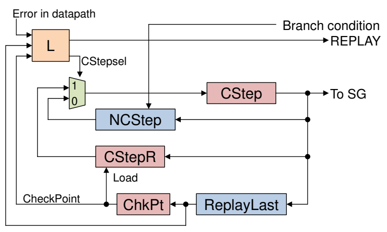


Fig. 6 The controller supporting the replay in DMR.

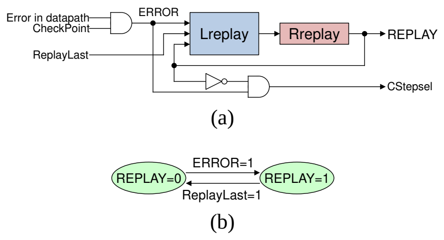


Fig. 7 The block L. (a) circuit. (b) state transition.

is not corrected by replay (RegP1 in this case). In addition, RegSs receive the correct data from either the error-free data from RegRs (d1s in this case), the corrected data from RegPs (d5s), or the output of FUs in replay (d6s). Then the normal operation resumes from CStep 5, which is indicated as r5, and the data in RegRs are updated as shown in Fig. 3(c).

The execution of operations are delayed when a replay is executed. The longest of the delay is called *delay penalty* and denoted as P_d. In real time applications, an upper tolerance limit is imposed on the delay penalty. The delay length of a replay is equal to the number of CSteps between two successive checkpoints. Therefore checkpoints must be set with intervals not longer than P_d to satisfy the tolerance.

3. 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 4 shows an outline of the (non-redundant) circuit. The controller consists of a register CStep and combinational circuits 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 and the condition signals obtained in the datapath if data dependent conditional branches are involved. 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. The SG may also input data-dependent condition signals to conditionally switch operations in a control step. The condition signals are omitted if necessarily in the remaining figures for simplicity. Let CCore denote the set of a CStep and a NCStep.

The datapath consists of functional units (FU), registers (REG), and multiplexers (MUX) that select and transfer data between FUs and REGs.

3.1 Full TMR

Figure 5 shows a full TMR for the controller and the datapath. The CCore, SG, and datapath 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.

3.2 Controlling replay in DMR

When an error is detected at a checkpoint, the error is corrected by replay. The replay can be done by rewinding the operation schedule to the previous checkpoint. For example in Fig. 3, CSteps 2 and 5 are the checkpoints. When an error is detected at a checkpoint, CStep 5, the operations scheduled from the previous checkpoint, CStep 2, to the current checkpoint, CStep 5, are executed in replay.

The normal CStep and its corresponding CStep in replay are assigned the identical binary code and they are distinguished by the signal REPLAY. The value of REPLAY becomes 0 in normal operation and 1 in replay, as shown in Fig. 3(c) for example. In the CStep in replay, the same operations as in the corresponding normal CStep are executed except that the error-free input data are obtained from REGsR. Therefore assigning the identical binary code to these CSteps is expected to minimize the complexity of the combinational circuits for control signal generations.

Figure 6 shows a controller to support controlling the replay. The register CStep holds the binary code of the current CStep. The combinational circuit NCStep calculates the next CStep value. The combinational circuit ReplayLast outputs 1 when the CStep is at one step before any of the

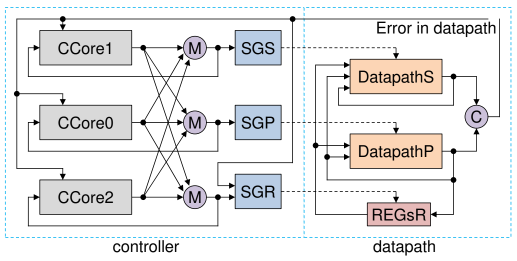


Fig. 8 The configuration of TMR controller and DMR datapath.

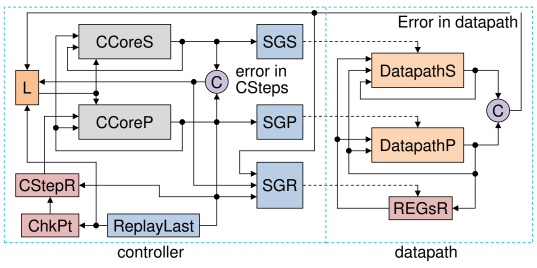


Fig. 9 DMR configuration of controller and datapath.

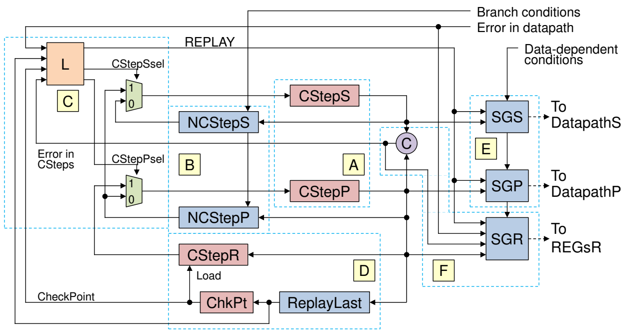


Fig. 10 The proposed architecture of DMR controller.

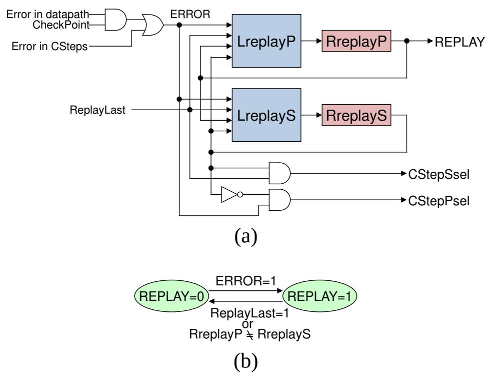


Fig. 11 The block L. (a) circuit. (b) state transition.

checkpoints. Therefore ReplayLast = 1 indicates the last CStep in replay. The register CStepR stores the binary code of the previous checkpoint. ReplayLast is stored in a 1-bit register ChkPt and the output becomes 1 when the current CStep is at one of the checkpoints. Therefore, the output of ChkPt is used as the load control signal of CStepR.

The block L in Fig. 6 calculates the signal REPLAY. The detailed circuit of L is shown in Fig. 7(a). The combinational circuit Lreplay and a 1-bit register Rreplay implements the state transition shown in Fig. 7(b). When an error is detected in datapath in normal operation (REPLAY = 0), the signal ERROR = 1, and CStepsel becomes 1 to set the checkpoint stored in CStepR to CStep to start a replay and REPLAY becomes 1. When the last CStep in replay is reached (REPLAY = 1 and ReplayLast = 1), REPLAY returns to 0.

3.3 TMR controller and DMR datapath

Figure 8 shows a configuration where the datapath is in DMR to reduce the area of the datapath. The duplicated datapaths are DatapathP for primary execution and DatapathS 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. Each of CCores in Fig. 8 corresponds to the controller shown in Fig. 6. The majorities of the CStep value and the REPLAY signal are respectively taken and the results are used by the combinational circuits SGP, SGS, and SGR to generate the control signals for DatapathP, DatapathS, and REGsR, respectively. Since SGR receives the signal indicating an error in the datapath so that, when an error occurred in the datapaths, REGsR do not overwrite the current data which are the error-free input data for replay. The complexity of the controller increases compared to the full TMR case, and the area increases accordingly. The effect of increased controller area is tripled in TMR controller.

3.4 Proposed DMR controller

Figure 9 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. 10.

The CStep register and the combinational circuit NCStep in Fig. 6 are doubled, and they are called CStepP and CStepS, and NCStepP and NCStepS, respectively. In normal operation, the value of CStepP and CStepS are identical and thus the outputs of NCStepP and NCStepS are also identical. The value of CStepP and CStepS, and the value of the signal REPLAY are given to the combinational circuits SGP and SGS to generate control signals for the datapaths, DatapathP and DatapathS, respectively. The value of CStepP is also used by the combinational circuit SGR to generate control signals for the replay input registers REGsR.

The detailed circuit of the block L in Fig. 10 is shown in Fig. 11(a). When an error is detected either in datapath at a checkpoint, CStepP, or CStepS in normal operation (REPLAY = 0), the ERROR becomes 1. The circuit Lreplay and the 1-bit register Rreplay are duplicated and called LreplayP and LreplayS, and RreplayP and RreplayS, respectively. When an error occurred in any of these components, the value of RreplayP and RreplayS differ. In this case, because of the error model in which if an error occurred in L, no other error occurred, the correct value of RreplayP and RreplayS is known to be 0. Therefore, LreplayP and LreplayS receive both RreplayP and RreplayS and when the values differ, RreplayP and RreplayS are corrected to 0. The third register to store error-free data for error correction is not necessary. The state transition of L is shown in Fig. 11(b). When the last CStep in any replay is reached (REPLAY = 1 and ReplayLast = 1), REPLAY returns to 0. CStepPsel becomes 1 to set CStepP to the checkpoint stored in CStepR when an error is detected (Error = 1) in normal operation (REPLAY = 0). CStepSsel becomes 1 to set CStepS to the correct value obtained from NCStepP to resume normal operation at the end of the replay (REPLAY = 1 and ReplayLast = 1).

When an error is detected either in datapath, CStepP, or CStepS all the load signals to REGsR become 0 by SGR so that the replay input data are not overwritten.

3.5 Responding to controller errors

The response to soft errors in each area of the controller will be explained according to Fig. 10. 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 CStepP or CStepS is detected by the comparator C and ERROR signal in Fig. 11(a) becomes 1 to start a replay. SGR sets the load signals of REGsR to 0 to keep the current replay input data in REGsR.

Area B: If there is an error in NCStepP or NCStepS, the next values of CStepP and CStepS do not match. Then the replay will start as in the case of area A.

Area C: If the ERROR signal in Fig. 11(a) becomes 1 by an error, that is a false detection of error, replay starts unnecessarily. Since all the data are correct and the replay itself is performed correctly, there is no real harm other than a delay penalty. If REPLAY becomes 1 by an error, replay operation is executed in datapathP and the result does not match with the result of datapathS. Then it is detected by the comparator in the datapath and a valid replay will start. If CStepPsel becomes 1 by an error, the value of CStepP becomes not the same as CStepS in the next CStep, and the error is detected and a replay will start. CStepSsel becoming 1 by an error does not matter since the outputs of NCStepP and NCStepS are identical in this case.

Area D: ReplayLast would have the incorrect value 1 by an error in normal operation (REPLAY = 0) because of the error model. The value 1 of ReplayLast is ignored in normal operation by the block L. An error in either ChkPt or CStepR makes the value of CStepR erroneous, but a replay does not start, and the value is not used. CStepR will be overwritten by correct value before a next error occurs.

Area E: An error in either 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: If the comparator C outputs 1 by an error, SGR sets

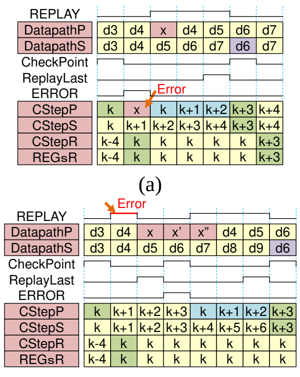


Fig. 12 Error and replay examples. (a) error in CStepP. (b) error in REPLAY signal.

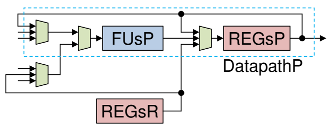


Fig. 13 The configuration of DatapathP and REGsR.

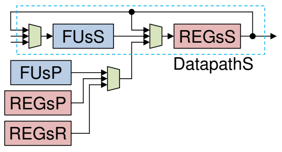


Fig. 14 Copying the replay results from DatapathP to DatapathS.

the load signals of REGsR to 0 to keep the current replay input data in REGsR. A replay starts by the error of the comparator C becoming 1, the replay itself is done using the correct replay input data.

An error in SGR causes REGsR to hold incorrect replay input values, but the replay will not start and REGsR will be overwritten by correct replay input data at the next checkpoint before a next error occurs.

Figure 12 shows examples of errors and the corresponding replay. In this example, the checkpoints are CStep k and k + 3. In Fig. 12(a), an error occurred in CStepP at CStep k + 1. The comparator in the controller detects the error and the signal ERROR becomes 1. In the next CStep, the replay starts using the replay input data stored in REGsR. At CStep k + 2 in the replay, ReplayLast = 1 and the correct CStep value is set to CStepS, REGsS in datapathS receive correct data from either REGsR, REGsP, or FUsP, and the normal operation resumes from CStep k + 3.

In Fig. 12(b), REPLAY becomes 1 by an error in LreplayP or RreplayP at CStep k + 1. REPLAY returns to the correct value 0 at CStep k + 2 but erroneous operations are executed in datapaths and the results mismatch at CStep k + 2. The signal ERROR becomes 1 at the checkpoint CStep k + 3. The replay starts and and the normal operation resumes from CStep k + 3.

3.6 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).

Operations in replay read necessary replay input data directly from REGsR. Therefore MUXs are connected between FUsP and REGsR as shown in Fig. 13. As the replay proceeds, operations may read the data corrected by replay and stored in the primary data registers REGsP.

At the end of replay, to prepare to resume the normal operation, REGsS must receive correct data. The correct data are obtained from either REGsR storing error-free data, REGsP storing the data corrected by the operations in replay, or FUsP executing the operation at the last CStep in replay. Therefore MUXs are connected between REGsS, REGsR, REGsP, and FUsP as shown in Fig. 14.

Each register of REGsR receives data from the corresponding register of REGsP and no MUX is necessary from the output of REGsP to the input of REGsR.

4. Experimental Results

We evaluate the area of DMR LSI using the proposed DMR controller. Targeted processing is 5th order wave elliptic filter (WEF) [12] (26 additions, 8 multiplications), 8 input 8 output processing (DFG1) (30 adds, 14 muls), 16 input 16 output processing (16x16) (64 adds, 32 muls) and the DFGs are shown in Figs. 15 to 17, respectively. In the DFGs, '+' represents an addition and '$*$' 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. 18 to 20. 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. 18. Binding was optimized by simulated annealing [13] to minimize the area of MUXs.

The checkpoints were determined so that the delay penalty P_d is 4 CCs for all the cases. The checkpoints are shown with small rectangles in Figs. 18 to 20. 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. 18 to 20. 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 48 registers were used for DFG1, and 4 adders, 4 multipliers, and 72 registers for 16x16.

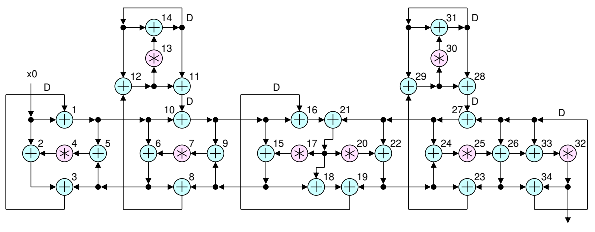


Fig. 15 The DFG of 5th order wave elliptic filter (WEF).

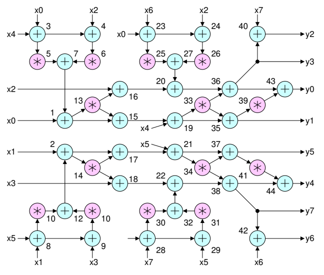


Fig. 16 The DFG of 8-input, 8-output processing (DFG1).

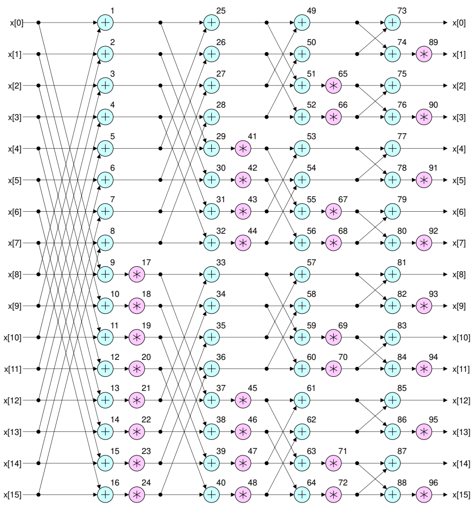


Fig. 17 The DFG of 16-input, 16-output processing (16x16).

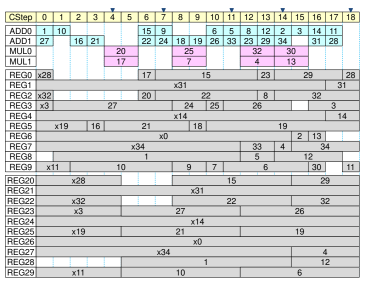


Fig. 18 The schedule and binding of WEF.

The LSI areas of 16-bit resources are shown in Table 1. 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

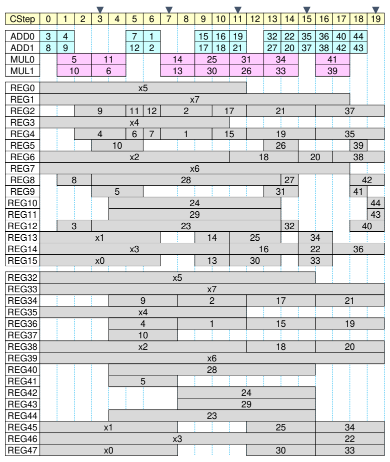


Fig. 19 The schedule and binding of DFG1.

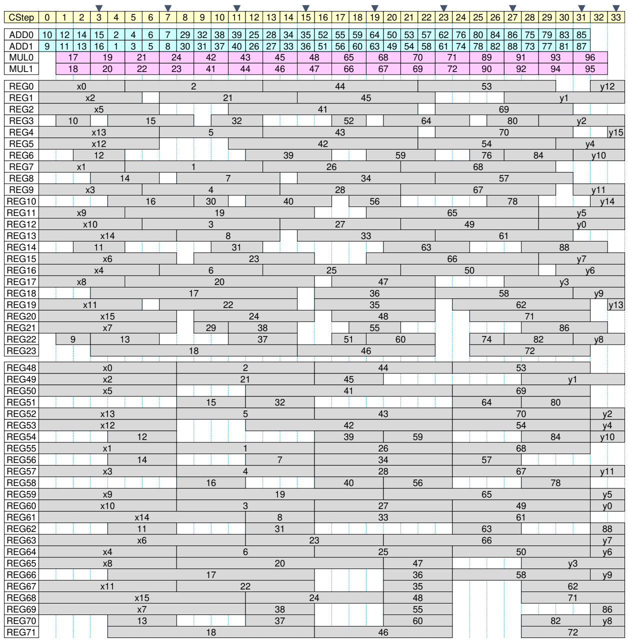


Fig. 20 The schedule and binding of 16x16.

area is obtained by logic synthesis in the same way as the resources.

It is possible that the replay is done both in the primary and the secondary parts. Let this be called simple DMR. The area of the SGs and MUXs are compared in Table 2 where MUXP is the area of MUXs in datapathP, and MUXS in datapathS. In simple DMR, SGs and MUXs in datapaths should be identical respectively between the primary and the secondary parts, and the area of SGs and MUXs are twice the SGP and MUXP plus SGR. While the area of SGs and MUXs of simple DMR is smaller than that of the proposed DMR where the area of SGs and MUXs are the sum of SGP, MUXP, SGS, MUXS, and SGR in the case of 16x16, it is reversed in the case of WEF and DFG1. Which DMR method achieves the smaller would depend on the given DFG, schedule, the number of resources, and P_d.

Table 1 Resources (16 bits)

	Area [μm^2]
Adder	258
Multiplier	3973
Register	297
Comparator	115
Majority voter	120
2-to-1 Multiplexor	92
3-to-1 Multiplexor	135
4-to-1 Multiplexor	181
5-to-1 Multiplexor	240
6-to-1 Multiplexor	291
7-to-1 Multiplexor	339

Table 2 The area of the signal generators and multiplexors in datapath

	SGP	SGS	SGR	MUXP	MUXS
WEF	347	240	50	4216	3931
DFG1	233	193	31	4294	3900
16x16	600	530	140	6358	7182

The unit is μm^2.

In the remainder, the results for the proposed MDR rather than the simple DMR are presented.

The area of the controllers are shown in Fig. 21. ‘Full TMR’, ‘TMR Controller’, and ‘DMR Controller’ are the TMR controller for full TMR in Fig. 5, the TMR controller for DMR datapath in Fig. 8, and the proposed DMR controller in Fig. 10, respectively. The area of the proposed controller is about half the area of the TMR controller for DMR datapath, and slightly larger than the TMR controller for full TMR. The number of registers for CStep (CStep, CStepP, CStepS, and CStepR) were reduced from 6 in the TMR controller to 2 in the proposed DMR controller, the MUX and NCStep from 3 to 2, and the circuit ReplayLast and 1-bit register ChkPt from 3 to 1. Although a CStep comparator was added in the proposed DMR, the majority voters were removed. The combination of these factors was the reason of the reduction of the area of the proposed DMR controller from the TMR controller.

The comparison result of LSI area among the conventional full-TMR configuration and the TMR controller for DMR datapath and the proposed DMR configuration is show in Fig. 22. 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. 22 that the register areas for DMR datapath are almost the same as TMR datapath, which have tripled data registers. The number of FUs in the DMR datapath 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. 22. Comparing full TMR and DMR with proposed controller, the achieved area reduction is 19% for WEF, 16% for DFG1, and 11% for 16x16.

The area of the controllers without SGs were estimated when the schedule length is large. Figure 23 shows the area of the conventional TMR controller and the proposed DMR controller for the DMR datapath when the schedule length is

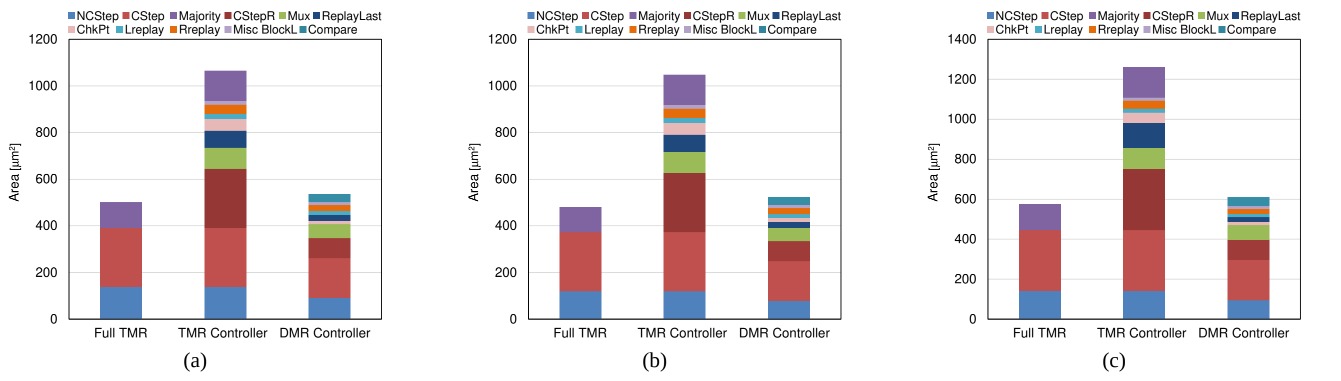


Fig. 21 The area of the redundant controllers. (a) WEF, (b) DFG1, (c) 16x16.

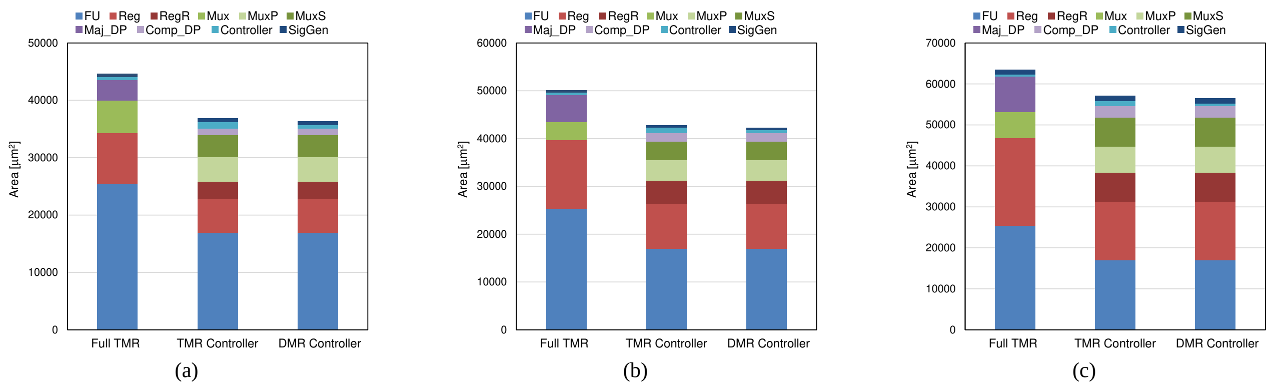


Fig. 22 The total area of the redundant processing systems. (a) WEF, (b) DFG1, (c) 16x16.

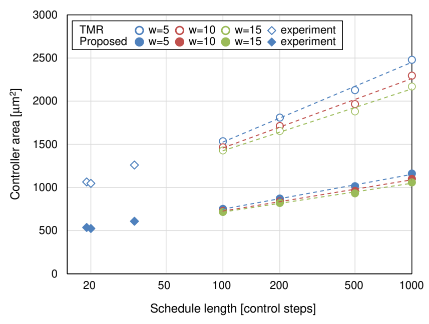


Fig. 23 The area of the controllers.

100, 200, 500, and 1000 steps, and checkpoints are set every $W = 5$, 10, and 15 CSteps to achieve $P_d = W$. The number of bits of the registers CStep, CStepP, CStepS, and CStepR, NCStep circuits and MUX were derived by the schedule length, and ReplayLast circuit was designed according to the checkpoints determined by W. In all the cases, it can be seen that the area of the proposed controller is approximately half that of the conventional controller. The area of the controllers for WEF, DFG1, and 16x16 are also shown in Fig. 23 and it is confirmed that the estimation is appropriate.

5. Conclusions

In this paper, an architecture of a DMR controller was proposed to implement the controller as well as the datapath in DMR. The area of the proposed DMR controller is about half the area of the TMR controller for DMR datapath. Total area of the LSI with the DMR configuration can be reduced up to 19% from full TMR configuration.

The proposed controller assumes that the operation and data storing schedules of the primary and secondary are identical. Consideration for the case that the schedules for the primary and secondary operations are not identical for any reason such as minimizing the number of FUs remains as future work.

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.
[11] A. Orailoğlu and R. Karri, "Coactive scheduling and checkpoint determination during high-level synthesis of self-recovering microarchitectures," IEEE Trans. VLSI Syst., vol.2, no.3, pp.304–311, 1994.
[12] S.M. Heemstra de Groot, S.H. Gerez, and O.E. Herrmann, "Range-chart-guided iterative data-flow graph scheduling," IEEE Trans. Circuits Syst.-I: Fund. Theory & Appl., vol.39, pp.351–364, 1992.
[13] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," Science, vol.220, no.4598, pp.671–680, 1983.

Katsutoshi Otsuka received the B.E. and M.E. degrees in Electrical and Electronic Engineering from Saitama University, Japan, in 2022 and 2024, respectively. Currently he is with KYOCERA Document Solutions Inc. His research interests include reliable LSI design.

Kazuhito Ito received the B.E., M.E., and Ph.D degrees in Electrical Engineering from Tokyo Institute of Technology, Japan, in 1987, 1989, and 1992, respectively. He is a professor of the Graduate School of Science and Engineering, Saitama University, Saitama, Japan. His research interests include high-level synthesis in LSI design, VLSI signal processing, and design automation of system LSIs. He is a member of IEICE, IPSJ, and IEEE.