Register Minimization in Double Modular Redundancy Design with Soft Error Correction by Replay

Yuya Kitazawa† Shinichi Nishizawa‡ Kazuhito Ito†

†Graduate School of Science and Engineering
Saitama University
Saitama 338-8570, Japan
{kitazawa,kazuhito}@elc.ees.saitama-u.ac.jp

‡Faculty of Engineering
Fukuoka University
Fukuoka 814-0180, Japan
nishizawa@fukuoka-u.ac.jp

Abstract— Double modular redundancy (DMR) is to execute an operation twice and detect soft error by comparing the duplicated operation results. The soft error is corrected by executing necessary operations again, called replay. The replay requires error-free input data and registers are needed to store such necessary error-free data. In this paper, a method to minimize the required number of registers is proposed where replay intervals are appropriately selected so as not to increase the register requirement. The experimental results show up to 27% reduction of required registers.

I. Introduction

When neutrons derived from cosmic rays or the like enter a large scale integrated circuit (LSI), a transient error such as inversion of a signal value in the circuit occurs if the energy exceeds a threshold. This is called a soft error [1, 2]. The threshold decreases as the signal energy in the circuit decreases due to the supply voltage lowering and miniaturization of semiconductor devices. The probability of occurrence of soft errors is increasing accordingly.

A redundancy method has been considered as a means for solving soft errors. In triple modular redundancy (TMR), the same operations are performed in triplicate and the majority is taken, thereby obtaining the correct result even if an error occurs in any one. TMR however has the disadvantage that the circuit size and power consumption are tripled.

Double modular redundancy (DMR) performs the same process in duplicate and detects a soft error by comparing the results. An error is corrected by re-execution [3, 4, 5]. As a re-execution method, a schedule dedicated to correcting the error is prepared, and when an error is detected, the dedicated schedule is invoked [6, 7]. In a rewind-style error correction scheme, a schedule dedicated to error correction is not used, but the necessary part of the original schedule is invoked [8]. While the delay of re-execution can be reduced by using a dedicated schedule for error correction, the execution control of the re-execution schedule is required in addition to the execution control of the original schedule, and therefore the increased control circuit area would be a disadvantage. The rewind-style re-execution simply executes the necessary part of the normal schedule to correct the error. Therefore, the change to the control circuit can be minimum and the increase in the control circuit area would be very small. In this study, we adopt rewind-style re-execution.

Re-execution of operations for error correction requires the input data that is guaranteed to be error-free. Thus in addition to potentially erroneous data, the error-free data for error correction must be prepared and many registers might be needed to store them. It is an important problem to minimize the registers for storing these error-free data for error correction. Methods of minimizing register cost in DMR are proposed in [9, 10]. In [9], errors that occur in registers are not taken into account. Although [10] aims to minimize the register area cost required on the premise that the input data for re-execution is stored in the radiation hardened register [4, 11], there is a problem that the hardened register has a larger area than the normal register. In this research, no radiation hardened register is used, and the number of registers is reduced which hold the input data for error correction as well as the ordinary data for normal operations.

The data that needs to be stored for re-execution depends on how the set of operations to be re-executed is selected. Hence the required number of registers can be reduced by appropriately selecting the operations to be re-executed. In this paper, we propose a method in DMR design that minimizes the required number of registers for a given DMR operation schedule by optimizing the combination of the re-execution operations.

The remainder of the paper is organized as follows. The error detection and the correction by re-executing operations are described in Sect. 2. The proposed method to minimize registers is presented in Sect. 3. Experimental results are presented in Sect. 4 and Sect. 5 concludes the work.

II. Error collection and register usage in DMR

A. Error model and DMR

To detect soft error in DMR, an operation execution and the input data to the operation are duplicated, and the equality of the duplicated operation results is checked. Error is modeled so that no more than one error occurs at the same time in 1) the input data, 2) duplicated operation execution, and 3) the equality check (comparison) for error detection, all of which

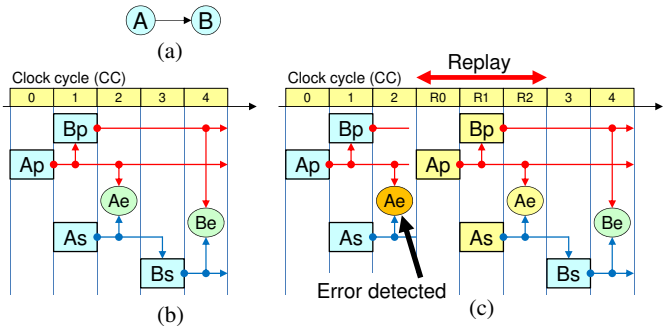

Fig. 1. An example for error correction by replay in DMR. (a) a DFG. (b) DMR schedule for operation execution. (c) replaying the schedule when an error is detected.

are related to an operation. When an error is detected for an operation, the error exists in one of the duplicated input data, in the operation executions, or in the equality check. If an error is detected, the operation is executed again. The re-execution is done within a sufficiently short time interval from the error and thus no error occurs during the re-execution.

B. Error correction by replay

When an error is detected in DMR, the error can be mitigated by executing necessary operations. Figure 1 shows an example of the error mitigation. Assume there are two operations 'A' and 'B' and there exists data dependency that 'B' uses the result of 'A' as shown in Fig. 1(a). Let 'B' be said a *child* of 'A'. 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. When both Ap and As are executed and the results are stored in registers, they are compared to check an error. Figure 1(b) is a time chart showing the schedule of operation executions and comparisons, where 'Ae' denotes a comparison. Let d(Gm) denote the result produced by Gm for operation G (m = p or m = s). For example, Ap and As start at clock cycle (CC) 0 and 1, respectively. Both Ap and As take one CC and d(Ap) and d(As) are stored in registers at the end of CC 0 and CC 1, respectively. An arrow in Fig. 1(b) represents the lifetime of data stored in a register. The comparison between d(Ap) and d(As) is done at CC 2. If the comparison is affirmative, i.e., the compared data are equal, any of Ap, As, and d(Ap) and d(As) stored in registers does not contain an error. If the comparison is negative, either Ap or As, or one of the data stored in registers, or even the comparison itself contains an error. Since it is not possible to identify which is erroneous and which are correct, both Ap and As are executed again, and d(Ap) and d(As) are stored in registers again. According to the error model where at most one error occurs within a short time period, re-executed Ap and As, and d(Ap) and d(As) stored in registers are error-free, and thus the error is mitigated.

When an error occurred with operation A, the child (children) of operation A which has executed before re-execution of operation A might have used erroneous d(Ap) and d(As). Thus the child (children) must be re-executed when operation

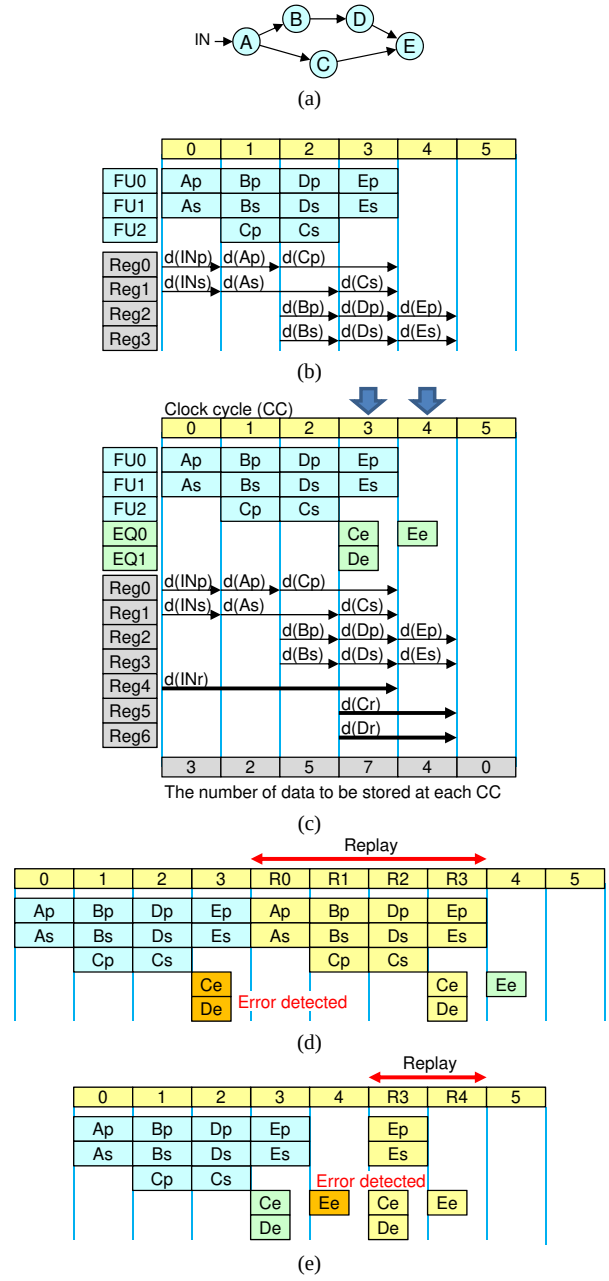

Fig. 2. Error-free input data for error correction by replay. (a) a DFG. (b) DMR schedule for operations and data. (c) error detection schedule and the necessary input for replay. (d) replay invoked when an error is detected at CC 3. (e) replay invoked when an error is detected at CC 4.

A is re-executed. Such re-execution of a child (children) can be easily achieved by repeating the schedule from the start of Ap. Let this scheme be called *replay*. Figure 1(c) shows an example of replay when an error occurred for operation A. Since Ap is scheduled at CC 0 and the error occurred for operation A is detected at CC 2, the part of the schedule from CC 0 to CC 2 is executed again in the replay. The replayed CC is denoted as 'Rt' for the original CC t. By re-executing Ap, the error-free d(Ap) is stored in a register, and Bp is also re-executed using the error-free d(Ap). After the replay, the original schedule resumes.

C. *Error-free input data for replay*

Suppose that the DMR operation schedule of the DFG in Fig. 2(a) is given as shown in Fig. 2(b). It is assumed that all the operations take 1 CC and 3 functional units (FUs) are used. In Fig. 2(a), 'IN' indicates the input data to the operation A. The data to Ap is denoted as d(INp) and the data to As as d(INs). d(INp), d(INs), and other data are bound to 4 registers as shown in Fig. 2(b). Let p_D denote the allowed maximum delay penalty caused by the replay. Here p_D is specified as 4 CCs. To satisfy p_D, comparisons Ce and De are executed at CC 3 using two equality check units EQ0 and EQ1, and a comparison Ee at CC 4 using EQ0 as shown in Fig. 2(c). If Ce or De detect an error, operations A, B, C, and D need be re-executed. Operations Ep and Es have been executed, and they might have used erroneous data. Therefore Ep and Es also must be re-executed. The replay of 4 CCs from R0 to R3 is performed as shown in Fig. 2(d). If Ee detects an error, the replay of 2 CCs of R3 and R4 is performed as shown in Fig. 2(e).

Re-execution of Ap and As requires the error-free input data 'IN'. Thus another copy of 'IN' denoted as d(INr) is kept in a register not affected by the error in d(INp) or d(INs). 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 or De, d(INr) is ensured to be error-free. Let such data for replay be called *replay input*. When Ce or De detect an error at CC 3, d(INr) is copied to registers Reg0 and Reg1 at the end of CC 3, and the re-executed Ap and As respectively use the data at the next CC of R0. This suggests that d(INr) must be kept in a register by CC 3. When Ee detects an error at CC 4, Ep and Es are re-executed and they require d(Cp), d(Cs), d(Dp), and d(Ds) in Reg0, Reg1, Reg2, and Reg3, respectively. Therefore the error-free replay inputs d(Cr) and d(Dr) are copied respectively to Reg0 and Reg1, and to Reg2 and Reg3 at CC 4 when the error is detected. This suggest that d(Cr) and d(Dr) must be stored in registers at the end of CC 2 and kept by CC 4. A possible register binding with the minimized number of registers is shown in Fig. 2(c) and 7 registers are required.

III. Register Minimization

Here we present a method to choose a set of error detection points which minimizes the required number of registers for a given operation schedule.

A. *Register Requirement*

Figure 3(a) shows an example DFG. All the operations A through G are assumed to take 1 CC. Operation A takes an external input 'IN'. Let a DMR operation schedule with two FUs be given as shown in Fig. 3(b). The figure also shows the lifetime of data and their binding to registers. Without considering the replay input, the number of data to be stored in each CC is shown in the bottom of Fig. 3(b). The values are determined according to the given schedule and each value indicates

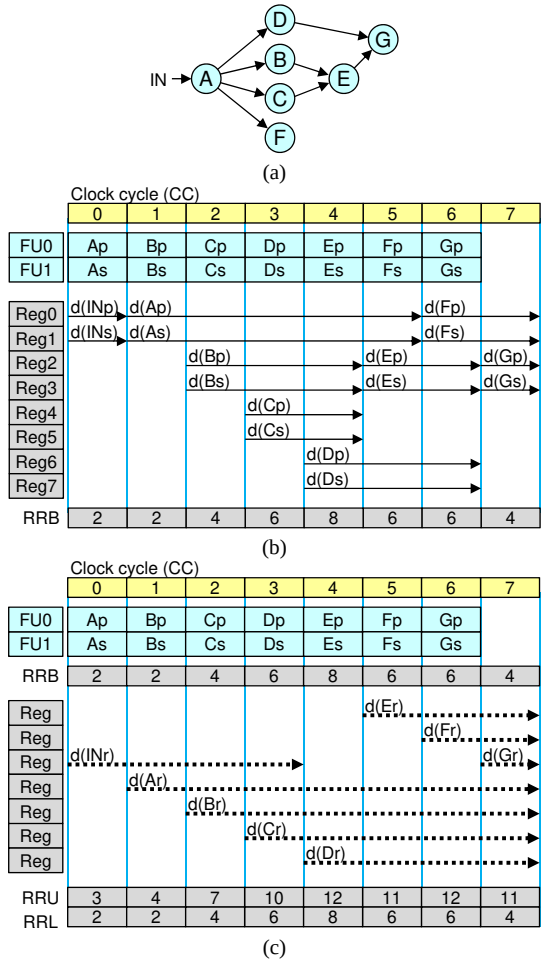

Fig. 3. Register requirement for DMR. (a) a DFG. (b) baseline register requirement for the primary and secondary operations. (c) the upper and lower bounds of register requirement when no replay interval is considered.

the required minimum number of registers in the corresponding CC. Let the values be called *RRB, the baseline of register requirement*.

By taking replay input into account, considering every possible replay execution, the replay inputs are tentatively required as shown in Fig. 3(c). Since which replay is executed and which replay inputs are needed are not determined at this moment, the lifetime of replay inputs are indicated in dotted bold arrows. The minimum number of registers known to be mandatory is denoted as *RRL, the lower bound of register requirement*. On the other hand, the maximum number of registers possibly required at most is denoted as *RRU, the upper bound of register requirement*. When any replay has not been determined, RRL is equal to RRB, and RRU is the sum of RRB and the number of replay inputs which tentatively have to be stored in each CC. The RRL and RRU values are shown in the bottom of Fig. 3(c).

For the given operation schedule, assume that the equality checks of the primary and secondary operations of D, E, and F are executed at CC 6 and the schedule from CC 3 to CC 6 is replayed when a soft error is detected as shown in Fig. 4(a). Since the replay of D, E, and F requires the replay inputs of

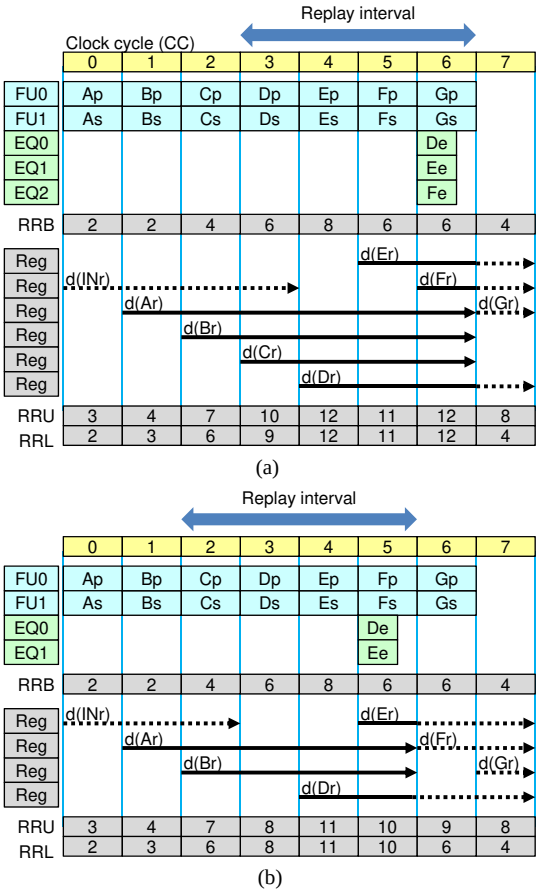

Fig. 4. The upper and lower bounds of register requirement. (a) for $Replay(3,6)$. (b) for $Replay(2,5)$.

d(Ar), d(Br), and d(Cr), these data must be stored in registers until CC 6, at which the equality checks are scheduled. Thus the lifetimes of d(Ar), d(Br), and d(Cr) are determined and these are shown in solid bold arrows in Fig. 4(a). As the replay inputs for G, d(Cr) and d(Dr) are required to be stored in registers from CC 3 and CC 5, respectively, to at least CC 6. In addition, d(Fr) is required to be stored in a register at least at CC 6 since d(Fr) would be needed as a replay input for operation(s) succeeding this DFG. Here the values of RRL are updated as the sum of RRB and the number of determined replay inputs. The values of RRU are also updated, and these updated values are shown in the bottom of Fig. 4(a). The RRL and RRU are identical at CC 4 (and CC 6 too), and it suggests that at least 12 registers are necessary.

Assume another case where the equality checks are executed for operations D and E at CC 5 and the schedule from CC 2 to CC 5 is replayed when a soft error is detected as shown in Fig. 4(b). The operations C, D, and E are replayed and the replay inputs d(Ar) and d(Br) are required to be stored until CC 5. Thus the lifetimes of d(Ar) and d(Br) are determined as shown in solid bold arrows in Fig. 4(b). As the replay inputs for G, d(Dr) and d(Er) are required to be stored in registers at least until CC 5. The updated values of RRL and RRU are shown in the bottom of Fig. 4(b). The RRL and RRU are identical at CC 4, and it suggests that at least 11 registers are necessary.

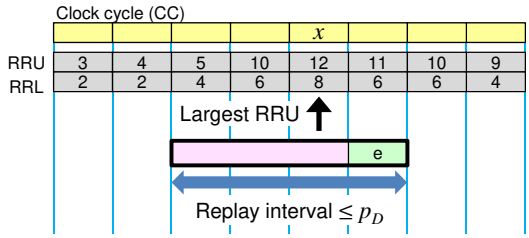

Fig. 5. Replay interval including the CC with the largest RRU.

This example shows that the required number of registers can be minimized by selecting optimized combination of the equality check points for error detection.

B. *Register Minimization Algorithm*

When an equality check Qe for some operation Q detects an error, the related replay re-executes the operations which satisfy the conditions: 1) there is a path from the operation to Q and 2) the error-freeness of the operation is not checked by any equality checks other than Qe. The interval of CCs between the CC at which the earliest re-executed operation is scheduled and the CC at which Qe is scheduled is called *a replay interval*. To satisfy the constraint of p_D, every replay interval must be no longer than p_D. Let $Replay(f,t)$ denote the replay interval which is invoked when an error is detected by one of the equality checks scheduled at CC t and starts at CC f. For example, Fig. 2(d) shows a replay interval $Replay(0,3)$ for the operation schedule of Fig. 2(b). In this case, $Replay(0,3)$ consists of CCs from CC 0 to CC 3 and the length is within $p_D = 4$. If the equality check(s) at CC t confirms no error, the error-freeness is ensured for the operations scheduled within $Replay(f,t)$ except the operations scheduled at CC t. For example, as shown in Fig. 2(c), Ep, Es, Ce, and De are scheduled at CC 3. Although Ep and Es are included in $Replay(0,3)$, the error-freeness of E is not ensured because Ce and De are not data dependent on Ep and Es. It is said that the operations scheduled in a replay interval $Replay(f,t)$ except for CC t are *covered* by $Replay(f,t)$. In this example, operations A, B, C, and D are covered by $Replay(0,3)$. To detect any soft error in the system, every operation must be covered by a replay interval. Thus in this example, another replay interval $Replay(3,4)$, which is invoked if an error is detected by Ee and whose length is 2 CCs, is necessary to cover operation E. It is important to note that the first CC of a replay interval overlaps with the last CC of another replay interval.

When a replay interval is adopted, while the numbers of registers required at CCs in the replay interval are determined, the register requirement at other CCs would be increased and it might result in a design with excessively many registers. Hence the register minimization strategy taken in this work is as follows: the larger the register requirement for the CC is, the earlier the required number of registers at the CC is fixed as small as possible by appropriately selecting a replay interval which includes the CC. This is illustrated in Fig 5. The RRU is largest at CC x among all the CCs and 12 registers would

Input: DFG $= (N, E)$ describing the processing algorithm,
the upper limit of delay penalty p_D,
a DMR schedule with T clock cycles (CCs)
Output: replay intervals to minimize registers
1: Compute RRB for all the CCs;
2: $L \leftarrow$ all the CCs from 0 to $T-1$;
3: **while** L is not empty **do**
4: Evaluate RRU and RRL for all the CCs;
5: $C \leftarrow$ CCs in L with the largest RRU value;
6: $W \leftarrow \emptyset$;
7: **foreach** $x \in C$ **do**
8: **foreach** $w = Replay(f,t)$ satisfying (1) $f \leq x \leq t-1$, (2) $t-f+1 \leq p_D$, and (3) all the CCs from f to $t-1$ are included in L **do**
9: Evaluate RRU and RRL for all the CCs assuming w is adopted;
10: Put w with RRU and RRL values into W;
11: **end foreach;**
12: **end foreach;**
13: Select the replay interval $w = Replay(f,t) \in W$ according to the following criteria applied in order
(1) the largest RRL is smallest,
(2) RRL at CC f is smallest,
(3) the interval $t-f+1$ is largest,
(4) f is earliest;
14: Remove all the CCs k $(f \leq k \leq t-1)$ from L;
15: **end while;**

Fig. 6. Pseudocode of the proposed algorithm for register minimization.

be necessary at CC x in the worst case (RRU $= 12$). Thus a replay interval $Replay(f,t)$ which includes CC x in its interval $(f \leq x \leq t-1)$ is selected to fix the register requirement at CC x. Many replay intervals $Replay(f,t)$ including CC x may exist. For each $Replay(f,t)$ which satisfies both $f \leq x \leq t-1$ and $t-f+1 \leq p_D$, the RRU and RRL are evaluated assuming the replay interval is adopted. Then the one replay interval is selected where the largest of RRL values for all the CCs is the smallest. If there is a tie, it is broken by the following priorities: 1) RRL at CC f is the smallest, and 2) the interval $t-f+1$ is the largest.

More than one CC with the largest RRU may exist. In that case, as the third priority to break a tie, the replay interval with the earliest f is selected. If a tie still exists, select one arbitrarily.

Greedily selecting a replay interval is repeated until all the operations are covered by replay intervals. The pseudocode of the proposed method is shown in Fig. 6.

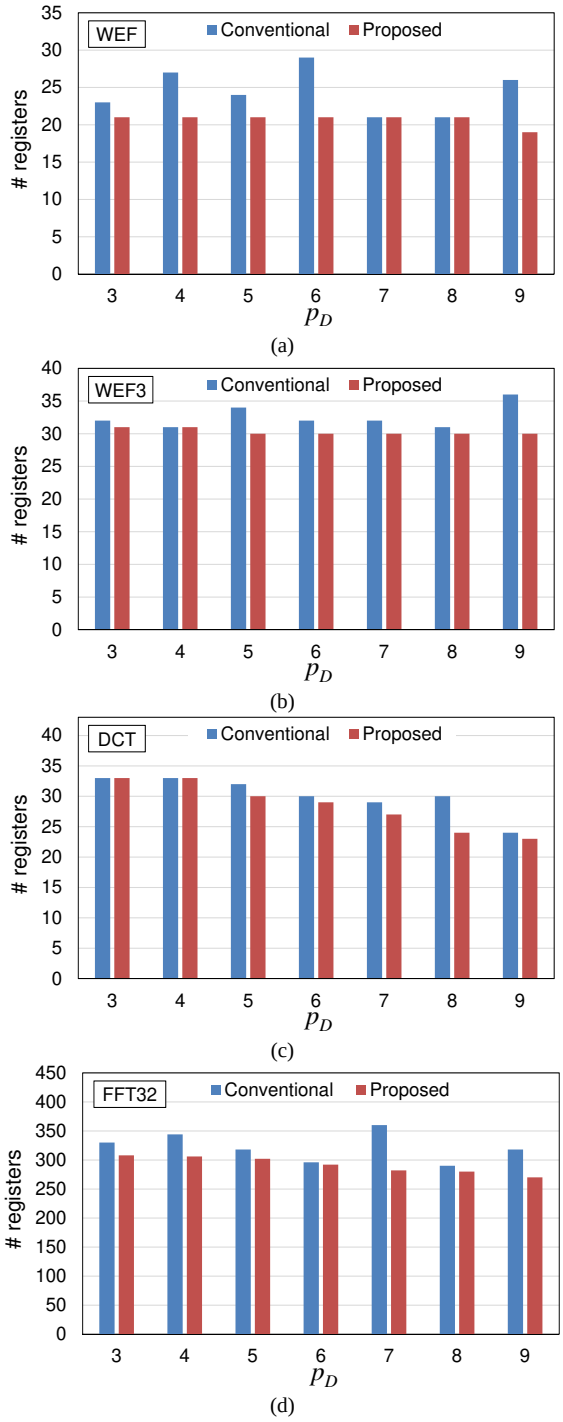

Fig. 7. The result of the register minimization. (a) WEF, (b) WEF3, and (c) DCT with 6 adders and 6 multipliers. (d) FFT32 with 40 adders and 40 multipliers.

IV. Experimental Results

The proposed register minimization method was implemented using Python programming language. The processing algorithms used in the experiments were the 5th order wave elliptic filter (WEF) [12] consisting of 8 multiplications and 26 additions, the WEF unfolded [13] by factor 3 (WEF3, 24 multiplications and 78 additions), 8-point 1D DCT [14] (DCT) consisting of 11 multiplications and 29 additions, and 32 point FFT (FFT32) consisting of 256 multiplications and 448 additions. The DMR schedules for the processing algorithms were obtained by list scheduling [15] assuming 6 adders and 6 multipliers are available for WEF, WEF3, and DCT, and 40 adders and 40 multipliers are available for FFT32. An addition takes 1 CC, a multiplication takes 2 CCs with pipelining, and an equality check takes 1 CC. It is assumed that an arbitrary number of

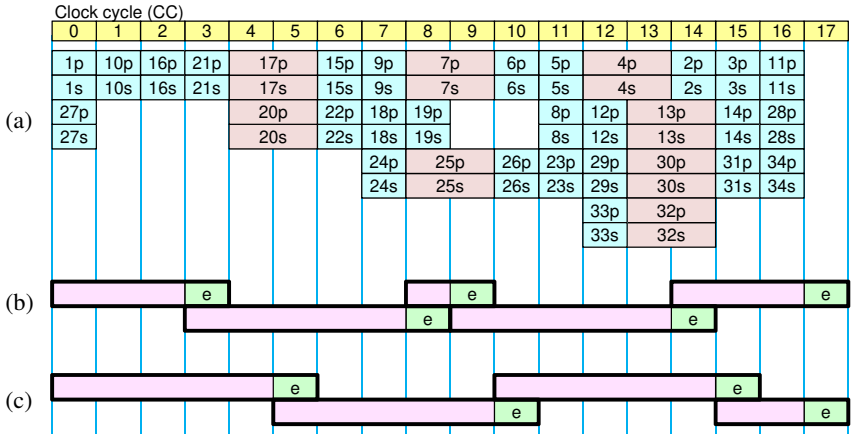

Fig. 8. The resultant set of replay intervals. (a) a DMR schedule for WEF with 6 adders and 6 multipliers. (b) the result obtained by the proposed method. (c) the result obtained by the conventional method.

comparators can be used for equality checks.

For comparison, as a conventional method, replay intervals of the length equal to p_D are selected from the beginning of the schedule. That is, the design with the conventional method consists of the replay intervals $Replay(0, p_D - 1)$, $Replay(p_D - 1, 2p_D - 2)$, $Replay(2p_D - 2, 3p_D - 3)$, and so on.

Figure 7 shows the minimized number of registers for p_D values from 3 to 9. In some cases the results of the proposed and the conventional methods are identical. However, the proposed method obtains DMR design with the smaller number of registers than the conventional method in many cases, and up to 27% reduction is achieved.

The selection of replay intervals for the case of WEF and $p_D = 6$ is shown in Fig. 8. While the proposed method uses more replay intervals than the conventional method, a DMR design with the smaller number of registers is obtained.

V. Conclusions

In this paper, a method to minimize registers in DMR design is proposed. By considering the replay style soft error correction, the delay penalty for the error correction, and the requirement for providing error-free input data for the replay, the proposed method greedily selects replay intervals. The proposed method obtains the DMR design with less registers in many cases, and the reduction is up to 27%.

The minimization of the cost other than registers, such as necessary multiplexors, and the minimization of comparators for error detection remain as future work. In this paper, registers are minimized for a given DMR schedule. For further minimization, optimizing the DMR schedule would be an interesting problem.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] K. Ito and T. Negishi, “Minimization of register area cost for soft-error correction in low energy DMR design,” Proc. Workshop on Synthesis And System Integration of Mixed Information Technologies, pp.56–51, 2015.

[11] M. Masuda, K. Kubota, R. Yamamoto, J. Furuta, K. Kobayashi, and H. Onodera, “A 65 nm low-power adaptive-coupling redundant flip-flop,” IEEE Trans. Nucl. Sci., vol.60, no.4, pp.2750–2755, 2013.

[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] K.K. Parhi and D.G. Messerschmitt, “Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding,” IEEE Trans. Computers, vol.40, pp.178–195, 1991.

[14] C. Loeffler, A. Ligtenberg, and G.S. Moschytz, “Practical fast 1-D DCT algorithms with 11 multiplications,” Proc. IEEE ICASSP ’89, pp.988–991, 1989.

[15] G.D. Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, New York, 1994.