システムＬＳＩ設計技術　106－11
(2002. 5. 24)

冗長化アルゴリズムにおける資源共有を考慮した耐故障データパス合成

尾塩　和亮†　　金子　峰雄†

† 北陸先端科学技術大学院大学 情報科学研究科　〒923–1292 石川県能美郡辰口町旭台 1-1
E-mail: †{k-oshio,mkaneko}@jaist.ac.jp

あらまし　本稿では，レジスタ転送レベルでの耐故障データパス合成について検討し，計算アルゴリズム三重化と多数決挿入を基礎とするオンライン誤り訂正可能データパスを提案している．特にデータパスを合成する上で障害となる結線複雑さを低減するための「多数決–書き戻し」構成を提案し，こうした誤り訂正モデルの下で，多数決器を含めた任意の単一構成要素故障に対して誤り訂正可能性 (3 つの対応する計算結果に少なくとも 2 つの正しい計算結果を含む) を保証するための，資源割り当てと多数決挿入の条件を明かにしている．多数決の挿入は三重化されたアルゴリズム間での資源共有を可能にし，単純なモジュール三重化に比べ，時間–資源効率の良いデータパスを合成できる．
キーワード　誤り訂正，三重化，多数決，データパス合成，ASIC，先行制約グラフ

Fault Tolerant Datapath Based on Algorithmic Redundancy and Voting

Kazuaki OSHIO† and Mineo KANEKO†

† Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Hokuriku
Asahidai 1–1, Tatsunokuchi, Ishikawa 923–1292, Japan
E-mail: †{k-oshio,mkaneko}@jaist.ac.jp

Abstract　In this paper, we propose a combination of triple algorithmic redundancy and vote–and–writeback configuration for realizing concurrently error correctable RT–level datapath of a specified computation algorithm. The vote–and–writeback configuration would contribute to reduce wire complexity around voters, compared with other conventional voting scheme. When a register assigned for a voted data is faulty, the vote–and–writeback configuration can not correct erroneous voted data on this register. It is interesting that, even though, fault tolerance of the datapath with respect to single fault for any constituent of the datapath can be guaranteed by this vote–and–writeback configuration and appropriate insertion of voters.
Key words　concurrent error correction, triple redundancy, vote, datapath synthesis, ASIC, dependence graph

1. Introduction

By the continuing progress of the fabrication technology, the possible feature size of each device is becoming smaller and the possible chip size becoming larger, which enable us to integrate huge number of transistors and nets in a single chip. For such highly integrated systems, failure rate becomes also higher, and faults at operation time may lead to serious failures. To prevent such failures, various approaches to fault tolerance have been proposed; for example, spatial redundancy such as triple modular redundancy and reconfiguration with spare resources, time redundancy such as rollback, and mixture of them. Algorithm–based fault tolerance (ABFT)[1] is another type of approach to fault tolerance, which introduces redundant algorithm for error–detection/correction to the original algorithm. ABFT has a great advantage in fault tolerance especially for computation algorithms in linear algebra with high regularity, such as matrix operations and signal processing. However, on the other hand, computation algorithms to which ABFT is applicable are heavily limited and also operations for error correction are somewhat complicated.

Antola, et al.[2] proposed high–level synthesis of data paths with concurrent self–checking abilities based on duplication of an input computation algorithm. They treated only the on–line error detectability, but not the issue of error correction. Hashimoto, et al.[3] proposed scheduling of duplicated tasks for fault tolerance. Their main target is the fault tolerance in multiprocessor systems, and assumptions adopted there, such as fail–stop ability of processing element, the

message–arrival triggered task execution, and reliable network, are not accepted for fault–tolerance in RT–level architectural design.

In this paper, we focus our attentions on concurrently error correctable ASICs for applications which need highly real–time responses. We introduce triple algorithmic redundancy and vote operation to satisfy real–time–ness and to enable us to apply our scheme to various types of computation algorithms; algorithms may contain not only linear algebra but also nonlinear operations, conditional branches, etc. And we propose a novel error–correction scheme in the context of datapath synthesis.

At first, we define behavior in nominal and faulty state for each datapath element. Especially, vote operation has a special feature in our scheme, that is, it receives input data to be voted only from registers and it writes the result back, when it detects disagreement between input data, to the register which holds incorrect data. In our error–detection/correction model, interconnection fault is equivalent to an element fault which drives the interconnection, and multiplexer fault is equivalent to an element fault which receives multiplexer's output. Hence we can conclude that k fault tolerant system for faults on functional units, registers and voters becomes a k fault tolerant system for faults on functional units, registers, voters, multiplexers and nets. Next we show conditions for datapath, which executes triplicated algorithm, to possess single fault tolerant property. Here, we introduce a special subgraph which consists of directed paths from voted data to voted data for the algorithm, and we derive necessary and sufficient condition for datapath to tolerate any single fault and to continue producing at least two correct outputs among three corresponding outputs in triplicated algorithm. As the result, relation between resource sharing and vote insertion is clarified, which can be used to generate fault tolerant RT level description in the context of datapath synthesis. Lastly we demonstrate how our scheme can generate better solutions in the trade–off between required resource and makespan.

2. Error–Correctable Datapath Based on TRAV

2.1 Datapath Model and Fault/Error Model

Datapath considered in this paper contains "functional units", "registers", "multiplexers", "nets" and "voters".

- A functional unit has multiple inputs and a single output.
- A register has a single input and a single output.
- A multiplexer has multiple inputs and a single output which is an input of either a functional unit, register or voter.
- A net has a single input terminal(an output terminal of a functional unit, a register or a voter) and multiple output terminals(each of which is an input terminal of a functional unit, a register, a multiplexer or a voter).
- A voter has three inputs and two outputs; one output is a voted result and the other is a control signal which indicates inconsistent input.

Fault is considered on every constituent shown above. Output(s) of each faulty constituent may be erroneous. We assume that output of each constituent, even if it is faulty, can be seen as a logically meaningful value, i.e., logical 0 or 1 for every signal bit, from other constituents. When the number of faulty constituents is less than or equals to k, we call it as k faults.

2.2 Triplication with Votes

A computation algorithm is $\langle G, D_P \rangle$, where $G = (V, A)$ is a dependence graph and D_P is a set of primary outputs.

- V is the disjoint union of a set of operations O and a set of variables D, and A is a set of directed arcs form an operation to a variable or from a variable to an operation.
- Every operation in O has its indegree at least one and its outdegree exactly one.
- Every variable in D has its indegree at most one. A variable having indegree 0 is called *primary input*, the other variable having indegree exactly one is called *internal variable*. We denote a set of primary inputs as D_I and a set of internal variables as D_O.
- There is an one–to–one correspondence between O and D_O, and $d(o)$ denotes a variable generated by an operation $o \in O$, whereas $o(d)$ does an operation which generates a value for a variable $d \in D_O$. On the other hand, $pred(o)$, $o \in O$, denotes a set of input variables used by o, and $succ(d)$, $d \in D$, denotes a set of operations which use a variable d.
- A subset of D_O is specified as a set of *primary outputs*. we denote a set of primary outputs as D_P.

In this paper, we focus our attention on on–line error correctable datapaths. Among various algorithm–level redundancies, we choose triple algorithmic redundancy and error correction by voting as our basic strategy for error correction, because of its simplicity in error–correcting mechanism and its wide applicability without depending the type of input algorithms.

Given an input computation algorithm $\langle G, D_P \rangle$, output of our problem is a Triple Redundant computation Algorithm with Voting (TRAV, in short) $\langle G_T, D_{TP} \rangle$ and its hardware–temporal mapping. A dependence graph of a TRAV $G_T = (V_T, A_T)$ contains three copies of the input

algorithm $G_1 = (O_1 \cup D_1, A_1)$, $G_2 = (O_2 \cup D_2, A_2)$ and $G_3 = (O_3 \cup D_3, A_3)$, votings, output variables of votings and voting–related arcs.

$$V_T = \left(\bigcup_{i=1}^{3} (O_i \cup D_i) \right) \cup O_V \cup D_V$$
$$A_T = A_1 \cup A_2 \cup A_3 \cup A_V$$

where O_V is a set of votings, D_V a set of voting results, and A_V a set of voting–related arcs.

For a system whose dependability relies on voting, undependability of a voter generating a primary output becomes its drawback, and in many cases we need to use a sufficiently reliable voter for it. However, a primary output is a kind of interface between a computation algorithm and outside world, and the requirement to this interface may vary depending on applications and outside world. Hence, we now introduce a somewhat relaxed definition on "fault tolerance" as follows.

[Definition 1] A RT level datapath which performs TRAV is called "k–fault tolerant" when, for each primary output of an input computation algorithm, at least two of corresponding three data in TRAV are error free even if any k or less constituents of the datapath are faulty.

2.3 Error–Correction Model

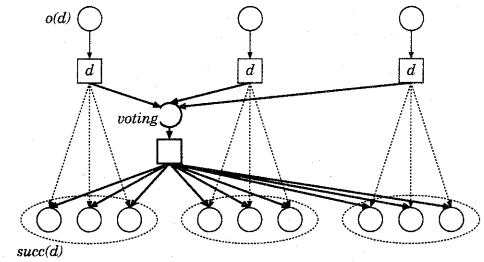
With respect to vote insertion, we can consider several alternatives. Fig.1 shows two typical and conventional vote insertion styles. In a single–vote configuration (Fig.1(a)), the possibility of fault on the vote becomes a bottle-neck for the system reliability. In a triplicated–vote configuration (Fig.1(b)), the hardware overhead in the number of voters and redundant nets may possibly become larger.

With the purpose of reducing network complexity in RT level architecture, we propose a vote–and–writeback configuration (Fig.2). In this model, if one of the corresponding three data is claimed to be erroneous by a vote, it is replaced with the vote result.

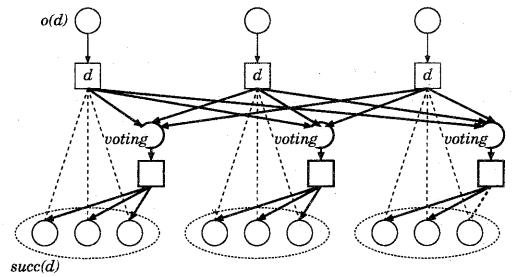
Corresponding to this vote–and–writeback model, we assume the following with respect to RT level architecture (also refer to Fig.3).

- A voting is always performed on three corresponding data from each of $G_i, i = 1, 2, 3$, stored at registers.
- A voter is a three–input, two–output constituent of datapaths, where one output is a voting result and the other output is the code to represent which one of three input disagrees with others.
- A faulty voter may possibly outputs erroneous voting result and/or erroneous code to identify erroneous input.

Under the vote–and–writeback configuration and assumptions imposed above, the following holds.



(a) Single vote



(b) Triplicated vote

Fig. 1 Two conventional configurations of vote insertion.

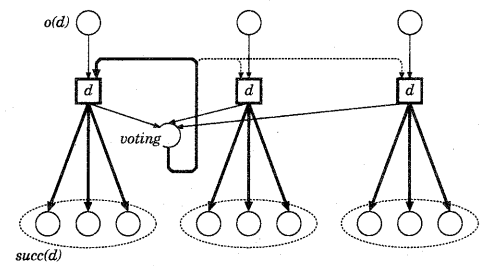


Fig. 2 Vote and write-back.

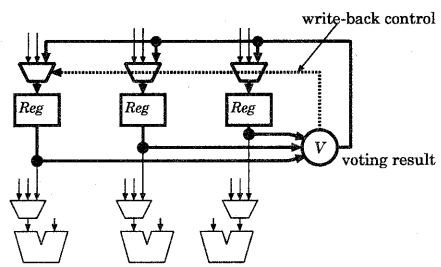


Fig. 3 Hardware around voting.

- A fault on a net can be treated equivalently as a fault on a module(functional unit, register or voter) whose output is the input terminal of the net.
- A fault on a multiplexer can be treated equivalently as a fault on a module(functional unit, register or voter), one of whose inputs is the output terminal of the multiplexer.

As a result, we can conclude the following.

[Lemma 1] If a RT level datapath is k fault tolerant with

respect to faults on functional units, registers and voters, then the datapath is k fault tolerant with respect to faults on any constituents, i.e., functional units, registers, voters, multiplexers and nets.

3. Condition for Single Fault Tolerance

As we have mentioned before, voting will be done on data at registers, not on data appeared at the output of functional units. We introduce some definitions for convenience's sake.

[Definition 2] If corresponding three data of a variable are voted, the data is called a *voted data*. Primary inputs are treated as voted data, while they need not to be voted.

[Definition 3] For a voted data or a primary output data d, a subgraph of DG, which contains d and all vertices (unvoted or voted data or operation) and arcs from which d is reachable without passing through any other voted data, is called a *corn* induced by d. d is called the output of the corn and other voted data in the corn is called the input of the corn.

Note that any voted data (except primary input data) is contained in two or more corns, for one of which as an output and for the other as input. An *output value of a corn* is the voted result if its original output is not coincide with other corresponding two original values, otherwise its original value.

[Definition 4] The corn decomposition in each copy of input dependence graph is the same, since the topology and voted data are identical for all three copies of the dependence graph. Three corresponding corns are called *a stage*. *Outputs of a stage* are the three outputs of corns of the stage.

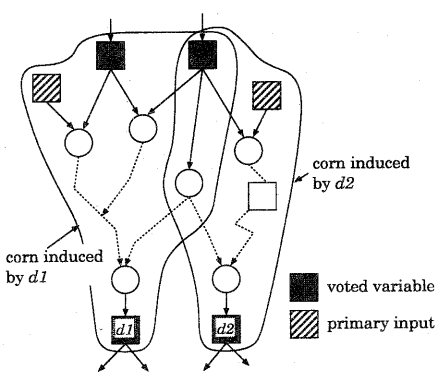


Fig. 4 Example of a corn.

[Definition 5] If outputs of a stage are error-free, the stage is called *fair*. If outputs of a stage contains exactly one erroneous value, the stage is called *marginally fair*. If outputs of a stage contains more than one erroneous values, the stage is called *unfair*.

Now we assume without loss of generality that an input dependence graph is irredundant, i.e., for any vertex (operation or data) in the graph, there exists at least one primary output which is reachable from the vertex. For such an irredundant dependence graph, a RT level datapath is fault tolerant if and only if every stage is either fair or marginally fair for faults under consideration.

The following shows conditions for a RT level datapath to be single fault tolerant.

[Theorem 1] A RT level datapath based on TRAV and vote-and-writeback model is single fault tolerant if and only if the following two conditions are satisfied.

C1: Any two corns in a stage do not share a same resource(functional unit, register, etc.).

C2: Any two votings for inputs and output of a stage do not share a same voter.

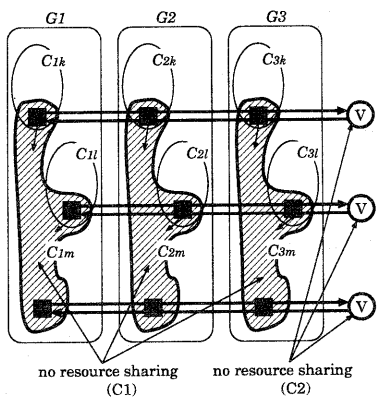


Fig. 5 Illustration of the conditions *C1* and *C2*.

The necessity of *C1* and *C2* is straightforward. (1) If two corns in a stage share a resource r and r is faulty, the original outputs (before voting) of these two corns may possibly be erroneous. Hence, voting can not correct error and the stage becomes unfair. (2) If two set of corresponding three inputs (a, a', a'') and (b, b', b'') to a stage share a voter v and v is faulty, then one of a, a' and a'' in a corn of the stage and one of b, b' and b'' in another corn of the stage may possibly be erroneous. As a result, the original outputs (before voting) of these two corns may possibly be erroneous. Hence, voting can not correct error and the stage becomes unfair. (3) If a set of corresponding three inputs (a, a', b'') and a set of corresponding three original outputs (before voting) (y, y', y'') of a stage share a voter v and v is faulty, then one of a, a' and a'' in a corn of the stage (say a) may possibly be erroneous and hence one of y, y', y'' in the same corn with erroneous input (that is, y) is erroneous before voting. In addition to this, the voting for (y, y', y'') may possibly overwrite correct

value (y' or y'') with an incorrect value. As a result, the stage becomes unfair.

The sufficiency of *C1* and *C2* is proven by showing the following two auxiliary lemmas, but their details are omitted for lack of space.

[Lemma 2] If C1 and C2 are satisfied, then three original outputs (before voting) of every corresponding three corns (i.e., every stage) contain at most one erroneous data.

[Lemma 3] If C1 and C2 are satisfied and three original outputs (before voting) of corresponding three corns (i.e., a stage) contain at most one erroneous data, then the stage is either fair or marginally fair.

4. Synthesis Problem and Design Examples

Fault tolerant datapath synthesis problem, where its fault tolerance is based on TRAV and vote–and–write-back model, is summarized briefly as follows. An input instance of the problem is a computation algorithm $\langle G, D_P \rangle$ and resource information, and output is a TRAV $\langle G_T, D_{TP} \rangle$, a schedule of $O_1 \cup O_2 \cup O_3 \cup O_V$, and resource assignment; $O_1 \cup O_2 \cup O_3$ to a set of functional units, O_V to a set of voters and $D_1 \cup D_2 \cup D_3$ to registers. The resource assignment and vote insertion must satisfy conditions *C1* and *C2* of Theorem 1.

Development of a synthesis method and related algorithms is left as a future problem. Here we only demonstrate how resource sharing between different copies of a computation algorithm contributes to the optimality of datapath design. Fig.6 shows the differential equation benchmark which is used as an input computation algorithm, where loops in the original algorithm are broken to form a DAG. If we apply TRAV without resource sharing between copies of a computation algorithm, operations in each copy is to be executed sequentially (makespan in this case is 15) even if four or five ALUs are available. Fig.7 shows two schedules under resource sharing, one in case that four ALUs are available (its makespan is 12) and the other in case of five ALUs (its makespan is 10).

Fig.8 shows the dependence graph of a modified fourth-order Jaumann wave digital filter which is used as the second example, where loops in the original algorithm are broken to form a DAG. If we apply TRAV without resource sharing between copies of a computation algorithm, operations in each copy is to be executed sequentially (makespan in this case is 21) even if four or five ALUs are available. Fig.9 shows two schedules under resource sharing, one in case that four ALUs are available (its makespan is 18) and the other in case of five ALUs (its makespan is 16).

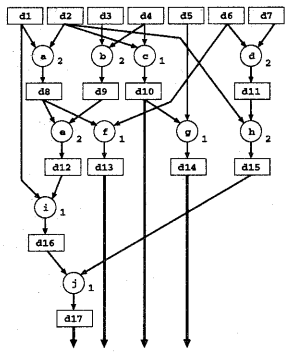


Fig. 6 Differential equation benchmark.

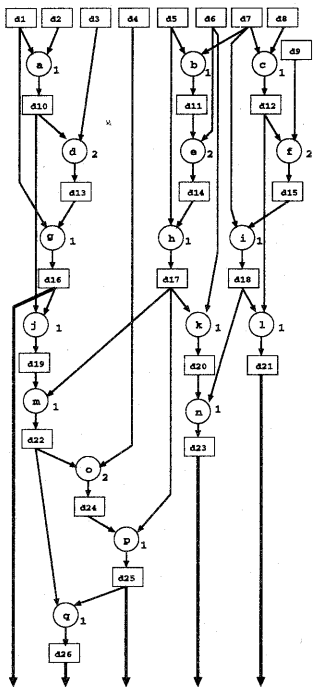


Fig. 8 Fourth-order Jaumann wave digital filter benchmark.

5. Conclusion

In this paper, we propose the vote–and–writeback model in Triple Redundant computation Algorithm with Voting(TRAV) for fault tolerant datapath. The vote–and–writeback configuration would contribute to reduce wire complexity around voters, compared with other conventional voting scheme. When a register assigned for a voted data is faulty, the vote–and–writeback configuration can not correct erroneous voted data on this register. It is interesting that, even though, fault tolerance of the datapath with respect to single fault for any constituent of the datapath can be guaranteed by this vote–and–writeback configuration and

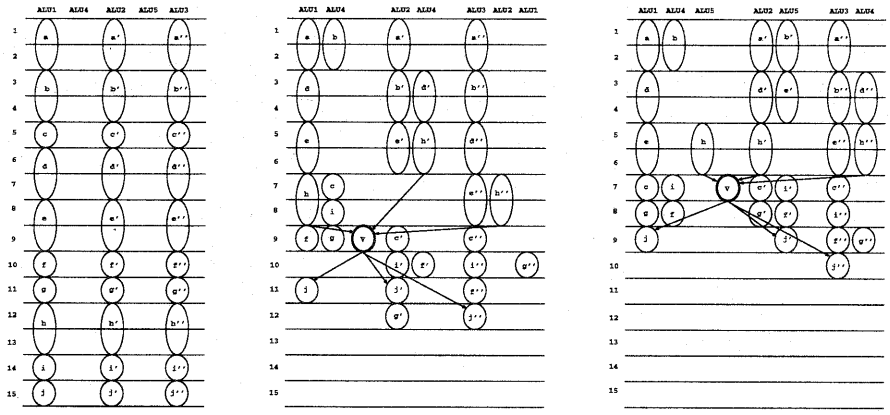


Fig. 7 Single fault tolerant schedule and assignment of modified differential equation benchmark. Left: without resource sharing, Center: in case of four ALUs, Right: in case of five ALUs.

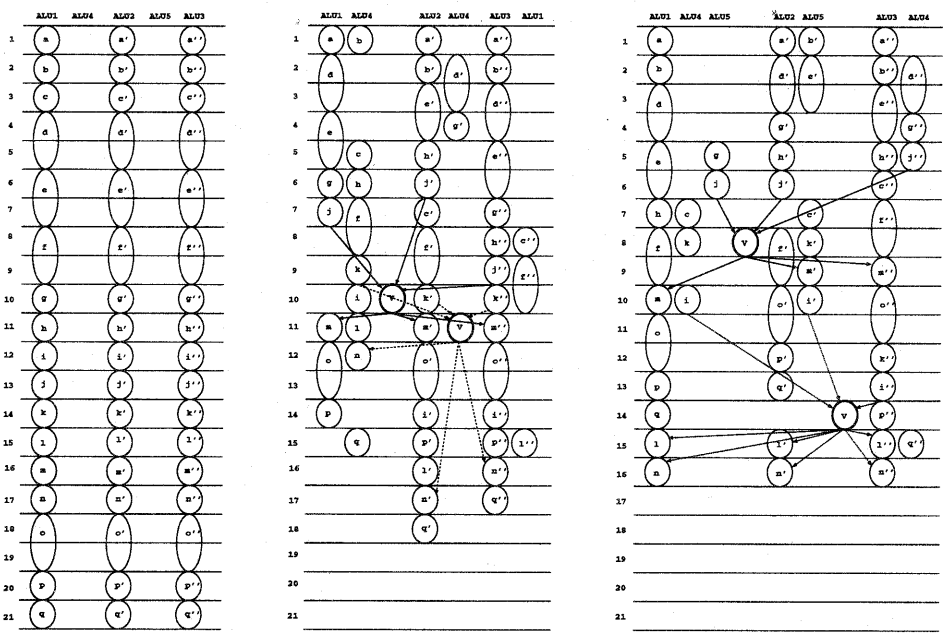


Fig. 9 Single fault tolerant schedule and assignment of fourth-order Jaumann wave digital filter benchmark. Left: without resource sharing, Center: in case of four ALUs, Right one: in case of five ALUs.

appropriate insertion of voters.

In TRAV, vote insertion, assignment and schedule interact each other, which makes the design problem being complicated. Development of a synthesis method and related algorithms is left as a future problem.

Acknowledgement – This work is partly supported by Research Body CAD21, Tokyo Institute of Technology, Japan.

References

[1] Kuang-Hua Huang, and Jacob A. Abraham, "Algo-rithm-Based Fault Tolerance for Matrix Operations", IEEE Trans. Computer, Vol.c-33, No.6, June 1984.

[2] Anna Antola, Vincenzo Piuri, and Mariagiovanna Sami, "High Level Synthesis of Data Paths with Concurrent Error Detection", Proc. IEEE Symp. DFT in VLSI Systems, pp.292-299, 1998.

[3] Koji Hashimoto, Tatsuhiro Tsuchiya, and Tohru Kikuno, "Effective Scheduling of Duplicated Tasks for Fault Tolerance in Multiprocessor Systems", IEICE Trans. Inf. and Syst., Vol.E85–D, No.3, pp.525–534, 2002.

[4] Ramesh Karri, Kyosun Kim, and Miodrag Potkonjak, "Computer Aided Design of Fault-Tolerant Application Specific Programmable Processors", IEEE Trans. Computer, Vol.49, No.11, pp.1272-1284, November 2000.