Algorithm-Based Fault Tolerant Systems Based on Graph-Theoretic Error Occurrence/Propagation Models

by

Choon-Sik PARK

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Associate Professor Mineo Kaneko

School of Information Science
Japan Advanced Institute of Science and Technology

March 2000

Abstract

High performance parallel computer systems become feasible solution of computationally intensive problems by the advent of cost-effective VLSI components in the past few years. Since the probability of one or more processors to become faulty in such multiple processor systems is quite large, it is desirable for improving the reliability of them to build some on-line fault tolerance features into them. However, the requirements for high performance and fault tolerance are seemingly contradictory: parallel architectures and algorithms have been developed to achieve maximum utilization of each of processors, while fault tolerance requires redundant computations and checking operations to ensure that the computation results are correct. To incorporate fault tolerance into multiprocessor systems at lower cost, several ABFT techniques have been proposed. However, most of these discussions are target dependent and less effort has been made at the generalization.

The objective of this research is to construct some general model which can be used for both analysis and synthesis of ABFT systems. Fault detectability/locatability under some practical error occurrence/propagation models and formal design methods of checking scheme of ABFT systems are also discussed on this general model. The essentials of ABFT technique are to encode data at system level and to modify the target algorithm to operate on the encoded data. To analyze and to control fault detectability/locatability of such a system, the error occurrence/propagation model at the algorithm level plays an important role. The model for ABFT system considered here can fully utilize a specified error occurrence/propagation model by using data dependency, and it can give us tighter conditions for fault detectability/locatability than previous models do. In the turn for synthesis, these properties contribute to the cost-effective checking scheme by reducing redundant checking operations.

In the result, the analysis/synthesis model and some relevant discussions done in this research will provide important bases for reliable parallel computing systems.

Acknowledgments

The author wishes to express his sincere gratitude to his principal advisor Associate Professor Mineo Kaneko of Japan Advanced Institute of Science and Technology for his constant encouragement and kind guidance during this work. The author would like to thank his advisor Professor Milan Vlach of Japan Advanced Institute of Science and Technology for his helpful suggestions and kind encouragements.

The author devotes his sincere thanks to Professor Yasusi Hibino of Japan Advanced Institute of Science and Technology, Associate Professor Kunihiko Hiraishi of Japan Advanced Institute of Science and Technology and Professor Eiji Fujiwara of Tokyo Institute of Technology for their valuable suggestions and discussions. The author would like to thank Research Associate Satoshi Tayu of Japan Advanced Institute of Science and Technology for his helpful discussions. The author is also grateful to all who have affected or suggested his areas of research.

Finally, the author would like to thank his friend Hyo-Yeong Bae and his family for their love and continuous encouragements.

Contents

List of Figures

List of Abbreviations

- **ABFT**(**A**lgorithm-**B**ased **F**ault **T**olerance) is a new technique for tolerating faults at lower cost, and has concurrent error detection and fault location capability.
- **CED**(**C**oncurrent **E**rror **D**etection) is on-line fault tolerant technique for detecting faults, and has been used in fault tolerant signal processing applications.
- **DC**(**D**ata-**C**heck) graph: represents the relation of data elements and checks, and gives an useful information about the checking schemes of fault tolerant system.
- **DG**(**D**ependence **G**raph) represents the data dependency between operations associated with data element, or between operation and primary input or primary output, for a given algorithm.
- **ED**(**E**rror-**D**ata) graph: is an undirected bipartite graph, and represents the relation of error patterns and data elements.
- **EMPDC**(**E**xtended **M**odified **P**rocessor-**D**ata-**C**heck) graph is to be extended from MPDC graph to introduce the redundancy for computing data element to be compared to the sum of data elements in a check and to map the checking operation to system processors.
- **FIR**(**F**inite **I**mpulse **R**esponse) filter is always stable and can be made to have linear phase response which is characteristic that makes it extremely attractive in audio and sonar applications.
- **IIR**(**I**nfinite **I**mpulse **R**esponse) filter is more efficient than FIR filter in analog equivalence and cost effectiveness.
- **MID**(**M**ultiple-**I**nputs **D**riven) model is a sophisticated error generation/propagation model used in this research.
- **MPD**(**M**odified **P**rocessor-**D**ata) graph is to be modified from PD graph to introduce data dependency between computation results.
- **MPDC**(**M**odified **P**rocessor-**D**ata-**C**heck) graph represents ABFT system which is constructed by using the checking scheme based on MPD graph model.
- **PD**(**P**rocessor-**D**ata) graph represents the relation of processors and data elements, and gives the information about the architecture after mapping operations to a set of processors.
- **PDC**(**P**rocessor-**D**ata-**C**heck) graph is a tripartite graph which consists of PD graph and DC graph, and has been used by many researchers in analyzing and designing ABFT system.

- **PE**(**P**rocessing **E**lement) is a circuit block in array architecture, or is a processor in multiprocessor system.
- **SFD**(**S**ingle-**F**ault **D**etection) is the process of recognizing that a single-fault has occurred.
- **SFL**(**S**ingle-**F**ault **L**ocation) is the process of determining where a single-fault has occurred.
- **SFL/TFD**(**S**ingle-**F**ault **L**ocation/**T**wo-**F**ault **D**etection) is the process of determining where a single-fault has occurred and recognizing that a two-fault has occurred.
- **SID**(**S**ingle-**I**nput **D**riven) model is a simple error generation/propagation model used in this research.
- **TFD**(**T**wo-**F**ault **D**etection) is the process of recognizing that a two-fault has occurred.
- **VLSI**(**V**ery **L**arge **S**cale **I**ntegration) is an integrated circuit containing more than 10,000 logic gates or more than 30,000 transistors.
- **WCC**(**W**eighted **C**hecksum **C**ode) is a code for both detecting and correcting error in signal processing applications.

Chapter 1

Introduction

High performance parallel computer systems are commonly used for signal processing and other computationally intensive problems which require high-speed data processing. A fault in the system can have damaging consequences on the result of a computation. Also the probability of one or more processors to become faulty in such multiple processor systems is quite large. Therefore, it is desirable for improving the reliability of them to build some on-line fault tolerance features into them. However, the requirements for high performance and fault tolerance are seemingly contradictory: parallel architectures and algorithms have been developed to achieve maximum utilization of each of processors, while fault tolerance requires redundant computations and checking operations to ensure that the computation results are correct. Algorithm-based fault tolerance(ABFT) is one of techniques for solving such problems, which provides concurrent error detection and location capability to the system [1].

To incorporate fault tolerance at lower cost without sacrificing the performance, various ABFT techniques have been proposed, and also model-based analysis methods for these ABFT systems have been investigated [1], [2], [3], [4], [5], [6], [7], [19], [20], [23], [25], [26], [29], [31], [32], [34], [36]. There have been many applications of these techniques to a variety of problems including FFT [17], [19], [20], sorting, signal processing applications [6], [11], [12], [14], [29], such as finite impulse response(FIR) filtering, and matrix operations [1], [6], [14], [21], [23], [26], [34]. It has also been applied to various architectures such as array [6], [11], [12], [14], [15], [19], [20], [23] and hypercube [21].

One of the main goals of research in ABFT is to design cost-effective systems which have fault detectable(or locatable) property so that the computation complexity and the number of checks are minimized. Designing k-fault locatable or detectable systems involves many degrees of freedom. One could assume that the architecture is not given *a priori*. In this case one could add checks to the algorithm to make it error tolerant and then project its data dependence graph(DG) to obtain the optimal fault tolerant architecture. Vinnakota and Jha [6] proposed two-stage approach to the synthesis of ABFT systems: (1) a system-level code is chosen to encode the data used in the algorithm, (2) the optimal architecture to implement the scheme is chosen by using DG. Liu and Jen [23] presented a systematic design methodology which maps a matrix arithmetic algorithm to a fault-tolerant array processor: input data of an algorithm are coded in weighted checksum code(WCC) and the DG of the modified algorithm is mapped to array processor. On the other hand, one could assume that the algorithm and architecture are already given and that the checks must be added for some desired fault tolerance.

Many works have been done on the formal analysis and design of ABFT systems. Banerjee and Abraham [5] proposed a graph-theoretic model to represent ABFT systems: an ABFT system is modeled by an undirected tripartite graph which is called processor-data-check(PDC) graph. It consists of three sets of nodes each of which corresponds to processors, data elements or checks and edges between processor and data or data and check. The PDC graph can be divided into two bipartite graphs: processor-data(PD) graph and data-check(DC) graph which represent PD relationships and DC relationships, respectively. Designing PD relationships can be said to be a synthesis for ABFT system, while designing DC relationship can be called a design for ABFT system. Also, demonstrating the ability of the desired fault tolerant system when both PD graph and DC graph are given, can be said to be an analysis for ABFT system. The PDC graph model has been used by several researchers in synthesizing, designing and analyzing ABFT systems. Nair, Abraham and Banerjee [4] proposed a matrix-based model which was derived from the graph-theoretic model. Banerjee and Abraham in [5] also gave a construction methods for ABFT systems by using the graph-theoretic model. In any of these methodologies, a systematic procedure to design DC relationships which can detect and locate a specified number of faults has attracted a lot of attention due to the important role it plays in synthesizing, designing and analyzing ABFT systems. Sitaraman and Jha [3] showed how to design the DC relationship for error detection and location.

However, the conventional graph-theoretic models have not addressed error occurrence and error propagation for a given system, which result in redundant error patterns in designing and analyzing ABFT systems. These redundant error patterns cause the increase of the complexity and the number of checks in designing and analyzing ABFT systems. To exclude such redundant error patterns, we introduce data dependent information between computation results computed by processors. Since data dependency gives an useful information for error occurrence/propagation, we can simplify the analysis procedure and reduce the number of checks. An analysis model based on modified processor-data(MPD) graph is proposed, and it is shown that the number of error patterns to be considered can be reduced by utilizing data dependency between computation results.

We employ two error occurrence/propagation models named as single-input driven(SID) model and multiple-inputs driven(MID) model for the MPD graph model. The SID model is such a model that an erroneous input to a computation will always result in erroneous computation result regardless of the other inputs nor the status of the processor. In this case, once we construct a set of data elements reachable from each data element, each one of error patterns can be generated by simple union operation to properly selected sets of reachable data elements, and we can drastically reduce the number of error patterns to be considered compared to the conventional PDC graph model. The MID model is more sophisticated with taking into account of some possibilities in the practical situations: a computation result with multiple erroneous inputs may possibly be error-free, and also a faulty processor may possibly generate error-free computation result when some of inputs are erroneous. As a result, the set of error patterns to be considered with MID model becomes a superset of the one with SID model, but still a subset of the one from the conventional PDC graph model. In general, the computational complexity and the number of checks in analyzing and designing ABFT systems increase as the number of error patterns increases. While the complexity and the number of checks tend to increase compared to the MPD graph with the SID model in compensation for improving the accuracy of error propagation model, the effectiveness of MPD graph model with MID model holds good

compared to the conventional PDC graph model.

On the other hand, the redundant computations and checking operations are usually a part of an ABFT system and are likely to be performed on the system processors. However, the problem of how to map checks to system processors has been little concerned in designing ABFT system. Nair, Abraham and Banerjee [4] introduced check evaluating nodes in their graph model and showed the way how to analyze such a system for fault tolerance. Yajnik and Jha [7] used an extended PDC graph model for considering processors computing checks to be a part of the ABFT system. They proposed a deterministic solution to concurrent error detection and fault location with graceful degradation, and presented a general method for designing one-fault locating/s-fault detecting ABFT systems. However, their graph model did not show information about how to map checks to system processors, and also they did not consider the problem of designing cost-effective ABFT systems such that the number of checks and redundant computations are reduced. We propose a novel strategy for mapping checks to system processors with minimizing the number of checks and redundant computations so that fault tolerant capability of the system is still maintained even after some permanent faults are detected.

Furthermore many efforts have been made in synthesizing ABFT systems on various VLSI array architectures [6], [11], [12], [14], [15], [19], [20], [21], [23]. The conventional fault tolerant schemes for array architectures have been mainly concentrated on concurrent error detection(CED) schemes. Gupta and Bayoumi [11] proposed a novel CED scheme termed as logarithm based on-line error detection which is based on the use of logarithmic coding for inputs and results in a self-testing systolic cell. Vinnakota and Jha [6] proposed a method for synthesizing single-fault detectable ABFT system from DG of FIR filtering by introducing an useful checking scheme. But they did not attend to the fault location which is an important key to correcting errors or reconfiguring system for permanent faults. On the other hand, Kung [10] presented error detection and correction based on interleaved DG. The idea is to perform the same computation twice in adjacent PEs at two different but close enough time periods and then compare the results. If they match there is no fault. Otherwise a roll-back is necessary to correct the fault. However some faults can not be exactly located when such faults are permanent. Also a fault in checking operation which is to compare two results: one is *primary* output computed in a processing element(PE), the other is *redundant* output computed in adjacent PE, was not considered, that is, checking operations were assumed to be fault-free. Cosentino [12] proposed a scheme of concurrent error correction in systolic architecture of FIR filtering at a cost of halving the maximum throughput rate by performing the same computation twice in adjacent processing elements and comparing such two results. Thus the conventional schemes have been mainly proposed in the area of concurrent error detection and correction. While less efforts have been made in fault location which plays an important role in the area of reconfiguration of the system to bypass the faulty processor. The problem of locating faults in systolic array system can not be solved in simple schemes. To solve the problem, more complex fault tolerant schemes which are considered both *time* redundancy and *hardware* redundancy, are required. We present a method of designing fault tolerant FIR filter on systolic architecture by using the scheme for synthesizing ABFT systems based on an extended modified processor-data-check(EMPDC) graph model.

Fault tolerance involves four steps: (1) detection of errors due to a fault at some processor(or module) output, (2) correction of the errors, (3) identification of the faulty processor, and (4) reconfiguration of the system to bypass the faulty processor. In this

thesis, we will concentrate on (1) and (3) due to playing an important role in (2) and (4). The correction of a error can be achieved by *roll back* technique or the identification of the error which provides an useful information for identifying the faulty processor. The reconfiguration can be achieved by using spare processors and switch modules, or applying graceful degradation techniques.

The rest of this thesis is organized as follows. In Chapter 2, several terms with regard to ABFT systems are defined, and k-fault detectability and k-fault locatability in terms of error patterns are discussed. Two error models to be employed in this research and an analysis model based on MPD graph are present in Chapter 3. For each of two error models: SID model and MID model, checking schemes and design examples for single-fault detectable and locatable ABFT systems are represented in Chapter 4 and Chapter 5. In Chapter 6, a novel strategy for mapping checks to system processors and fault tolerant FIR filter as a design example are discussed. Finally, Chapter 7 is used for conclusions.

Chapter 2

Terminologies with regard to ABFT systems

2.1 Fault, Error and Check

Now, we will define faults, errors, and checks(checking operations) with regard to ABFT systems. The basic definitions are based on [1], [4] and [5].

A *fault* is any condition that causes a malfunction in processor(s). An *error* is any discrepancy between the expected result of an operation and the actual result of the operation. A fault in a processor is assumed to be manifested as an error in one or more data elements affected by it. In general, the problem of detecting faults is translated into the problem of detecting errors in computation results. However, we must note that certain types of faults may not produce any error at all. If a particular fault does not produce any error in data elements computed by a processor, the fault is said to be *unobservable* and the presence of these faults is disregarded in this research. A collection of all faulty processors is called a *fault pattern*. Fault patterns consisting of k or fewer elements(faulty processors) are called *k-fault*. On the other hand, a collection of all erroneous data elements is called an *error pattern*.

A *check* is any combination of hardware and software procedures performed on the data elements to generate an output either 1 or 0. The set of data elements checked by a check is called its *data set*. A (g, h) check is defined on g data elements such that (1) the check is correct(either outputs 0 or 1) if the number of erroneous data elements among these g data elements does not exceed h, and (2) the check is invalid(may output 0 or 1) if more than h data elements are erroneous. We assume that the capability of a check is limited to a $(g,1)$ check, and its behavior is as follows:

$\mathcal{C}1$. A check outputs a 1 if there is exactly one data element in its data set being in error.

$\mathcal{C}2$. A check outputs a 0 if there are no errors in the data elements in its data set.

$\mathcal{C}3$. A check is unpredictable if the number of erroneous elements in its data set is greater than one.

The outputs of the checks in the system can be represented as a finite binary sequence which is usually called the *syndrome*.

2.2 k-Fault Detectability and Locatability

We define k-fault detectability and locatability as follows.

Definition 2.1 (k-Fault Detectability) *An ABFT system is said to be k-fault detectable if for every error pattern induced by k-fault, there is at least one check that certainly outputs 1.*

Definition 2.2 (k-Fault Locatability) *An ABFT system is said to be k-fault locatable if for any pair of error patterns, one is induced by a fault pattern in k-fault and the other is induced by any other fault pattern in k-fault, there is some check that certainly gives a different output.*

Now, we introduce several notations to describe k-fault detectability and locatability in terms of error patterns. f_i^l is the l-th set of i processors and indicates one fault pattern of the size i. F_i:

$$F_i = \bigcup_{l=1}^{C(M,i)} \{f_i^l\} \tag{2.1}$$

is the set of all fault patterns of their size exactly i, where $C(M,i)$ is the number of combinations which have i processors for a given system with M processors, that is, $C(M,i) = \frac{M!}{(M-i)!i!}$. And F^k:

$$F^k = \bigcup_{i=1}^{k} F_i \tag{2.2}$$

is the set of all k-fault patterns.

On the other hand, e_i^{lj} is a subset of data elements and indicates one error pattern induced by f_i^l. e_i^l:

$$e_i^l = \{e_i^{l1}, e_i^{l2}, \cdots, e_i^{lj_l}\} \tag{2.3}$$

is the set of all error patterns induced by a fault pattern f_i^l. E_i and E^k:

$$E_i = \bigcup_{l=1}^{C(M,i)} e_i^l \tag{2.4}$$

$$E^k = \bigcup_{i=1}^{k} E_i \tag{2.5}$$

are the sets of error patterns induced by fault patterns in F_i and those in F^k, respectively.

Now, we describe fault detectable and locatable system with regard to ABFT systems. To simplify the notations, let f_i be a fault pattern in F^k and let $e_{iu} (\in E^k)$ be the u-th error pattern induced by a fault pattern f_i. To describe fault detectability and locatability, first we introduce an *undirected bipartite* graph $G_{ED}(V_{ED}, E_{ED})$ which describes the relation between error patterns and data elements. The set of vertices $V_{ED} (= E^k \cup V_d)$ denotes the set of error patterns (E^k) and the set of data elements (V_d), and the set of edges

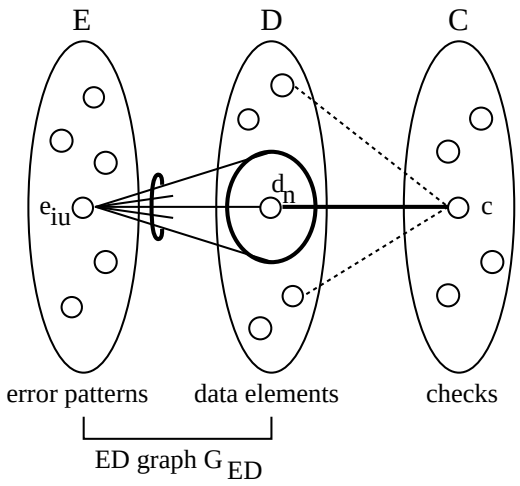


Figure 2.1: Checks for k-fault detectable ABFT system.

E_{ED} denotes the relation between error patterns and data elements : if a data element d_n is contained in an error pattern e_{iu} induced by a fault pattern f_i, then there exists an undirected edge(e_{iu}, d_n).

In the following, let $C = \{c_1, c_2, \cdots, c_q, \cdots, c_Q\}$ be a set of checks, where the check c_q is a subset of data elements.

Lemma 2.1 *An ABFT system is k-fault detectable if and only if for each error pattern* e_{iu}, $1 \leq i \leq |F^k|$, *there is at least one check* $c \in C$ *such that* $|c \cap e_{iu}| = 1$.

Proof: (1) *sufficient condition*: If $|c_q \cap e_{iu}|$=1, then the check c_q certainly outputs 1 for an error pattern e_{iu} because exactly one data element in c_q is in error. Hence, there is at least one check which certainly outputs 1 for each error pattern induced by f_i, $1 \leq i \leq |F^k|$. (2) *necessary condition*: The proof is by contradiction. Suppose that, for a certain error pattern e_{iu}, $^{\forall}c_q$, $[|c_q \cap e_{iu}| = 0$ or $|c_q \cap e_{iu}| \geq 2]$. If $|c_q \cap e_{iu}|$=0, then c_q certainly outputs 0 for the error pattern e_{iu}. If $|c_q \cap e_{iu}| \geq 2$, then the check c_q is unpredictable because the check c_q is $(g, 1)$ check. Hence, there is no check that can certainly output 1 for e_{iu}. This is a contradiction. □

Now, we will discuss k-fault locatable system. Analyzing ABFT system for its fault locatability is a much harder problem when compared to the problem of analyzing the fault detectability. This is the reason that, in the case of fault locatability, we have to determine not only whether a fault pattern is detectable but also whether the fault pattern is distinguishable from other fault patterns.

Lemma 2.2 *If an ABFT system is k-fault locatable, then for any pair of error patterns* e_{iu} *and* e_{jv}, $i \neq j$, $1 \leq i \leq |F^k|$, $1 \leq j \leq |F^k|$, $e_{iu} \oplus e_{jv} \neq \emptyset$, *where* $\oplus$ *denotes the symmetric difference.*

Proof: The proof is by contradiction. Suppose that for $i \neq j$, the symmetric difference of the error patterns e_{iu} and e_{jv} is empty. When f_i and f_j induce e_{iu} and e_{jv}, respectively, the syndrome for f_i and the one for f_j are the same because $e_{iu} = e_{jv}$. Hence, there is no way that the checks can distinguish f_i and f_j for $i \neq j$. This is a contradiction. □

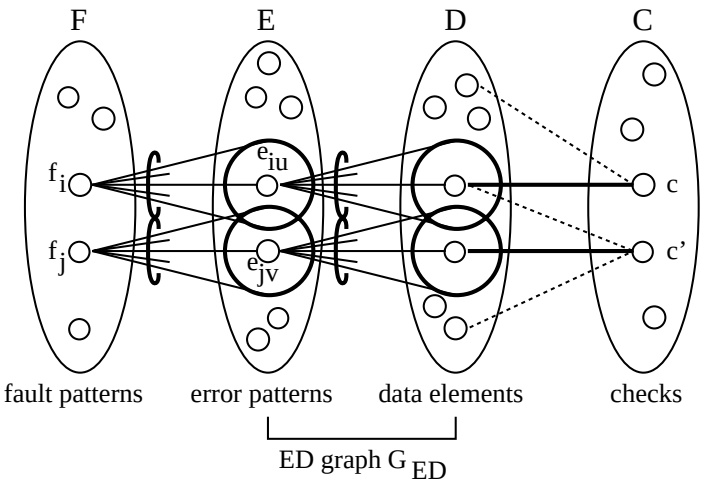


Figure 2.2: Checks for k-fault locatable ABFT system.

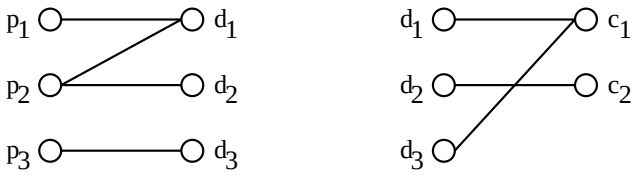
Theorem 2.1 *An ABFT system is k-fault locatable if and only if for each pair of error patterns e_{iu} and e_{jv}, $i \neq j$, $1 \leq i \leq |F^k|$, $1 \leq j \leq |F^k|$, there is at least one check-pair $c \in C$ and $c' \in C$ such that*

(1) $|c \cap (e_{iu} \oplus e_{jv})| = 1$ *and* $|c \cap (e_{iu} \cap e_{jv})| = 0$
(2) $|c' \cap e_{jv}| = 1$ *if* $|c \cap (e_{iu} - e_{jv})| = 1$
(3) $|c' \cap e_{iu}| = 1$ *if* $|c \cap (e_{jv} - e_{iu})| = 1$.

Proof: (1) *sufficient condition*: From the definition of k-fault locatability, the syndromes for two error patterns e_{iu} and e_{jv} which are induced by two-distinct fault patterns f_i and f_j, respectively, have to be different. Let d_{ij} be a data element contained in both $e_{iu} \oplus e_{jv}$ and c so that $|c \cap (e_{iu} \oplus e_{jv})|$=1 and $|c \cap (e_{iu} \cap e_{jv})|$=0. If d_{ij} is in e_{jv}, then $|c' \cap e_{iu}| = 1$ and the partial syndrome cc' is 01 and 11(or 1X) for f_i and f_j, respectively, where X denotes that the check is unpredictable. Similarly, if d_{ij} is in e_{iu}, then $|c' \cap e_{jv}|$=1 and the partial syndrome cc' is 11(or 1X) and 01 for f_i and f_j, respectively. Therefore, f_i and f_j are detected and distinguished by the check-pair c and c'. (2) *necessary condition*: The proof is by contradiction. Suppose that, for a certain pair of error patterns e_{iu} and e_{jv}, (a) $^{\forall}c_q$, $[|c_q \cap (e_{iu} \oplus e_{jv})| \neq 1]$, (b) $^{\exists}c_q$, $[|c_q \cap (e_{iu} \oplus e_{jv})| = 1$, $|c_q \cap (e_{iu} \cap e_{jv})| = 0]$ and $^{\forall}c'_q$, $[|c'_q \cap e_{iu}| \neq 1$(or $|c'_q \cap e_{jv}| \neq 1)]$. For the case of (a), there is no check that can give a different output because c_q has the same output for e_{iu} and e_{jv}, or that can certainly output 1 because c_q is either 0 or unpredictable for e_{iu}(or e_{jv}). Also, for the case of (b), there is no check that can certainly output 1 for e_{iu}(or e_{jv}) because c'_q is either 0 or unpredictable for e_{iu}(or e_{jv}). This is a contradiction. □

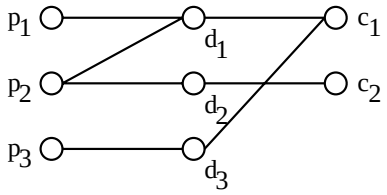
Corollary 2.1 *If an ABFT system is k-fault locatable due to Theorem 2.1, then it is also $2k$-fault detectable.*

Proof: It is clear that if an ABFT system is k-fault locatable, then it is also k-fault detectable because from the proof of Theorem 2.1, the check-pair c and c' can detect e_{iu} and e_{jv}. The error patterns induced by $2k$-fault are obtained by taking all available pairwise unions of elements in E^k and elements in E_k. Hence we will prove whether the



(a) PD graph.

(b) DC graph.



(c) PDC graph.

Figure 2.3: An example of conventional PDC graph model.

checks detect the error pattern $e_{iu} \cup e_{jv}$, $1 \leq i \leq |F^k|$, $1 \leq j \leq |F^k|$. Since exactly one data element in $e_{iu} \cup e_{jv}$ in E^{2k} is in data set of the check c, the error pattern $e_{iu} \cup e_{jv}$ can be detected by c from Lemma 2.1. Therefore, the checks which consist of these check-pairs can detect $2k$-fault. □

2.3 Conventional PDC Graph Model

In the graph-theoretic model proposed earlier in [5], an ABFT system system is represented by an undirected tripartite graph called the PDC graph whose vertex set is $P \cup D \cup C$ and its edge set is $PD \cup DC$, where P, D and C are sets of processors, data and checks, respectively, and PD and DC are sets of edges between P and D and between D and C, respectively. An edge $(u, v) \in PD$ implies that processor u affects the value of data element v in the computation: if processor u fails, v could have an error. An edge $(v, w) \in DC$ implies that data element v is checked by a check w.

A simple example for the PDC graph model is illustrated in Fig. 2.3. The processor p_1 affects the data element d_1 in the computation: d_1 is erroneous if p_1 fails. Similarly, p_2 affects d_1 and d_2: d_1 or/and d_2 are erroneous if p_2 fails, and also p_3 affects d_3: d_3 is erroneous if p_3 fails. The error pattern induced by a faulty processor is given as one of all available unions for data elements which are affected by the processor. According to notations defined in Section 2.2, the set of fault patterns F^1 for single-fault($k = 1$) is given as $\{\{p_1\}, \{p_2\}, \{p_3\}\}(= \{f_1^1, f_1^2, f_1^3\} = F_1)$. And the set of error patterns E^1 induced by fault patterns in F^1 is given as $\{\{d_1\}, \{d_2\}, \{d_1, d_2\}, \{d_3\}\}(= \{e_1^{11}(= e_1^{21}), e_1^{22}, e_1^{23}, e_1^{31}\} = e_1^1 \cup e_1^2 \cup e_1^3 = E_1)$. Note that the error pattern e_1^{11} induced by the fault pattern $\{p_1\}$ is the same to one(e_2^{11}) of error patterns induced by the fault pattern $\{p_2\}$. Since it is against the fault locatability of Lemma 2.2, there is no way that the fault patterns $\{p_1\}$ and $\{p_2\}$ can be distinguished by some checks.

Suppose that we want to design DC relations(Fig. 2.3(b)) for detecting single-fault from the PD relations(Fig. 2.3(a)). One possible solution is the use of two checks $c_1 =$

$\{d_1, d_3\}$ and $c_2 = \{d_2\}$: the check c_1 detects all error patterns including either d_1 or d_3, similarly c_2 detects all error patterns including d_2. Finally, the PDC graph for this configuration is shown in Fig. 2.3(c).

The importance of the graph-theoretic model is that the fault detection and location properties of the computation can be derived directly as a property of the graph. Hence the PDC graph model has been used by several researchers in analyzing and designing ABFT systems in practice. On the other hand, the PDC graph can be divided into two bipartite graphs: PD graph and DC graph which represent PD relationships and DC relationships, respectively. Designing PD relationships can be said to be a synthesis for ABFT system, while designing DC relationship can be called a design for ABFT system. Also, demonstrating the ability of the desired fault tolerant system when both PD graph and DC graph are given, can be said to be an analysis for ABFT system. In the PDC graph model, when a processor is faulty, the error pattern induced by the faulty processor is given as one of all available unions for data elements which are produced by the processor. However, if we introduce the appropriate error models which reflect the characteristics of error propagation between computation results, the number of error patterns induced by the faulty processor can be reduced.

Chapter 3

MPD Graph Model

3.1 Motivations

The first attempt analyzing ABFT systems was made by Banerjee and Abraham [2] who proposed a graph-theoretic model. Also, the matrix-based model presented in [4] simplified the analysis procedure by introducing the new necessary and sufficient conditions for the fault detectability and locatability of ABFT systems. These models use a PD graph to represent a given multiprocessor system. The PD graph provides information about which processors affect data elements, but it does not provide any information about how processors affect data elements. Specially, the PD graph excludes the dependent information between data elements. However the data dependency provides an important information about how the error for a computation result of a processor propagates to computation results of other processors. In general, error patterns consist of all available unions of sets of data elements reachable from each data element computed by a faulty processor. In the PD graph, each data element reachable from data element computed by a faulty processor becomes an error pattern, but it is not in practical applications which have some dependencies between computation results. Therefore, there may exist some redundant error patterns in analyzing and designing ABFT systems.

Such redundant error patterns may be a cause of an increase in the complexity and the number of checks in analyzing and designing ABFT systems, respectively. This situation motivates the investigation of more efficient model so that the analysis procedure is simplified and the number of checks is reduced. To achieve such objectives, we introduce data dependent information between computation results computed by processors. Since data dependency gives an useful information for error occurrence/propagation, we can simplify the analysis procedure and reduce the number of checks by using data dependent information in analyzing and designing ABFT systems.

The rest of this chapter is organized as follows. In Section 3.2, we will discuss two error models to be employed in this research. An analysis model based on MPD graph is presented in Section 3.3, and its effectiveness in analyzing and designing ABFT systems is described in Section 3.4. And a method to construct error patterns for each of SID model and MID model is shown in Section 3.5. Finally, Section 3.6 is used for conclusion.

3.2 Error Occurrence/Propagation Models

In most cases, a *hardware fault* will be identified by the observation of *data error*. Differing from the off-line tests for verifying chip/system functionality and performance, it is not possible, in the case of on-line test, to apply specific test vectors with regarding *controllability* and *observability* of a target error, and in turn for them, we should introduce appropriate model for error occurrence and error propagation. Considering the linear algebra based computations such as matrix operations and signal processing as our typical applications of ABFT systems, we employ the following two models.

Single Input-Driven(SID) Model An erroneous input to a computation will always result in erroneous computation result regardless of the other inputs nor the status of the processor. That is,

$\mathcal{E}$1-1. If the number of erroneous inputs used for a computation is *more than zero*, then the computation result is *erroneous* regardless of the status of the processor computing the result.

$\mathcal{E}$1-2. If the number of erroneous inputs used for a computation is *zero*, then the computation result depends on the status of the processor computing the result as follows:

- $\mathcal{E}$1-2a. If the processor is normal(fault-free), then the result is *correct*.
- $\mathcal{E}$1-2b. If the processor is faulty, then the result is either *erroneous* or *correct*.

In this case, once we construct a set of data elements reachable from each data element, each one of error patterns can be generated by simple union operation to properly selected sets of reachable data elements, and we can drastically reduce the number of error patterns to be considered compared to the conventional PDC graph model.

Multiple Inputs-Driven(MID) Model This model is more sophisticated with taking into account of some possibilities in the practical situations: a computation result with multiple erroneous inputs may possibly be error-free, and also a faulty processor may possibly generate error-free computation result when some of inputs are erroneous. That is,

$\mathcal{E}$2-1. If a processor is faulty, then each of its computation results is either *erroneous* or *correct* regardless of inputs used for its computation.

$\mathcal{E}$2-2. If a processor is normal(fault-free), then each of its computation results depends on the number of erroneous inputs used for its computation as follows:

- $\mathcal{E}$2-2a. If the number of erroneous inputs is *zero*, then the result is *correct*.
- $\mathcal{E}$2-2b. If the number of erroneous inputs is *one*, then the result is *erroneous*.
- $\mathcal{E}$2-2c. If the number of erroneous inputs is *more than one*, then the result is *undetermined*.

As a result, the set of error patterns to be considered with MID model becomes a superset of the one with SID model, but still a subset of the one from the conventional PD graph model because the error patterns in PD graph model consist of all available unions of data elements neighboring a faulty processor. In general, the computational complexity and the number of checks in analyzing and designing ABFT systems increase

as the number of error patterns increases. While the complexity and the number of checks tend to increase compared to the MPD graph with SID model in compensation for improving the accuracy of error propagation model, the effectiveness of MPD graph model with MID model holds good compared to the conventional PDC graph model.

3.3 MPD Graph

An algorithm which can be represented by a DG $G_D(V_D,E_D)$, is considered, where V_D denotes the set of vertices each of which represents an operation associated with data element(the result of the operation), primary input or primary output, and E_D denotes the set of directed edges each of which represents the data dependency from source to destination vertices. An example of DG is illustrated in Fig. 3.1(a). The marked nodes in left side are primary inputs, and the marked nodes in right side are primary outputs. Also, the unmarked nodes in middle side represent operations.

In a practical implementation, nodes in DG are mapped onto a set of processors, and each data element is classified into either internal or external data element, where the former is used only within a processor while the latter is used for computations in other processors or a primary output. In Fig. 3.1(a), the nodes in each dashed circle on operation nodes are mapped onto a processor. Here the operation nodes o_1, o_2, o_3 and o_7 are mapped to the processor p_1. Similarly o_4 and o_5 are mapped to p_2, and o_6 is mapped to p_3. Throughout this research, we assume that only external data elements can be checked by checks, and a given algorithm is executed on M processors and generates N external data elements.

We introduce a directed graph $G(V, E)$ called MPD graph which is defined as follows. V is the set of M processor nodes and N external data nodes. E is the set of processor-data relation edges and data dependency edges. The processor-data relation edge (p_m, d_n) is in E if the processor p_m produces the external data element d_n. On the other hand, the data dependency edge (d_i, d_j) is in E if the external data element d_i is sent to the processor which generates d_j and is used for computing d_j. An example of a MPD graph is shown in Fig. 3.1(b). Note that MPD graph allows multiple edges, and (d_i, d_j) is multiplied if the internal dependency graph peculiar to the processor which generates d_j has multiple paths from entry node(s) of d_i to the operation node associated with d_j(Fig. 3.1(c)(d)).

In this research, a MPD graph is assumed to be acyclic, that is, we will limit our typical algorithms to linear algebra based computations without feedback loops which include matrix operations and FIR type signal processing.

Unlike the conventional PD graph model, the error pattern on MPD graph model is obtained by the full utilization of data dependency. For both SID model and MID model, when a processor p_m is faulty, at least one of data elements which are adjacent to p_m is erroneous(from $\mathcal{E}$1-2b, $\mathcal{E}$2-1 and disregarding unobservable fault). Data elements which are reachable from p_m through such erroneous data element(s) are either erroneous or error-free($\mathcal{E}$2-2b, $\mathcal{E}$2-2c). Especially, data elements which are reachable from p_m along only one path containing erroneous data element adjacent to p_m are always erroneous($\mathcal{E}$1-1, $\mathcal{E}$2-2b). For example, when the processor p_1 of Fig. 3.1(b) is faulty, the error pattern on SID model is one of $\{d_1, d_4, d_5\}$, $\{d_2, d_3\}$, $\{d_5\}$, $\{d_1, d_2, d_3, d_4, d_5\}$ and $\{d_2, d_3, d_5\}$, while the error pattern on MID model is one of $\{d_1, d_4, d_5\}$, $\{d_2, d_3\}$, $\{d_5\}$, $\{d_1, d_2, d_3, d_4, d_5\}$,

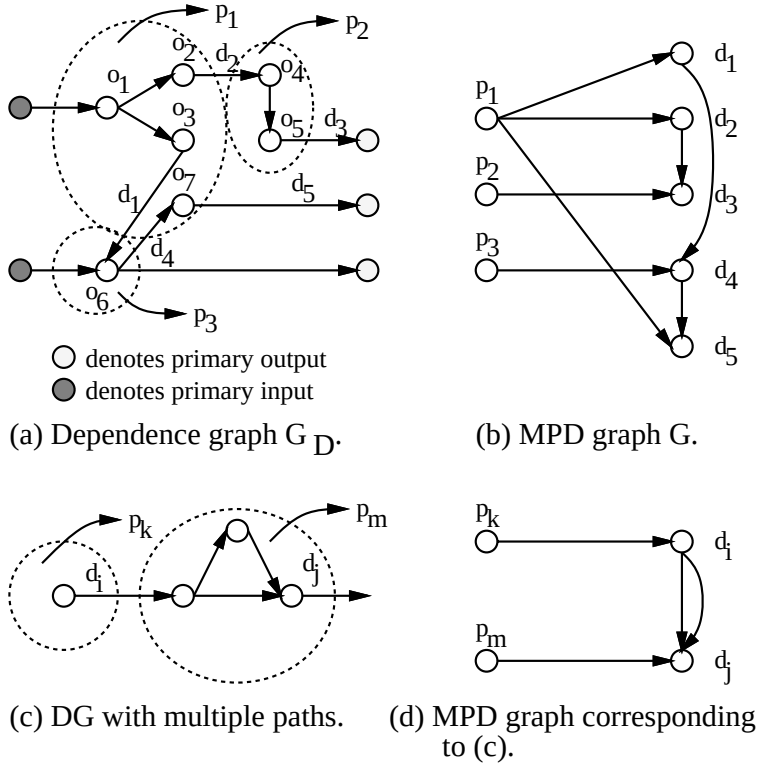


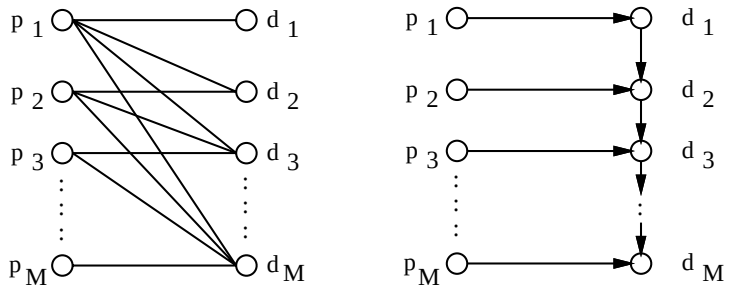
Figure 3.1: MPD graph for a DG.

$\{d_2, d_3, d_5\}$, $\{d_1, d_4\}$ and $\{d_1, d_2, d_3, d_4\}$. Note that the last two error patterns $\{d_1, d_4\}$ and $\{d_1, d_2, d_3, d_4\}$ are newly added in MID model because d_5 is error-free if the error term produced in computing d_1 is cancelled in computing d_5 when the computations for d_1 and d_5 are simultaneously faulty.

3.4 Effectiveness

To show the effectiveness of MPD model, we will consider the following algorithm as an example. Data element d_i is computed by operation o_i for $i = 1, 2, \cdots, M$. And data element d_i is used as an input for computing the data element d_{i+1} for $i = 1, 2, \cdots, M-1$ and d_M is not used as inputs for computing any other data elements. We assume that operations are mapped into processors in *one-to-one* fashion so that data element d_i is computed by processor p_i. The conventional PD graph and the proposed MPD graph for this situation are illustrated in Fig. 3.2.

Assume that we want to construct a set of checks such that the designated system is *single-fault detectable*. For the case of the conventional PD graph, M checks are needed to detect *single-fault* because a fault in p_1 may affect all of data elements to be erroneous, so all available error patterns induced by faulty processor p_1 are all of the available combinations of data elements $d_1, d_2, \cdots, d_M$. However, for the case of the proposed MPD graph which introduces data dependent information, we can detect single-fault to just one check for d_M because error pattern induced by faulty processor p_i always includes



(a) Conventional PD graph. (b) MPD graph.

Figure 3.2: PD graph and MPD graph to be considered as an example.

the data element d_M. Also there does not exist any check to locate single-fault in the conventional PD graph because the faulty processor p_1 and another faulty processors can not be identified by the *syndrome*. But single-fault in MPD graph can be located by M checks: one check for each data element d_i. Thus, there always exist checks to locate single-fault in the proposed MPD graph. In most cases that there exist data dependencies between data elements computed by processors, the designated ABFT system in the proposed MPD graph can be implemented with fewer checks than the conventional PD graph.

3.5 Construction of Error Patterns

In the following, we will describe a method for constructing error patterns from MPD graph for SID model and MID model. For a given MPD graph $G(V, E)$ with M processor nodes and N external data nodes, $Adj(p_m)$ and $D(p_m)$ denote the set of data elements adjacent to the processor p_m and the one reachable from the processor p_m, respectively. Let $D_m = \{D_{m1}, D_{m2}, \cdots, D_{m|Adj(p_m)|}\}$, $m = 1, 2, \cdots, M$, be a subset family of $D(p_m)$. Note that $D(p_m)$ is equivalent to the largest error pattern induced by single-fault pattern $\{p_m\}$. D_{ml} denotes the set of all data elements which are reachable from the l-th adjacent data element of the processor p_m. $|Adj(p_m)|$ is the number of adjacent data elements of p_m.

3.5.1 SID Model

First we construct E_1 from D_m, $1 \leq m \leq M$. A method for constructing E_1 is described in Algorithm 3.1, where $\uplus$ denotes *pairwise unions* of all elements in two sets. On the other hand, $E_2, E_3, \cdots, E_k$ can be obtained from E_1, recursively. That is, E_i consists of all of available pairwise unions of all elements in E_1 and those in E_{i-1}.

Algorithm 3.1

FIND(E_1)
input : MPD graph (V, E)
construct D_{ml}, $1 \leq m \leq M$, $1 \leq l \leq |Adj(p_m)|$

$E_1 \leftarrow \emptyset$
$for\ 1 \leq m \leq M$
{
  $E_{1m} \leftarrow \emptyset$
  $count \leftarrow |Adj(p_m)|$
  $while(count \neq 0)$
  {
    $E_{1m} \leftarrow D_m \uplus E_{1m}$
    $count \leftarrow count - 1$
  }
  $E_1 \leftarrow E_1 \cup E_{1m}$
}

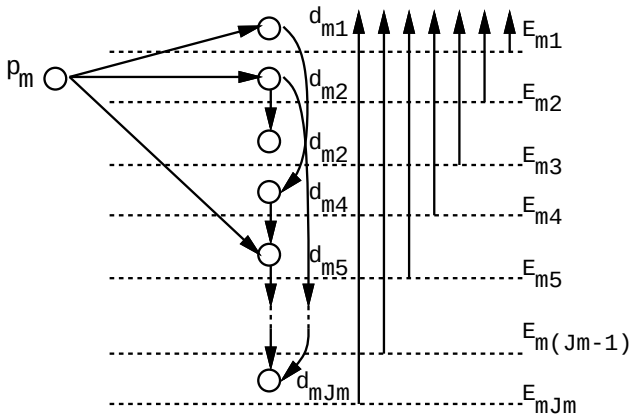
3.5.2 MID Model

Let L_{mi} be a set of such data elements that the number of edges on the longest path(s) from the processor p_m to these elements in D(p_m) is i. Let $d_{m1}d_{m2}\cdots d_{m|D(p_m)|}$ be a sequence of the elements in $D(p_m)$ according to the order of L_{mi}. That is, elements in L_{m1} are the first, which are followed by elements in L_{m2}, and so on.

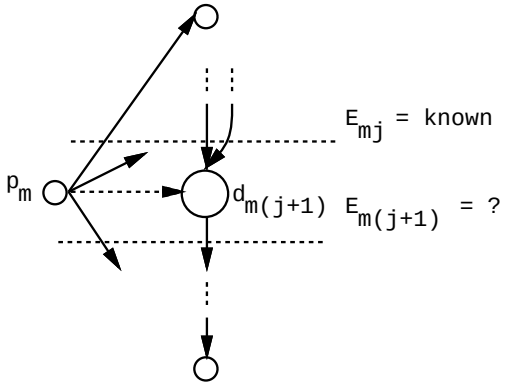
We will consider to construct error patterns recursively. Let $E_{mj} = \{e_{j1}, e_{j2}, \cdots, e_{jl}, \cdots, e_{jL_j}\}$ be the set of error patterns which consist of $d_{m1}, d_{m2}, \cdots, d_{mj}$. Let $I_{m(j+1)}$ be the set of input data elements for computing $d_{m(j+1)}$. Then the set of error patterns $E_{m(j+1)}$ can be obtained by the procedure shown in Algorithm 3.2, which is recursively repeated until $E_{m|D(p_m)|}$ is generated.

Algorithm 3.2

FIND($E_{m(j+1)}$)
$input: E_{mj}, I_{m(j+1)}, D(p_m)$
$E_{m(j+1)} \leftarrow \emptyset$
$for\ each\ e_{jl}\ in\ E_{mj}$
{
  $if\ d_{m(j+1)} \in Adj(p_m)$
  {
    $E_{m(j+1)} \leftarrow E_{m(j+1)} \cup \{e_{jl} \cup \{d_{m(j+1)}\}\}$
    $E_{m(j+1)} \leftarrow E_{m(j+1)} \cup \{e_{jl}\}$
  }
  $else$
  {
    $if\ |I_{m(j+1)} \cap e_{jl}| \geq 1$
    {
      $E_{m(j+1)} \leftarrow E_{m(j+1)} \cup \{e_{jl} \cup \{d_{m(j+1)}\}\}$
      $if\ |I_{m(j+1)} \cap e_{jl}| \geq 2,\ E_{m(j+1)} \leftarrow E_{m(j+1)} \cup \{e_{jl}\}$
    }
    $else\ E_{m(j+1)} \leftarrow E_{m(j+1)} \cup \{e_{jl}\}$
  }
}



(a) Illustration of the bevavior for error patterns.



(b) Construction of error pattern $E_{m(j+1)}$.

Figure 3.3: Behavior of error patterns for MID model.

As a result, $E_{m|D(p_m)|}$ is the set of all error patterns induced by the fault pattern $\{p_m\}$. When we construct error patterns for multiple fault, a similar procedure can be applied. That is, to generate error patterns for a fault pattern $\{p_{m_1}, p_{m_2}, \cdots, p_{m_k}\}$, nodes $p_{m_1}, p_{m_2}, \cdots, p_{m_k}$ in MPD graph are merged together, and $D(p_{m_1}) \cup D(p_{m_2}) \cup \cdots \cup D(p_{m_k})$ is considered instead of $D(p_m)$. From the nature of the above error pattern construction procedure, generated error patterns for $\{p_{m_1}, p_{m_2}, \cdots, p_{m_k}\}$ contain error patterns for any subset fault pattern of $\{p_{m_1}, p_{m_2}, \cdots, p_{m_k}\}$. Then, to generate error patterns induced by k-fault, it is enough to apply the above procedure to all fault patterns each of which contains exactly k faulty processors.

Finally, the set of error patterns to be considered with MID model becomes a superset of the one with the SID model, but still a subset of the one from the conventional PDC graph models. In general, the computational complexity and the number of checks in analyzing and designing ABFT systems increase as the number of error patterns increases. While the complexity and the number of checks tend to increase compared to the MPD graph with SID model in compensation for improving the accuracy of error propagation

model, the effectiveness of MPD graph model with MID model holds good compared to the conventional PDC graph models.

3.6 Conclusion

In this chapter, we have introduced two error occurrence/propagation models: SID model and MID model, and MPD graph model for analyzing and designing ABFT systems on these error occurrence/propagation models. The proposed analysis model fully utilizes data dependent information in generating error patterns, and it can suppress redundant error patterns. Also, we discussed a method for constructing error patterns for each of two error occurrence/propagation models. As a result, the error patterns induced by k-fault can be recursively constructed. However, to generate error patterns is a costly task, and also the number of error patterns is still too large to maintain for analysis and design of ABFT systems. Therefore, we will discuss single-fault detectability and locatability in analyzing and designing ABFT systems on MPD graph without constructing error patterns.

Chapter 4

Checking Scheme for SID Model

4.1 Introduction

Many works have been done in designing and analyzing of ABFT systems. Banerjee and Abraham [5] proposed a graph-theoretic model to represent ABFT systems. They also presented construction methods for ABFT systems by using the graph-theoretic model. The model was used by several researchers on the design and analysis of ABFT systems. Nair, Abraham and Banerjee [4] proposed a matrix-based model which was derived from the graph-theoretic model.

In this chapter, we will discuss a checking scheme based on MPD graph model proposed in Chapter 3. The checking scheme to be considered is discussed on a simple error occurrence/propagation model: SID model. The checking scheme for single-fault detectability and locatability is defined on MPD graph so that checks can be directly obtained from the MPD graph without constructing error patterns. A basic algorithm for constructing checks in designing single-fault locatable/two-fault detectable(SFL/TFD) ABFT systems is provided, and a design example for SFL/TFD ABFT system is described to demonstrate the basic algorithm.

The rest of this chapter is organized as follows. A checking scheme for single-fault detectability and locatability under SID error model, is discussed in Section 4.2. In Section 4.3, a basic algorithm for constructing checks for SFL/TFD ABFT system and a design example are shown. Section 4.4 is used for conclusion.

4.2 Single-Fault Detection and Location

In Chapter 2, k-fault detectability and locatability are discussed in terms of error patterns. However, in general, to generate error patterns is a costly task, and also the number of error patterns is still too large to maintain for analysis and design of ABFT systems. In this section, we describe single-fault detectability and locatability on MPD graph.

Theorem 4.1 *An ABFT system is single-fault detectable if for each D_i which is the subset family of $D(p_i)$, $1 \leq i \leq M$, there is a set of checks $C_i = \{c_0, c_1, \cdots, c_s, \cdots, c_{S_i}\} \subseteq C$ such that C_i is recursively(until D_i^s is empty) defined as follows.*

(1) $D_i^0 = D_i$

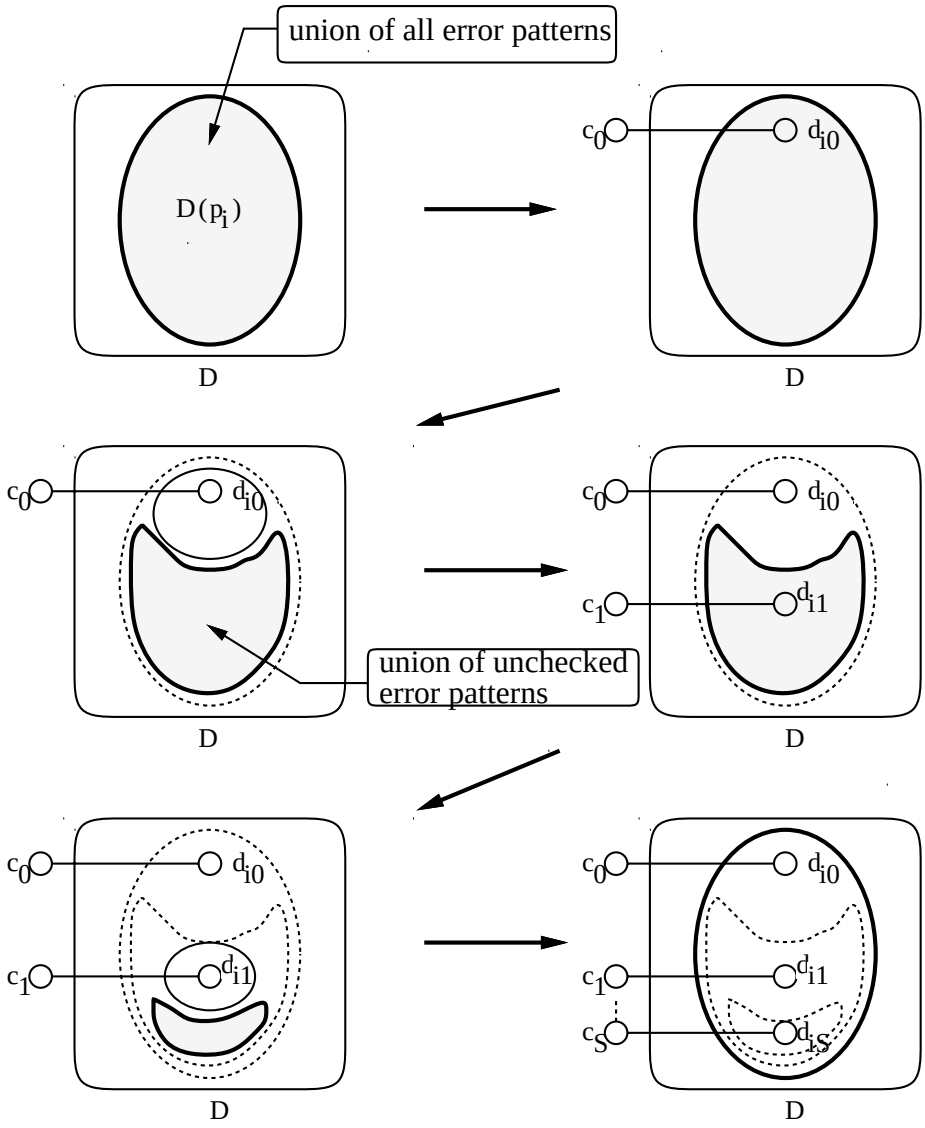


Figure 4.1: Construction of checks for single-fault detection.

(2) $\left| c_s \cap \left(\bigcup_{D_{iw} \in D_i^s} D_{iw} \right) \right| = 1$

(3) $D_i^{s+1} = D_i^s - \bigcup_{|c_s \cap D_{iw}|=1} \{D_{iw}\}$

Proof: Note that all of the available unions of elements in D_i become all error patterns for the single-fault pattern $\{p_i\}$. The check c_0 detects the largest error pattern $D(p_i)$ and some other error patterns which are all of the available unions of D_{iw}'s such that $|c_0 \cap D_{iw}| = 1$. The check c_s detects error patterns which are all of the available unions of D_{iw}'s such that $|c_s \cap D_{iw}| = 1$. Together with the definition of D_i^{s+1}, $D_i - D_i^{s+1} = \bigcup_{\alpha=1}^{s} \left[\bigcup_{|c_\alpha \cap D_{iw}|=1} \{D_{iw}\} \right]$

and $\{c_0, c_1, \cdots, c_s\}$ can detect error patterns which are all of the available unions of elements in $D_i - D_i^{s+1}$. Since c_{s+1} is selected so that $\left| c_s \cap \left(\bigcup_{D_{iw} \in D_i^{s+1}} D_{iw} \right) \right| = 1$, then $|D_i - D_i^{s+2}| \geq |D_i - D_i^{s+1}| + 1$, and hence $D_i - D_i^{s+1}$ becomes to contain all elements in D_i after appropriate iterations. □

Theorem 4.2 *An ABFT system is single-fault locatable if for each pair of D_i and D_j: the subset families of $D(p_i)$ and $D(p_j)$, respectively, $i \neq j$, $1 \leq i \leq M$, $1 \leq j \leq M$, there is a set of checks $C_{ij} = C_r \cup (\bigcup_{r=0}^{R} C_r') \subseteq C$, where $C_r = \{c_0, c_1, \cdots, c_r, \cdots, c_R\}$ and $C_r' = \{c_{r0}', c_{r1}', \cdots, c_{rb}', \cdots, c_{rB_r}'\}$ such that they are recursively(until either D_i^r or D_j^r is empty) defined as follows.*

(1) $D_i^0 = D_i$, $D_j^0 = D_j$

(2) $\left| c_r \cap \left(\bigcup_{D_{iw} \in D_i^r} D_{iw} \oplus \bigcup_{D_{jz} \in D_j^r} D_{jz} \right) \right| = 1$, $\left| c_r \cap \left(\bigcup_{D_{iw} \in D_i^r} D_{iw} \cap \bigcup_{D_{jz} \in D_j^r} D_{jz} \right) \right| = 0$

(3) *if* $\left| c_r \cap \left(\bigcup_{D_{iw} \in D_i^r} D_{iw} \right) \right| = 1$, *then*

$$\left[\begin{array}{l} D_j^{r0} = D_j^r \\ \left| c_{rb}' \cap \left(\bigcup_{D_{jz} \in D_j^{rb}} D_{jz} \right) \right| = 1 \\ D_j^{r(b+1)} = D_j^{rb} - \bigcup_{|c_{rb}' \cap D_{jz}|=1} \{D_{jz}\} \\ D_i^{r+1} = D_i^r - \bigcup_{|c_r \cap D_{iw}|=1} \{D_{iw}\} \\ D_j^{r+1} = D_j^r \end{array} \right]$$

(4) *if* $\left| c_r \cap \left(\bigcup_{D_{jz} \in D_j^r} D_{jz} \right) \right| = 1$, *then*

$$\left[\begin{array}{l} D_i^{r0} = D_i^r \\ \left| c_{rb}' \cap \left(\bigcup_{D_{iw} \in D_i^{rb}} D_{iw} \right) \right| = 1 \\ D_i^{r(b+1)} = D_i^{rb} - \bigcup_{|c_{rb}' \cap D_{iw}|=1} \{D_{iw}\} \\ D_i^{r+1} = D_i^r \\ D_j^{r+1} = D_j^r - \bigcup_{|c_r \cap D_{jz}|=1} \{D_{jz}\} \end{array} \right]$$

Proof: From Theorem 4.1, if $\left| c_r \cap \left(\bigcup_{D_{iw} \in D_i^r} D_{iw} \right) \right| = 1$, then c_r detects error patterns which are all of the available unions of D_{iw}'s such that $|c_r \cap D_{iw}|=1$ and C_r' detects all error patterns in $\bigcup_{D_{jz} \in D_j^r} D_{jz}$, while c_r outputs 0 for every error pattern in $\bigcup_{D_{jz} \in D_j^r} D_{jz}$. Accordingly, the partial syndrome $c_r c_{r0}' c_{r1}' \cdots c_{rB_r}'$ is different for any one of error patterns which are all of the available unions of D_{iw}'s such that $|c_r \cap D_{iw}|=1$ and any one

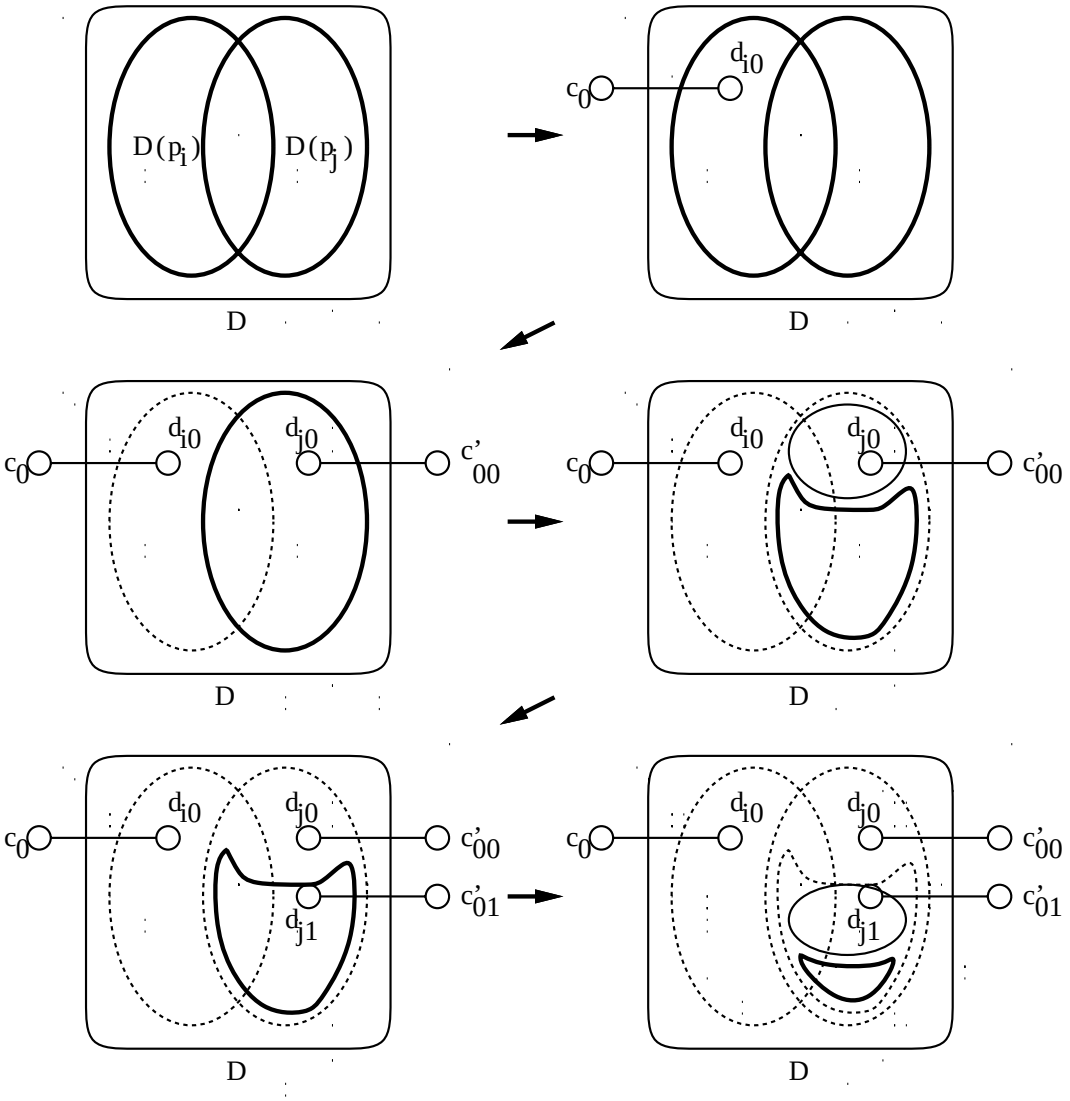


Figure 4.2: Construction(1) of checks for single-fault location.

of error patterns in $\bigcup_{D_{jz} \in D_j^r} D_{jz}$. Now the remained pairs of error patterns to be distinguished are all element pairs of $D_i^r - \bigcup_{|c_r \cap D_{iw}|=1} \{D_{iw}\} = D_i^{r+1}$ and $D_j^r = D_j^{r+1}$. The case $\left| c_r \cap \left(\bigcup_{D_{jz} \in D_j^r} D_{jz} \right) \right| = 1$ is the same but D_i^r and D_j^r are updated differently. Totally, the set of checks C_{ij} can distinguish fault patterns $\{p_i\}$ and $\{p_j\}$. Therefore, if there is a set of checks C_{ij} for $i \neq j$, $1 \leq i \leq M$, $1 \leq j \leq M$, then the ABFT system is single-fault locatable. □

Corollary 4.1 *If an ABFT system is single-fault locatable due to Theorem 4.2, then it is also two-fault detectable.*

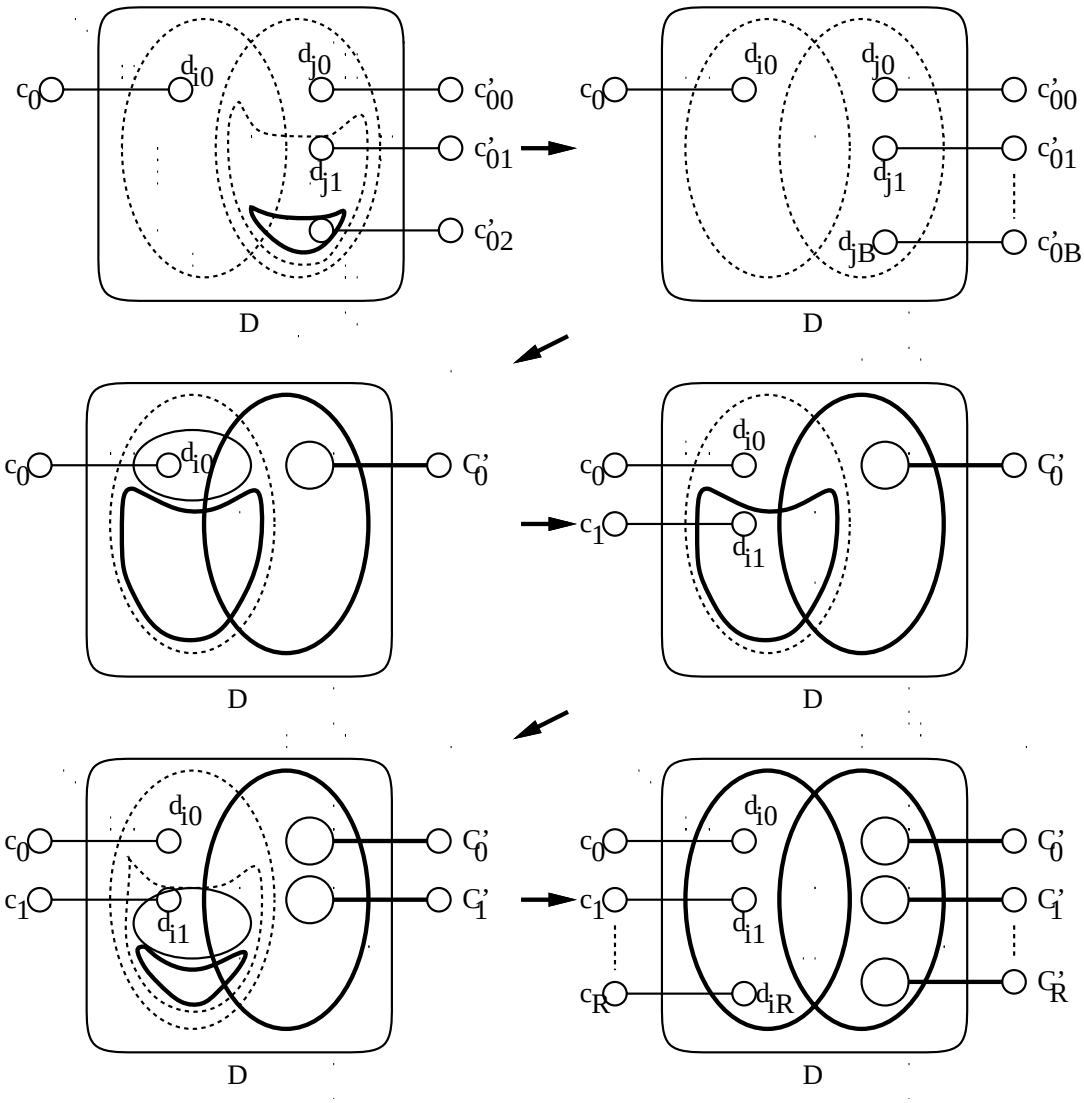


Figure 4.3: Construction(2) of checks for single-fault location.

Proof: Let U_{ix} be one of all available unions of elements in D_i and let U_{jy} be one of all available unions of elements in D_j. Then $U_{ij}(= U_{ix} \cup U_{jy})$ is one of the error patterns induced by the fault pattern $\{p_i, p_j\}$. From Theorem 4.2, since exactly one data element in U_{ij} is in the data set of c_r, U_{ij} can be detected by c_r. Since every error pattern for the fault pattern $\{p_i, p_j\}$ has the form $U_{ix} \cup U_{jy}$, the single-fault locatable ABFT system by Theorem 4.2 is also two-fault detectable system. □

4.3 Checks for SFL/TFD

4.3.1 A Basic Algorithm SFL-TFD I

Now, we introduce an algorithm to construct checks for single-fault locating and two-fault detecting ABFT system. Let c and c' be a check-pair satisfying Theorem 4.2: c and c' are

in C_r and C'_r, respectively. The algorithm SFL-TFD I shown in Algorithm 4.1 finds a set of checks C and a set of data elements $ptr(c)$ which have to be checked by check $c \in C$, and it always returns C and $ptr(c)$ if there exists a set of checks given by Theorem 4.2.

According to the algorithm SFL-TFD I, for a pair of Q_I and Q_J, there are four cases for constructing a check c satisfying Theorem 4.2:

(1) Case-I: exactly one data element of $ptr(c)$ for $c \in C$ such that $|c \cap (Q_I \cup Q_J)|$=1 is in $Q_I \oplus Q_J$.

(2) Case-II: exactly one data element of $ptr(c)$ for $c \in C$ such that $|c \cap (Q_I \cup Q_J)| \geq 2$ is in $Q_I \oplus Q_J$, and any data element of $ptr(c)$ is not in $Q_I \cap Q_J$.

(3) Case-III: any data element of $ptr(c)$ for $c \in C$ such that $|c \cap (Q_I \oplus Q_J)| \geq 1$ is not in $Q_I \cup Q_J$.

(4) Case-IV: for any check $c \in C$, any data element of c is not in $Q_I \oplus Q_J$, at least two data elements of $ptr(c)$ are in $Q_I \cup Q_J$, or at least one data element of $ptr(c)$ is in $Q_I \cap Q_J$.

Similarly, for Q, there are four cases for constructing a check c' satisfying Theorem 4.2:

(1) Case-A1(Case-B1): exactly one data element of $ptr(c')$ for the $c' \in C$ such that $|c' \cap Q|$=1 is in Q.

(2) Case-A2(Case-B2): exactly one data element of $ptr(c')$ for the $c' \in C$ such that $|c' \cap Q| \geq 2$ is in Q.

(3) Case-A3(Case-B3): any data element of $ptr(c')$ for the $c' \in C$ such that $|c' \cap Q| \geq 1$ is not in Q.

(4) Case-A4(Case-B4): for any check $c' \in C$, any data element of c' is not in Q, or at least two data elements of $ptr(c')$ are in Q.

On the other hand, the first "*while*" loop is executed until either Q_I or Q_J is empty set. In the next iteration, if c in Q_I, then Q_I is only updated. And if c in Q_J, then Q_J is only updated. Also, the second or third "*while*" loop is executed until Q is empty set.

Algorithm 4.1

SFL-TFD I
input : MPD graph (V, E)
construct D_{ml}, $1 \leq l \leq |Adj(p_m)|$, $1 \leq m \leq M$
$C \leftarrow \emptyset$
for $1 \leq i < j \leq M$
{
$T_I \leftarrow D_i$ and $T_J \leftarrow D_j$
$Q_I \leftarrow \bigcup_{w=1}^{|T_I|} D_{iw}$ and $Q_J \leftarrow \bigcup_{z=1}^{|T_J|} D_{jz}$
while($Q_I \neq \emptyset$ or $Q_J \neq \emptyset$)
{
(1) *find* $c \in C$ such that $|c \cap (Q_I \cup Q_J)| = 1$ and $|ptr(c) \cap (Q_I \oplus Q_J)| = 1$
if succeed(Case-I), *goto* (A)
(2) *find* $c \in C$ such that $|c \cap (Q_I \cup Q_J)| \geq 2$, $|ptr(c) \cap (Q_I \oplus Q_J)| = 1$

and $|ptr(c) \cap (Q_I \cap Q_J)| = 0$

if succeed(Case-II), $c \leftarrow c - ((Q_I \cup Q_J) - ptr(c))$ and *goto* (A)

(3) *find* $c \in C$ such that $|c \cap (Q_I \oplus Q_J)| \geq 1$ and $|ptr(c) \cap (Q_I \cup Q_J)| = 0$

if succeed(Case-III), *select* one element $d_n \in (c \cap Q_I \oplus Q_J)$,

$c \leftarrow c - ((Q_I \cup Q_J) - \{d_n\})$, $ptr(c) \leftarrow ptr(c) \cup \{d_n\}$ and *goto* (A)

(4) (1),(2) and (3) *fail*(Case-IV), *select* one element $d_n \in (Q_I \oplus Q_J)$,

$c \leftarrow D - ((Q_I \cup Q_J) - \{d_n\})$, $C \leftarrow C \cup \{c\}$ and $ptr(c) \leftarrow \{d_n\}$

(A) *if*$(|c \cap Q_I| = 1)$

{

$Q \leftarrow Q_J$ and $T \leftarrow T_J$

while$(Q \neq \emptyset)$

{

(A1) *find* $c' \in C$ such that $|c' \cap Q| = 1$ and $|ptr(c') \cap Q| = 1$

if succeed(Case-A1), *goto* (A0)

(A2) *find* $c' \in C$ such that $|c' \cap Q| \geq 2$ and $|ptr(c') \cap Q| = 1$

if succeed(Case-A2), $c' \leftarrow c' - (Q - ptr(c'))$ and *goto* (A0)

(A3) *find* $c' \in C$ such that $|c' \cap Q| \geq 1$ and $|ptr(c') \cap Q| = 0$

if succeed(Case-A3), *select* one element $d_n \in (c' \cap Q)$,

$c' \leftarrow c' - (Q - \{d_n\})$, $ptr(c') \leftarrow ptr(c') \cup \{d_n\}$ and *goto* (A0)

(A4) (A1),(A2) and (A3) *fail*(Case-A4), *sellect* one element $d_n \in Q$,

$c' \leftarrow D - (Q - \{d_n\})$, $C \leftarrow C \cup \{c'\}$ and $ptr(c') \leftarrow \{d_n\}$

(A0) $T \leftarrow T - \bigcup_{|c' \cap D_{jz}|=1} \{D_{jz}\}$ and $Q \leftarrow \bigcup_{D_{jz} \in T} D_{jz}$

}

$T_I \leftarrow T_I - \bigcup_{|c \cap D_{iw}|=1} \{D_{iw}\}$ and $Q_I \leftarrow \bigcup_{D_{iw} \in T_I} D_{iw}$

}

(B) *else*

{

$Q \leftarrow Q_I$ and $T \leftarrow T_I$

while$(Q \neq \emptyset)$

{

(B1) *find* $c' \in C$ such that $|c' \cap Q| = 1$ and $|ptr(c') \cap Q| = 1$

if succeed(Case-B1), *goto* (B0)

(B2) *find* $c' \in C$ such that $|c' \cap Q| \geq 2$ and $|ptr(c') \cap Q| = 1$

if succeed(Case-B2), $c' \leftarrow c' - (Q - ptr(c'))$ and *goto* (B0)

(B3) *find* $c' \in C$ such that $|c' \cap Q| \geq 1$ and $|ptr(c') \cap Q| = 0$

if succeed(Case-B3), *select* one element $d_n \in (c' \cap Q)$,

$c' \leftarrow c' - (Q - \{d_n\})$, $ptr(c') \leftarrow ptr(c') \cup \{d_n\}$ and *goto* (B0)

(B4) (B1),(B2) and (B3) *fail*(Case-B4), *sellect* one element $d_n \in Q$

$c' \leftarrow D - (Q - \{d_n\})$, $C \leftarrow C \cup \{c'\}$ and $ptr(c') \leftarrow \{d_n\}$

(B0) $T \leftarrow T - \bigcup_{|c' \cap D_{iw}|=1} \{D_{iw}\}$ and $Q \leftarrow \bigcup_{D_{iw} \in T} D_{iw}$

}

$$T_J \leftarrow T_J - \bigcup_{|c \cap D_{jz}|=1} \{D_{jz}\} \text{ and } Q_J \leftarrow \bigcup_{D_{jz} \in T_J} D_{jz}$$

}
}
}
return C and $\{ptr(c)|c \in C\}$

As we can see it, the algorithm SFL-TED I contains some indeterminacies in the selections of c(Case-I,II,III), c'(Case-A1,A2,A3 or Case-B1,B2,B3), and d_n(Case-IV, Case-A4 or Case-B4) among their plural candidates. Certain strategies for fixing these indeterminacies to minimize the number of checks are remained as a future problem.

Now, we will discuss the computational complexity of SFL-TFD I. We assume that union for two sets can be computed in $O(1)$ and the computations in "*while*" loop can be computed in $O(N)$. The analysis for complexity of SFL-TFD I is as follows.

1. "*construct*" all D_{ml}'s: $O(N^3)$
2. "*for*" loop: at most $\frac{M(M-1)}{2}$ iterations
3. computation of Q_I or Q_J: $O(N)$
4. each "*while*" loop: at most N iterations
5. computations in "*while*" loop: $O(N)$
6. computations of T, Q, $T_I(T_J)$ or $Q_I(Q_J)$: $O(N)$

Totally, the computation time T_c of SFL-TFD I is $O(M^2N^3)$.

4.3.2 A Design Example

We will consider a MPD graph illustrated in Fig. 4.4(a). From this MPD graph, $D_1 = \{\{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}\}, \{d_{12}\}\}$, $D_2 = \{\{d_{12}, d_{21}, d_{22}, d_{31}\}, \{d_{22}, d_{31}\}\}$, $D_3 = \{\{d_{31}\}, \{d_{32}\}, \{d_{33}, d_{41}, d_{51}, d_{52}\}\}$, $D_4 = \{\{d_{41}\}\}$ and $D_5 = \{\{d_{41}, d_{51}\}, \{d_{52}\}\}$. According to SFL-TFD I, we can construct checks for each pair of D_i and D_j, $1 \leq i < j \leq 5$. There are various solutions which depend on the method for choosing d_n for Case-IV(Case-A4 or Case-B4) and finding c(or c') in each "*find*" state within "*while*" loops. An example for constructing checks by SFL-TFD I is described as follows.

1. D_1 and D_j, $2 \leq j \leq 5$:
   $c_1 = \{\underline{d_{11}}\}$
   $c_2 = \{d_{11}, \underline{d_{21}}, \underline{d_{32}}\}$
   $c_3 = \{d_{11}, \underline{d_{12}}, d_{21}, \underline{d_{22}}, d_{32}, \underline{d_{33}}\}$
   $c_4 = \{d_{11}, d_{21}, \underline{d_{22}}, d_{32}, \underline{d_{33}}\}$
   $c_5 = \{d_{11}, d_{21}, d_{22}, \underline{d_{31}}, d_{32}\}$
   $c_6 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, d_{32}, d_{33}, \underline{d_{41}}, d_{51}, d_{52}\}$
   $c_7 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, d_{32}, d_{33}, d_{41}, d_{51}, \underline{d_{52}}\}$

2. D_2 and D_j, $3 \leq j \leq 5$:
   $c_4 = \{d_{11}, \underline{d_{22}}, \underline{d_{33}}\}$
   $c_6 = \{d_{11}, \underline{d_{41}}\}$

3. D_3 and D_j, $4 \leq j \leq 5$: no updated

4. D_4 and D_j, $5 \leq j \leq 5$:
   $c_7 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, d_{32}, d_{33}, \underline{d_{52}}\}$
   $c_8 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, d_{32}, d_{33}, \underline{d_{51}}, d_{52}\}$

Where, the underlined data elements of each check c_q, $1 \leq q \leq 8$, denote that such data elements are in $ptr(c_q)$. The result of SFL-TFD I by the above execution is illustrated in Fig. 4.4(b)(Result I). The thick line between data elements and checks denotes that the data element is in $ptr(c)$ for a check $c \in C$. On the other hand, if d_n is chosen with the maximum cardinality for $D_{d_n} = \{D_{ml} | d_n \in D_{ml}, 1 \leq m \leq M, 1 \leq l \leq |D_m|\}$, then the number of checks can be reduced in some cases. From the MPD graph of Fig. 4.4(a), $|D_{d_{11}}| = 1$, $|D_{d_{12}}| = 2$, $|D_{d_{21}}| = 2$, $|D_{d_{22}}| = 3$, $|D_{d_{31}}| = 4$, $|D_{d_{32}}| = 1$, $|D_{d_{33}}| = 1$, $|D_{d_{41}}| = 3$, $|D_{d_{51}}| = 2$, $|D_{d_{52}}| = 2$. A method using $|D_{d_n}|$ for constructing checks is represented in the following procedure.

1. D_1 and D_j, $2 \leq j \leq 5$:
   $c_1 = \{\underline{d_{11}}, \underline{d_{41}}\}$
   $c_2 = \{d_{11}, \underline{d_{31}}\}$
   $c_3 = \{\underline{d_{12}}\}$
   $c_4 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, \underline{d_{32}}, d_{33}, d_{41}, d_{51}, \underline{d_{52}}\}$

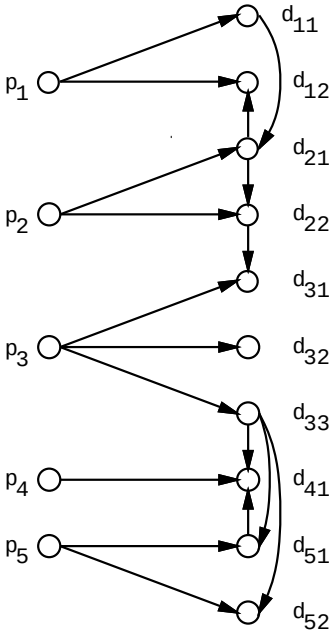
2. D_2 and D_j, $3 \leq j \leq 5$:
   $c_4 = \{d_{11}, d_{12}, d_{21}, \underline{d_{32}}, d_{33}, d_{41}, d_{51}, \underline{d_{52}}\}$
   $c_5 = \{d_{11}, d_{12}, d_{21}, \underline{d_{22}}, d_{32}, d_{33}, d_{41}, d_{51}, d_{52}\}$

3. D_3 and D_j, $4 \leq j \leq 5$:
   $c_5 = \{d_{11}, d_{12}, d_{21}, \underline{d_{22}}, \underline{d_{51}}\}$
   $c_6 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, \underline{d_{32}}\}$
   $c_7 = \{d_{11}, d_{12}, d_{21}, d_{22}, d_{31}, d_{32}, \underline{d_{33}}\}$

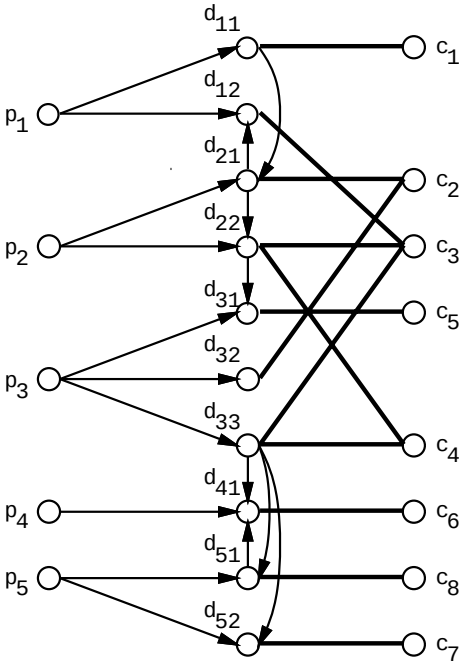
4. D_4 and D_j, $5 \leq j \leq 5$: no updated

The result of SFL-TFD I using $|D_{d_n}|$ is illustrated in Fig. 4.4(c)(Result II). Unfortunately, the number of checks for Result II can be reduced by merging checks. There are several merging candidates: (1) c_1 and c_6(or c_7), (2) c_3 and c_6(or c_7), and so on. The result merging c_6 into c_3 is illustrated in Fig. 4.4(d)(Result III).

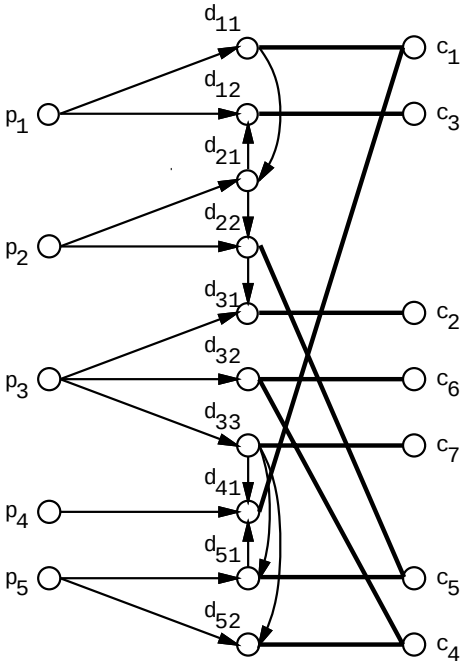
For Result I, II and III, the syndrome for each error pattern induced by single-fault patterns $\{p_1\}$, $\{p_2\}$, $\{p_3\}$, $\{p_4\}$, and $\{p_5\}$ is described in Fig. 4.5. Where X denotes that the check is unpredictable due to $(g,1)$ check($\mathcal{C}3$). Since the syndrome is different for any pair of fault patterns and at least one bit of the syndrome is 1 for any error pattern, the ABFT system is single-fault locatable and detectable. This ABFT system is also two-fault detectable from Corollary 4.1.



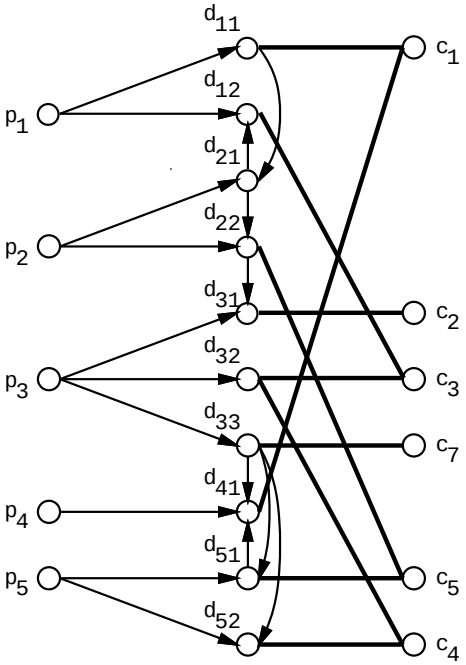
(a) A MPD graph G(V,E) .



(b) Result I for SFL-TFD algorithm.



(c) Result II for SFL-TFD algorithm.



(d) Result III merging c_6 and c_3 on Result II.

Figure 4.4: Examples for SFL/TFD ABFT system: SID.

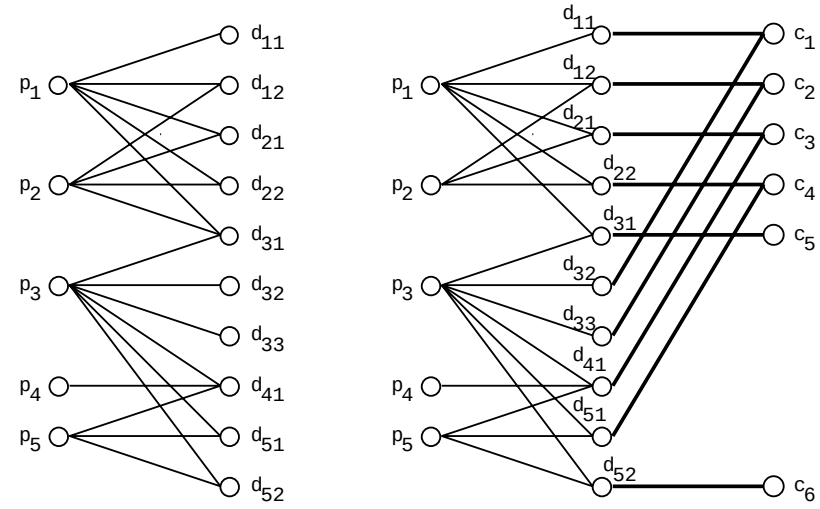
fault patterns	error patterns	syndrome I $c_1c_2c_3c_4c_5c_6c_7c_8$	syndrome II $c_1c_2c_3c_4c_5c_6c_7$	syndrome III $c_1c_2c_3c_4c_5c_6$
$\{p_1\}$	$\{d_{11}, d_{12}, d_{21}d_{22}, d_{31}\}$	1 1 X 1 1 0 0 0	1 1 1 0 1 0 0	1 1 1 0 1 0
	$\{d_{12}\}$	0 0 1 0 0 0 0 0	0 0 1 0 0 0 0	0 0 1 0 0 0
$\{p_2\}$	$\{d_{12}, d_{21}, d_{22}d_{31}\}$	0 1 X 1 1 0 0 0	0 1 1 0 1 0 0	0 1 1 0 1 0
	$\{d_{22}, d_{31}\}$	0 0 1 1 1 0 0 0	0 1 0 0 1 0 0	0 1 0 0 1 0
$\{p_3\}$	$\{d_{31}\}$	0 0 0 0 1 0 0 0	0 1 0 0 0 0 0	0 1 0 0 0 0
	$\{d_{32}\}$	0 1 0 0 0 0 0 0	0 0 0 1 0 1 0	0 0 1 1 0 0
	$\{d_{33}, d_{41}, d_{51}d_{52}\}$	0 0 1 1 0 1 1 1	1 0 0 1 1 1 1	1 0 0 1 1 1
	$\{d_{31}, d_{32}\}$	0 1 0 0 1 0 0 0	0 1 0 1 0 1 0	0 1 1 1 0 0
	$\{d_{31}, d_{33}, d_{41}d_{51}, d_{52}\}$	0 0 1 1 1 1 1 1	1 1 0 1 1 0 1	1 1 0 1 1 1
	$\{d_{32}, d_{33}, d_{41}d_{51}, d_{52}\}$	0 1 1 1 0 1 1 1	1 0 0 X 1 1 1	1 0 1 X 1 1
	$\{d_{31}, d_{32}, d_{33}d_{41}, d_{51}, d_{52}\}$	0 1 1 1 1 1 1 1	1 1 0 X 1 1 1	1 1 1 X 1 1
$\{p_4\}$	$\{d_{41}\}$	0 0 0 0 0 1 0 0	1 0 0 0 0 0 0	1 0 0 0 0 0
$\{p_5\}$	$\{d_{41}, d_{51}\}$	0 0 0 0 0 1 0 1	1 0 0 0 1 0 0	1 0 0 0 1 0
	$\{d_{52}\}$	0 0 0 0 0 0 1 0	0 0 0 1 0 0 0	0 0 0 1 0 0
	$\{d_{41}, d_{51}, d_{52}\}$	0 0 0 0 0 1 1 1	1 0 0 1 1 0 0	1 0 0 1 1 0

Figure 4.5: Syndromes for Result I, Result II and Result III: SID.

4.3.3 Comparison with Conventional PDC Graph Model

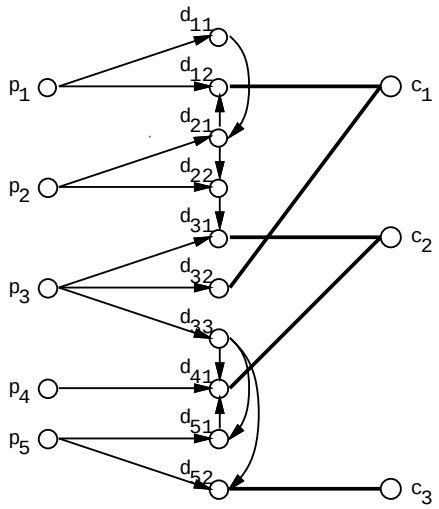
Consider MPD graph of Fig. 4.4(a) and assume that each external data element is primary output for a given algorithm. The corresponding conventional PD graph can be obtained by connecting each data element which is affected by a processor to the processor as shown in Fig. 4.6(a). In conventional PD graph, the set of error patterns induced by a faulty processor consists of all available combinations of data elements which are connected to the processor. Hence, there may be some redundant error patterns which do not appear in real applications. For example, the set of error patterns for the single-fault pattern $\{p_5\}$ is $\{\{d_{41}\}, \{d_{51}\}, \{d_{52}\}, \{d_{41}, d_{51}\}, \{d_{41}, d_{52}\}, \{d_{51}, d_{52}\}, \{d_{41}, d_{51}, d_{52}\}\}$ in the conventional PD graph and the one is $\{\{d_{41}, d_{51}\}, \{d_{52}\}, \{d_{41}, d_{51}, d_{52}\}\}$ in the MPD graph model under SID error model. There are four redundant error patterns $\{d_{41}\}$, $\{d_{51}\}$, $\{d_{41}, d_{52}\}$, and $\{d_{51}, d_{52}\}$. On the other hand, the error pattern $\{d_{41}\}$ is also induced by the single-fault patterns $\{p_3\}$ and $\{p_4\}$ in the conventional PD graph. Hence, there is no check which is distinguishable between $\{p_5\}$ and $\{p_3\}/\{p_4\}$.

Suppose that we want to construct a set of checks such that the designated system



(a) The conventional PD graph.

(b) A SFD ABFT system for (a).



(c) A SFD ABFT system for Fig.4.3(a): SID.

Figure 4.6: The comparisons in designing single-fault detectable ABFT system: SID.

is single-fault detectable. For the case of the conventional PD graph(Fig. 4.6(a)), at least *six* checks are required to detect single-fault(Fig. 4.6(b)). However, for the case of the MPD graph(Fig. 4.4(a)), we can detect single-fault by *three* checks(Fig. 4.6(c)). In fact, the set of error patterns obtained from the MPD graph model is a subset of the one obtained from the conventional PD graph model, and in most cases that there exist data dependencies between primary outputs after mapping DG for a given algorithm to processors, the designated ABFT system based on MPD graph model under SID error model can be implemented with fewer checks than the conventional PDC graph model.

On the other hand, there are various solutions depending on the mapping DG for a given algorithm to a set of processors. The problem how to map DG to processors is out of concern, and we assume that the mapping is given *a priori*. Also, there is no general way that can examine about how the number of checks affects the efficiency of ABFT system because its cost strongly depends on the complexity of an implementation

of checks. However, in general, there is a possibility that for $(g,1)$ check using checksum technique, which can be simply implemented, the desired ABFT system can be efficiently implemented by reducing the number of checks. The problem of how the number of checks affects the efficiency of the desired ABFT system is remained as a future work.

4.4 Conclusion

In this chapter, we proposed a checking scheme based on MPD graph model for single-fault detectability and locatability in analyzing and designing ABFT systems on a simple error occurrence/propagation model: SID Model. The checking scheme was defined on MPD graph so that checks can be directly obtained from the MPD graph without constructing error patterns. Also we gave a basic algorithm SFL-TFD I for constructing checks of SFL/TFD ABFT system, and demonstrated a design example for SFL/TFD ABFT system based on the basic algorithm. The algorithm contains some indeterminacies in the selections of c, c', and d_n among their plural candidates. Certain strategies for fixing these indeterminacies to minimize the number of checks are remained as a future problem. As a result, the desired ABFT system based on MPD graph under the SID error model was implemented with fewer checks compared to the conventional PDC graph model.

Chapter 5

Checking Scheme for MID Model

5.1 Introduction

An analysis model based on MPD graph under SID error model was discussed in Chapter 4, and it showed that the number of error patterns to be considered can be reduced by utilizing data dependency between computation results. However, the error occurrence/propagation model: SID model employed in Chapter 4 is so simple that an erroneous input to a computation will always result in erroneous computation result regardless of the other inputs nor the status of the processor. But, in the practical situations, a computation result with multiple erroneous inputs may possibly be error-free, and also a faulty processor may possibly generate error-free computation result when some of inputs are erroneous. Therefore, we take into account of these possibilities and introduce a sophisticated error occurrence/propagation model: MID model.

In this chapter, we will discuss a checking scheme based on MPD graph model for single-fault detectability and locatability with a sophisticated error occurrence/propagation model: MID model, and the checking scheme is defined on MPD graph so that checks can be directly obtained from the MPD graph without constructing error patterns. A basic algorithm for constructing checks in designing single-fault locatable/two-fault detectable(SFL/TFD) ABFT systems is provided, and a design example for SFL/TFD ABFT system is described to demonstrate the basic algorithm.

The rest of this chapter is organized as follows. A checking scheme for single-fault detectability and locatability under MID error model, is discussed in Section 5.2. In Section 5.3, a basic algorithm for constructing checks for SFL/TFD ABFT system and a design example are shown. Section 5.4 is used for conclusion.

5.2 Single-Fault Detection and Location

Now, each node of data elements in MPD graph is named as d_{mi} so that mi stands for i-th adjacent data element of a processor p_m. Following this notation, let D_{mi} be a set of data elements which are reachable from d_{mi}, and we call d_{mi} as a *source data* of D_{mi}. D_m is used for representing $\{D_{m1}, D_{m2}, \cdots, D_{m|Adj(p_m)|}\}$, where $Adj(p_m)$ denotes the set of data elements adjacent to the processor p_m. Furthermore, let $\tilde{T}_{mi}$ be the subset of D_{mi}, each element of which has two or more paths from d_{mi}.

Theorem 5.1 *An ABFT system with a set of checks C is single-fault detectable if for each m, $1 \leq m \leq M$, there is a sequence of checks $c_1 c_2 \cdots c_q \cdots c_{|Adj(p_m)|}$ and a permutation $\Pi_m = (I_1, I_2, \cdots, I_q, \cdots, I_{|Adj(p_m)|})$ of integers from 1 up to $|Adj(p_m)|$ such that $|c_q \cap (D_{mI_q} - \tilde{T}_{mI_q})| = 1$ and $|c_q \cap (\bigcup_{l=q+1}^{|Adj(p_m)|} D_{mI_l} \cup \tilde{T}_{mI_q})| = 0$, $1 \leq q \leq |Adj(p_m)|$.*

Proof: Note that any error pattern induced by the fault pattern $\{p_m\}$ contains at least one of $d_{m1}, d_{m2}, \cdots, d_{m|Adj(p_m)|}$. Now, we let $c_q \cap (D_{mI_q} - \tilde{T}_{mI_q})$ be $\{\tilde{d}_q\}$ and also let $d^{(q)}$ be the source data of D_{mI_q}. When p_m is faulty, there must be the first erroneous source data in the sequence of $d^{(1)} d^{(2)} \cdots d^{(|Adj(p_m)|)}$, and let it be $d^{(k)}$. Since $d^{(1)}, d^{(2)}, \cdots, d^{(k-1)}$ are all error-free, the error pattern contains only elements in $\bigcup_{l=k}^{|Adj(p_m)|} D_{mI_l}$. On the other hand, since $c_k \cap (\bigcup_{l=k+1}^{|Adj(p_m)|} D_{mI_l} \cup \tilde{T}_{mI_k}) = \emptyset$, the output of c_k is not affected by the status of any element in $\bigcup_{l=k+1}^{|Adj(p_m)|} D_{mI_l}$ nor any element in $\tilde{T}_{mI_k}$. Together with the fact that $|c_k \cap (D_{mI_k} - \tilde{T}_{mI_k})| = 1$, the check c_k certainly outputs 1. □

Theorem 5.2 *An ABFT system with a set of checks C is single-fault locatable if for each pair of i and j, $i \neq j$, $1 \leq i \leq M$, $1 \leq j \leq M$, there is a sequence of all elements in $D_i \cup D_j$, $D_{**}^{(1)} D_{**}^{(2)} \cdots D_{**}^{(q)} \cdots D_{**}^{(|Adj(p_i)|+|Adj(p_j)|)}$, where $**$ represents the subscript of each element in $D_i \cup D_j$, and a sequence of $|Adj(p_i)| + |Adj(p_j)|$ checks, $c_1 c_2 \cdots c_q \cdots c_{|Adj(p_i)|+|Adj(p_j)|}$, such that $|c_q \cap (D_{**}^{(q)} - \tilde{T}_{**}^{(q)})| = 1$ and $|c_q \cap (\bigcup_{l=q+1}^{|Adj(p_i)|+|Adj(p_j)|} D_{**}^{(l)} \cup \tilde{T}_{**}^{(q)})| = 0$, $1 \leq q \leq |Adj(p_i)| + |Adj(p_j)|$.*

Proof: Let $c_q \cap (D_{**}^{(q)} - \tilde{T}_{**}^{(q)})$ be $\{\tilde{d}_q\}$ and also let $d_{**}^{(q)}$ be the source data of $D_{**}^{(q)}$. Note again that any error pattern induced by the fault pattern $\{p_i\}(\{p_j\})$ contains at least one of source data of D_{ir}'s, $1 \leq r \leq |Adj(p_i)|(D_{jr}$'s, $1 \leq r \leq |Adj(p_j)|)$. When either p_i or p_j is faulty, we can find the first such erroneous source data in the sequence of $d_{**}^{(1)} d_{**}^{(2)} \cdots d_{**}^{(|Adj(p_i)|+|Adj(p_j)|)}$, and let it be $d_{**}^{(k)}$. Similar to the proof of Theorem 1, the check c_k certainly outputs 1 whether $d_{**}^{(k)}$ is in $Adj(p_i)$ or in $Adj(p_j)$. Moreover, $c_1, c_2, \cdots, c_{k-1}$ are all 0 because $c_q \cap (\bigcup_{l=q+1}^{|Adj(p_i)|+|Adj(p_j)|} D_{**}^{(l)}) = \emptyset$, $1 \leq q \leq k-1$ and error pattern contains only elements in $\bigcup_{l=k}^{|Adj(p_i)|+|Adj(p_j)|} D_{**}^{(l)}$. Since the source data of D_{i*}'s and those of D_{j*}'s are located at different positions in the sequence of $d_{**}^{(1)} d_{**}^{(2)} \cdots d_{**}^{(|Adj(p_i)|+|Adj(p_j)|)}$, the location of the first 1 in the syndrome $c_1 c_2 \cdots c_{|Adj(p_i)|+|Adj(p_j)|}$ is different for any pair of error patterns, one is induced by the fault pattern $\{p_i\}$ and the other is induced by $\{p_j\}$. □

Corollary 5.1 *If an ABFT system with a set of checks C is single-fault locatable due to Theorem 5.2, then it is also two-fault detectable.*

Proof: With respect to the fault pattern $\{p_i, p_j\}$, every data element adjacent to either p_i or p_j possibly becomes erroneous. Note that all error patterns induced by the fault pattern $\{p_i, p_j\}$ contains all error patterns induced by the fault pattern $\{p_i\}$ and those induced by $\{p_j\}$. If we modify the MPD graph so that the processor nodes p_i and p_j are merged into a single node p_*, the above situation is identical to the single-fault of the processor p_*. In this model, $D_* = D_i \cup D_j = \{D_{*1}, D_{*2}, \cdots, D_{*q}, \cdots, D_{*(|Adj(p_i)|+|Adj(p_j)|)}\}$. From Theorem 5.2, since there is a sequence of checks $c_1 c_2 \cdots c_q \cdots c_{|Adj(p_i)|+|Adj(p_j)|}$ and a permutation $\Pi_* = (1, 2, \cdots, q, \cdots, |Adj(p_i)| + |Adj(p_j)|)$ such that $|c_q \cap (D_{*q} - \tilde{T}_{*q})| = 1$ and $|c_q \cap (\bigcup_{l=q+1}^{|Adj(p_i)|+|Adj(p_j)|} D_{*l} \cup \tilde{T}_{*q})| = 0$, it satisfies Theorem 1 and so can detect fault patterns $\{p_i\}$, $\{p_j\}$ and $\{p_i, p_j\}$, $i \neq j$, $1 \leq i \leq M$, $1 \leq j \leq M$. □

In the next section, we will discuss the construction of checks for single-fault location/two-fault detection based on Theorem 5.2.

5.3 Checks for SFL/TFD

5.3.1 A Basic Algorithm SFL-TFD II

It is interesting to note that, for any acyclic MPD graph, we can always construct a set of trivial checks C_T such that each check in C_T contains only one data element and every data element is contained in one check; that is, $C_T = \{\{d_{11}\}, \{d_{12}\}, \cdots, \{d_{M|Adj(p_m)|}\}\}$.

Lemma 5.1 *For any acyclic MPD graph, the ABFT system with its set of trivial checks C_T is single-fault locatable.*

Proof: Since the MPD graph is acyclic, data elements in MPD graph can be topologically sorted. For any two processors, we can extract their adjacent data elements from this sorting result with preserving their order. As a result, the corresponding sequence of trivial checks satisfies the condition given in Theorem 5.2. □

Now, we introduce an algorithm to construct checks for SFL/TFD ABFT system. Let c be a check satisfying Theorem 5.2. Algorithm 5.1 finds a set of checks C and a set of data elements $ptr(c)$ which have to be checked by check $c \in C$, and it always returns C and $ptr(c)$ if there exits a set of checks given by Theorem 5.2.

According to the algorithm SFL-TFD II, for a pair of D_i and D_j, $1 \leq i < j \leq M$, the elements in $D_i \cup D_j$ are topologically sorted into $D^{(1)} \cdots D^{(l)} \cdots D^{(|Adj(p_i)|+|Adj(p_j)|)}$ so that $D^{(l)} - (\cup_{k=l+1}^{|Adj(p_i)|+|Adj(p_j)|} D^{(k)})$ is not empty. And then for $Q(= \cup_{k=l}^{|Adj(p_i)|+|Adj(p_j)|} D^{(k)})$, $1 \leq l \leq |Adj(p_i)| + |Adj(p_j)|$, there are four cases to construct a check c satisfying Theorem 5.2:

(1) Case-I: exactly one data element of $ptr(c)$ for $c \in C$ such $|c \cap Q| = 1$ is in $D^{(l)} - \tilde{T}^{(l)}$.

(2) Case-II: exactly one data element of $ptr(c)$ for $c \in C$ such that $|c \cap Q| \geq 2$ is in $D^{(l)} - \tilde{T}^{(l)}$, and any data element of $ptr(c)$ is not in $Q - (D^{(l)} - \tilde{T}^{(l)})$.

(3) Case-III: any data element of $ptr(c)$ for $c \in C$ such that $|c \cap (D^{(l)} - \tilde{T}^{(l)})| \geq 1$ is not in Q.

(4) Case-IV: for any check $c \in C$, any data element of c is not in Q, at least two data elements of $ptr(c)$ are in $D^{(l)} - \tilde{T}^{(l)}$, or at least one data element of $ptr(c)$ is in $Q - (D^{(l)} - \tilde{T}^{(l)})$.

Algorithm 5.1

SFL-TFD II
input : MPD graph (V, E)
construct D_{ml}, $1 \leq l \leq |Adj(p_m)|$, $1 \leq m \leq M$
construct $\tilde{T}_{ml}$, $1 \leq l \leq |Adj(p_m)|$, $1 \leq m \leq M$
$C \leftarrow \emptyset$
for $1 \leq i < j \leq M$
{
sort $D_{i1}, \cdots, D_{i|Adj(p_i)|}, D_{j1}, \cdots, D_{j|Adj(p_j)|}$ into $D^{(1)} D^{(2)} \cdots D^{(|Adj(p_i)|+|Adj(p_j)|)}$

so that $\left| D^{(l)} - \left(\bigcup_{k=l+1}^{|Adj(p_i)|+|Adj(p_j)|} D^{(k)} \right) \right| \geq 1$

for $l = 1$ *to* $|Adj(p_i)| + |Adj(p_j)|$

{

$Q \leftarrow \bigcup_{k=l}^{|Adj(p_i)|+|Adj(p_j)|} D^{(k)}$

(1) *find* $c \in C$ such that $|c \cap Q| = 1$ and $|ptr(c) \cap (D^{(l)} - \tilde{T}^{(l)})| = 1$
if succeed(Case-I), *goto* next iteration

(2) *find* $c \in C$ such that $|c \cap Q| \geq 2$, $|ptr(c) \cap (D^{(l)} - \tilde{T}^{(l)})| = 1$
and $|ptr(c) \cap (Q - (D^{(l)} - \tilde{T}^{(l)}))| = 0$
if succeed(Case-II), $c \leftarrow c - (Q - ptr(c))$ and *goto* next iteration

(3) *find* $c \in C$ such that $|c \cap (D^{(l)} - \tilde{T}^{(l)})| \geq 1$ and $|ptr(c) \cap Q| = 0$
if succeed(Case-III), *select* one element $d_n \in (c \cap (D^{(l)} - \tilde{T}^{(l)}))$
$c \leftarrow c - (Q - \{d_n\})$, $ptr(c) \leftarrow ptr(c) \cup \{d_n\}$ and *goto* next iteration

(4) (1),(2) and (3) *fail*(Case-IV), *select* one element $d_n \in (D^{(l)} - \tilde{T}^{(l)})$,
$c \leftarrow D - (Q - \{d_n\})$, $ptr(c) \leftarrow \{d_n\}$ and $C \leftarrow C \cup \{c\}$

}

}

return C and $\{ptr(c)|c \in C\}$

As a design issue, the number of checks should be reduced. The algorithm SFL-TFD II shown in Algorithm 5.1 is a basic algorithm to construct more general set of checks rather than the set of trivial checks for SFL/TFD ABFT systems. Algorithm SFL-TFD II finally returns the set of checks C and another set of checks $\{ptr(c)|c \in C\}$, where $c \in C$ is a maximal check(which may include unnecessary data elements) and $ptr(c) \subseteq C$ is a minimal check(none of whose elements can be excluded). As we can see it, SFL-TFD II contains some indeterminacies in the sorting of D_{**}'s and the selections of c(Case-I, II, III) and d_n(Case-III,IV) among their plural candidates. Certain strategies for fixing these indeterminacies toward check minimization are remained as a future problem.

Now, we will evaluate the computational complexity T_c of SFL-TFD II. The computation time of SFL-TFD II is analyzed as follows.

(1) "*construct*" all D_{ml}'s: $O(N^3)$
(2) "*construct*" all $\tilde{T}_{ml}$'s: $O(N^3)$
(3) the first "*for*" loop: $\frac{M(M-1)}{2}$ iterations
(4) "*sort*"(topological sort) operation: $O(N^2)$
(5) the second "*for*" loop: at most N iterations
(6) computations in the second "*for*" loop: $O(N^2)$

Totally, $T_c = O(M^2N^3)$.

5.3.2 A Design Example

We will consider a MPD graph shown in Fig. 5.1(a). The set of trivial checks for this system is given as follows,

$c_1 = \{d_{11}\}$
$c_2 = \{d_{12}\}$
$c_3 = \{d_{13}\}$
$c_4 = \{d_{21}\}$
$c_5 = \{d_{31}\}$
$c_6 = \{d_{32}\}$
$c_7 = \{d_{41}\}$
$c_8 = \{d_{42}\}$
$c_9 = \{d_{43}\}$
$c_{10} = \{d_{51}\}$

Figure 5.1(b) illustrates the resultant ABFT system(ABFT system I), where a thick line between data element and a check denotes that data element at its one end is contained in a check at the other end.

In the following, we will show a SFL/TFD ABFT system(ABFT system II) obtained by using Algorithm SFL-TFD II.

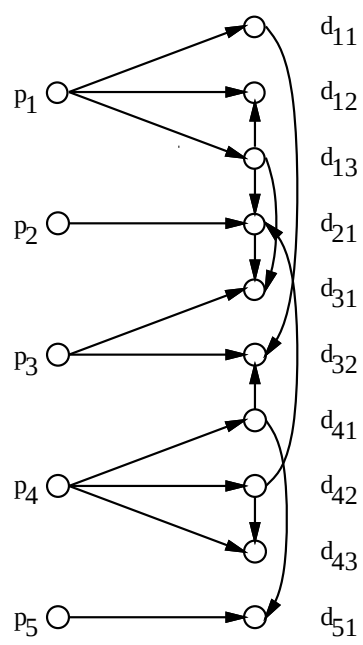
From the MPD graph in Fig. 5.1(a), $D_1 = \{\{d_{11},\ d_{32}\}, \{d_{12}\}, \{d_{13}, d_{12}, d_{21}, d_{31}\}\}$, $D_2 = \{\{d_{21}, d_{31}\}\}$, $D_3 = \{\{d_{31}\}, \{d_{32}\}\}$, $D_4 = \{\{d_{41}, d_{32}, d_{51}\}, \{d_{42}, d_{43},\ d_{21}, d_{31}\}, \{d_{43}\}\}$, $D_5 = \{\{d_{51}\}\}$ and $\tilde{T}_{13} = \{d_{31}\}$. As we have pointed out it in the previous section, Algorithm SFL-TFD II contains some indeterminacies. We introduce a simple heuristic in the selection of d_n from $c \cap (D^{(l)} - \tilde{T}^{(l)})$ or $(D^{(l)} - \tilde{T}^{(l)})$. That is, for each candidate d_{**}, we count the number of D_{mi}'s($1 \leq m \leq M,\ 1 \leq i \leq |Adj(p_m)|$) which contain d_{**}, and d_{**} which gives the maximum count is selected as d_n in Case-III and Case-IV. This heuristic is due to the expectation that a check containing d_{**} which is included in many D_{mi}'s may possibly be re-used for distinguishing various pairs of fault patterns. The final result of SFL-TFD II using d_n selection heuristic is shown in the following,

1. D_1 and D_j, $2 \leq j \leq 5$:
   $c_1 = \{\underline{d_{13}}\}$
   $c_2 = \{d_{13}, \underline{d_{32}}, d_{41}, d_{42}\}$
   $c_3 = \{d_{13}, \underline{d_{31}}\}$
   $c_4 = \{d_{11}, \underline{d_{12}}, d_{13}, d_{21}, d_{41}, d_{42}\}$
   $c_5 = \{\underline{d_{11}}, d_{13}, d_{21}, d_{41}, d_{42}, \underline{d_{43}}, d_{51}\}$
   $c_6 = \{d_{13}, d_{21}, d_{31}, d_{42}, \underline{d_{51}}\}$

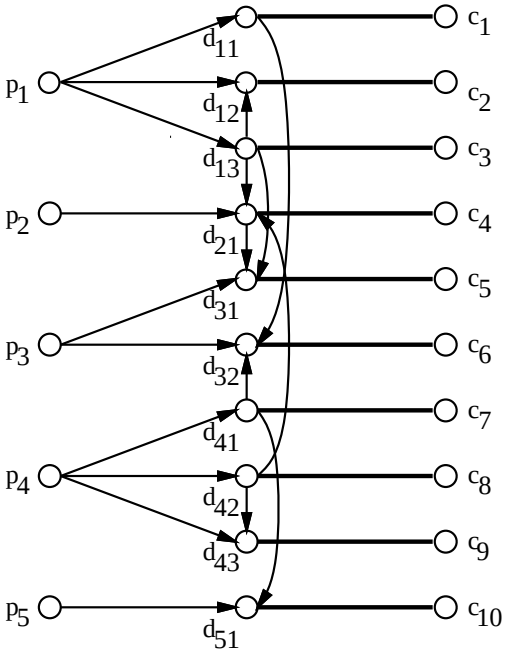
2. D_2 and D_j, $3 \leq j \leq 5$:
   $c_4 = \{d_{11}, \underline{d_{12}}, d_{13}, \underline{d_{42}}\}$
   $c_5 = \{\underline{d_{11}}, d_{13}, \underline{d_{21}}, d_{41}, d_{42}, \underline{d_{43}}, d_{51}\}$
   $c_6 = \{d_{13}, d_{42}, \underline{d_{51}}\}$

3. D_3 and D_j, $4 \leq j \leq 5$: no updated

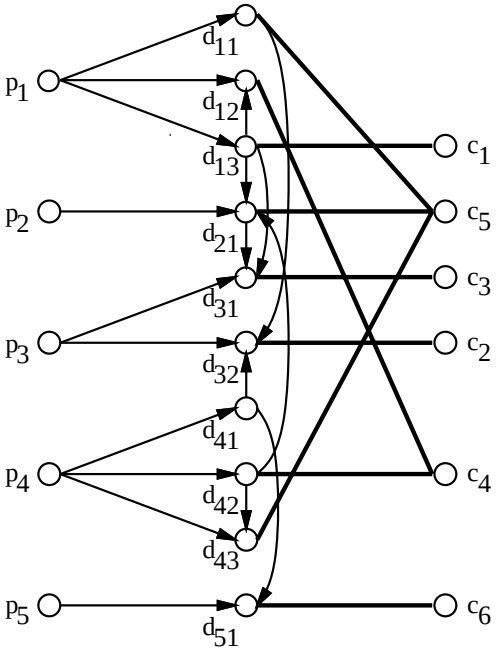
4. D_4 and D_j, $5 \leq j \leq 5$:
   $c_2 = \{d_{13}, \underline{d_{32}}, d_{42}\}$
   $c_5 = \{\underline{d_{11}}, d_{13}, \underline{d_{21}}, d_{41}, d_{42}, \underline{d_{43}}\}$



(a) A MPD graph G(V,E).



(b) The resultant ABFT system I with the set of trivial checks.



(c) The resultant ABFT system II by SFL-TFD II.

Figure 5.1: Examples for SFL/TFD ABFT system: MID.

fault patterns	error patterns	syndrome I $c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}$	syndrome II $c_1c_2c_3c_4c_5c_6$
$\{ p_1 \}$	$\{d_{11}, d_{32}\}$	1 0 0 0 0 1 0 0 0 0	0 1 0 0 1 0
	$\{d_{12}\}$	0 1 0 0 0 0 0 0 0 0	0 0 0 1 0 0
	$\{d_{12}, d_{13}, d_{21}, d_{31}\}$	0 1 1 1 1 0 0 0 0 0	1 0 1 1 1 0
	$\{d_{12}, d_{13}, d_{21}\}$	0 1 1 1 0 0 0 0 0 0	1 0 0 1 1 0
	$\{d_{11}, d_{12}, d_{32}\}$	1 1 0 0 0 1 0 0 0 0	0 1 0 1 1 0
	$\{d_{11}, d_{12}, d_{13}, d_{21}, d_{31}, d_{32}\}$	1 1 1 1 1 1 0 0 0 0	1 1 1 1 X0
	$\{d_{11}, d_{12}, d_{13}, d_{21}, d_{32}\}$	1 1 1 1 0 1 0 0 0 0	1 1 0 1 X0
	$\{d_{13}, d_{21}, d_{31}\}$	0 0 1 1 1 0 0 0 0 0	1 0 1 0 1 0
	$\{d_{13}, d_{21}\}$	0 0 1 1 0 0 0 0 0 0	1 0 0 0 1 0
	$\{d_{11}, d_{13}, d_{21}, d_{31}, d_{32}\}$	1 0 1 1 1 1 0 0 0 0	1 1 1 0 X0
	$\{d_{11}, d_{13}, d_{21}, d_{32}\}$	1 0 1 1 0 1 0 0 0 0	1 1 0 0 X0
$\{ p_2 \}$	$\{d_{21}, d_{31}\}$	0 0 0 1 1 0 0 0 0 0	0 0 1 0 1 0
$\{ p_3 \}$	$\{d_{31}\}$	0 0 0 0 1 0 0 0 0 0	0 0 1 0 0 0
	$\{d_{32}\}$	0 0 0 0 0 1 0 0 0 0	0 1 0 0 0 0
	$\{d_{31}, d_{32}\}$	0 0 0 0 1 1 0 0 0 0	0 1 1 0 0 0
$\{ p_4 \}$	$\{d_{32}, d_{41}, d_{51}\}$	0 0 0 0 0 1 1 0 0 1	0 1 0 0 0 1
	$\{d_{21}, d_{31}, d_{42}, d_{43}\}$	0 0 0 1 1 0 0 1 1 0	0 0 1 1 X1
	$\{d_{43}\}$	0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 0
	$\{d_{21}, d_{31}, d_{32}, d_{41}, d_{42}, d_{43}, d_{51}\}$	0 0 0 1 1 1 1 1 1 1	0 1 1 1 1 1
	$\{d_{32}, d_{41}, d_{43}, d_{51}\}$	0 0 0 0 0 1 1 0 1 1	0 1 0 0 1 1
	$\{d_{21}, d_{31}, d_{42}\}$	0 0 0 1 1 0 0 1 0 0	0 0 1 1 0 0
	$\{d_{21}, d_{31}, d_{32}, d_{41}, d_{42}, d_{51}\}$	0 0 0 1 1 1 1 1 0 1	0 1 1 1 1 1
$\{ p_5 \}$	$\{d_{51}\}$	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 1

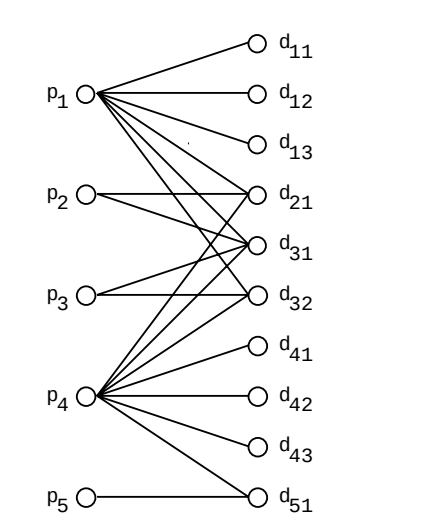
Figure 5.2: Syndromes for ABFT system I and II: MID.

In the above list, data elements with underlines in each check c_q are the elements in $ptr(c_q)$. The resultant ABFT system is illustrated in Fig. 5.1(c), where $\{ptr(c_q)|1 \leq q \leq 6\}$ is used as a complete set of checks.

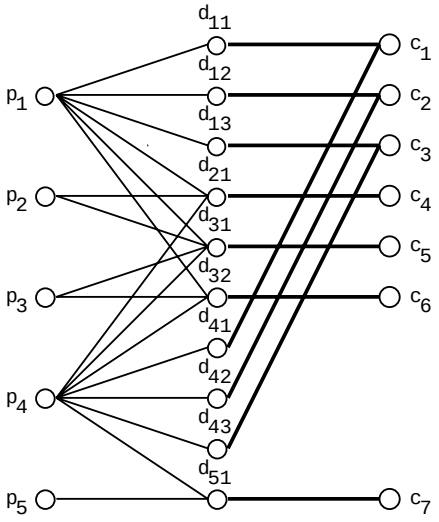
The syndrome for each error pattern induced by single-fault patterns $\{p_1\}$, $\{p_2\}$, $\{p_3\}$, $\{p_4\}$, and $\{p_5\}$ in ABFT system I and II of Fig. 5.1 is shown in syndrome I and syndrome II, respectively, of Fig. 5.2. In this figure, "X" denotes that the check is unpredictable due to $(g, 1)$ check($\mathcal{C}3$). Since the syndrome is different for any pair of fault patterns and at least one bit of the syndrome is 1 for any error pattern, both ABFT systems are certainly single-fault locatable and detectable. This ABFT system is also two-fault detectable from Corollary 5.1.

5.3.3 Comparison with Conventional PDC Graph Model

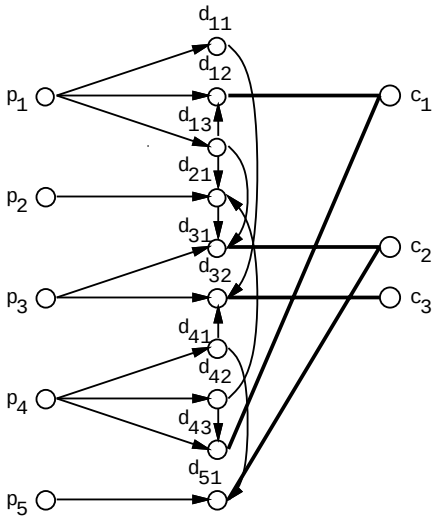
Consider MPD graph of Fig. 5.1(a). The corresponding conventional PD graph can be obtained by connecting each data element which is affected by a processor to the processor as shown in Fig. 5.3(a). In conventional PD graph, the set of error patterns induced by a



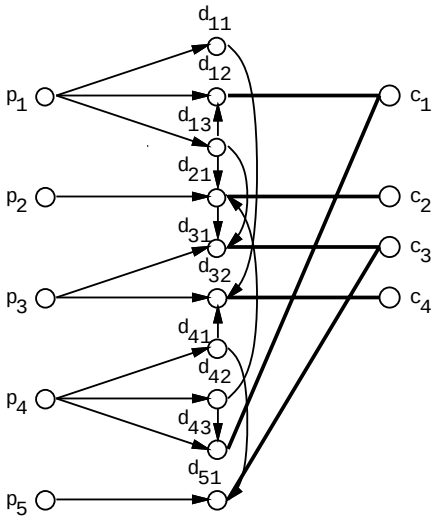
(a) The conventional PD graph.



(b) A SFD ABFT system for (a).



(c) A SFD ABFT system for Fig.5.1(a): SID.



(d) A SFD ABFT system for Fig.5.1(a): MID.

Figure 5.3: The comparisons in designing single-fault detectable ABFT system: MID.

faulty processor consists of all available combinations of data elements which are connected to the processor. Hence, there may be some redundant error patterns which do not appear in real applications. For example, the set of error patterns for the single-fault pattern $\{p_2\}$ is $\{\{d_{21}\},\{d_{31}\},\{d_{21},d_{31}\}\}$ in the conventional PD graph of Fig. 5.3(a) and the one is $\{\{d_{21},d_{31}\}\}$ in MPD graph of Fig. 5.1(a) under MID error model. There are two redundant error patterns $\{d_{21}\}$ and $\{d_{31}\}$. On the other hand, the error pattern $\{d_{31}\}$ is also induced by the single-fault patterns $\{p_1\}$, $\{p_3\}$ and $\{p_4\}$ in the conventional PD graph. Hence, there is no check which is distinguishable between $\{p_2\}$ and $\{p_1\}/\{p_3\}/\{p_4\}$.

Suppose that we want to construct a set of checks such that the designated system is single-fault detectable. For the case of the conventional PD graph(Fig. 5.3(a)), at least *seven* checks are required to detect single-fault(Fig. 5.3(b)). However, for the case of the MPD graph(Fig. 5.1(a)), we can detect single-fault to *three* checks and *four* checks under

SID error model(Fig. 5.3(c)) and MID error model(Fig. 5.3(d)), respectively. In fact, the set of error patterns to be considered under the MID model is a superset of the one under the SID model, but still a subset of the one under the conventional PD graph model, and in most cases that there exist data dependencies between primary outputs after mapping DG for a given algorithm to processors, the designated ABFT system based on MPD graph model under both SID error model and MID error model can be implemented with fewer checks than the conventional PDC graph model. Of course, the decrease of the number of checks does not always guarantee higher efficiency of the desired ABFT system. To design a well optimized ABFT system, we need to design checks with regarding hardware/software implementation of each check. However, for the case of a simple $(g, 1)$ check such as checksum technique, we can presume such tendency that fewer number of checks and smaller g of each check contribute to lower overhead. The examination of how the number of checks affects the efficiency of the desired ABFT system in a practical situation is remained as a future work.

5.4 Conclusion

In this chapter, we proposed a checking scheme for single-fault detectability and locatability on a sophisticated error occurrence/propagation model: MID model. Also, we gave a basic algorithm SFL-TFD II for constructing checks, and demonstrated a design example for SFL/TFD ABFT system based on the algorithm. The algorithm contains some indeterminacies in the sorting of D_{**}'s and the selections of c and d_n. Certain strategies for fixing these indeterminacies to minimize the number of checks are remained as a future problem. As a result, while the complexity and the number of checks tend to increase compared to SID error model in compensation for improving the accuracy of error propagation model, the effectiveness of the MPD graph model under the MID error model holds good compared to the conventional PD graph model.

Chapter 6

A Strategy for Mapping Checks to System Processors

6.1 Introduction

Most earlier work in the design of ABFT systems assumes that the operations for checking are performed by processors which are either fault-free or have some self-checking property. However, the checking operations are usually a part of an ABFT system and are likely to be performed on the system processors. When the checking operations on the system processors fail, such checks become unreliable. Therefore, the accuracy of the computations is dependent on the reliability of the processors performing checking operations as well. Banerjee and Abraham [4] introduced check evaluating nodes in their graph model and showed how to analyze such a system for fault tolerance. Also, Yajnik and Jha [7] used an extended graph-theoretic ABFT model to consider the processors for computing checks to be a part of the ABFT system and allowed faults in these processors.

The conventional fault tolerant schemes for VLSI array architectures have been mainly concentrated on CED schemes, and some efforts have been made for concurrent error correction [6], [10], [11], [12]. However, fault location scheme has not received much attention. To solve the problem of locating faults on VLSI array architectures, more complex schemes considering with both *time* redundancy and *hardware* redundancy, are required.

In this chapter, the ABFT system is extended by introducing some redundancies to be compared to the sum of data elements in checks and mapping such checks to the system processors such a way that the system still maintains the designated fault tolerance. And we present a checking scheme for synthesizing single-fault locatable FIR filter based on the extended modified processor-data-check(EMPDC) graph model. As a result, a fault tolerant FIR filter is implemented on systolic array.

The rest of this chapter is organized as follows. In Section 6.2, the single-fault detectability and locatability of ABFT system based on EMPDC graph model is represented. A method for designing single-fault locatable FIR filter is discussed in Section 6.3. Finally, Section 6.4 is used for conclusion.

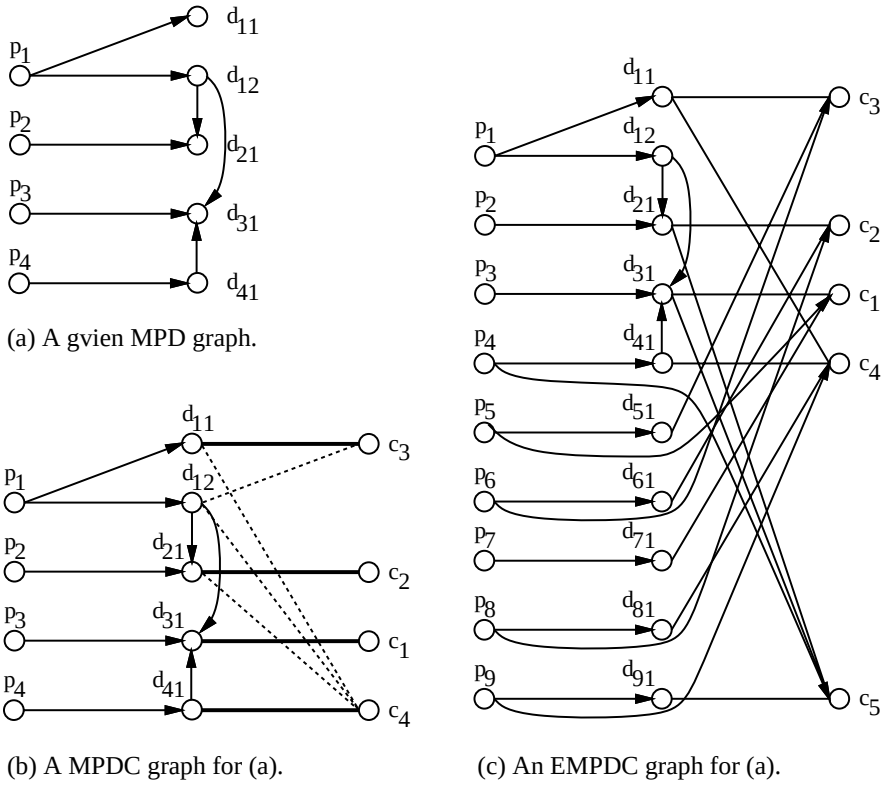


Figure 6.1: An example for EMPDC graph.

6.2 EMPDC Graph Model

In an EMPDC graph model to be considered, all the processor nodes, all the data nodes, and all the check nodes form the set P, D, and C, respectively. The EMPDC graph has five types of nodes.

1. **Primary processor nodes** which perform nominal computations or checking operations.
2. **Primary data nodes** which are results of the nominal computations.
3. **Check nodes** which represent checks.
4. **Redundant data nodes** which represent data elements to be compared to the results of the nominal computations.
5. **Redundant processor nodes** which perform redundant computations to produce redundant data elements and/or perform checking operations.

In general, the processors for computing checks can be of either the primary processor node type or the redundant processor node type. In EMPDC graph, there is a directed edge from the processor $p_i \in P$ to the data element $d_j \in D$ if p_i produces d_j. Similarly, there is a directed edge from p_i to the check $c_k \in C$ if c_k is implemented on p_i. Also, there is an edge from d_j to c_k if d_j is checked by c_k.

For a given MPD graph(Fig. 6.1(a)), a modified processor-data-check(MPDC) graph for SFL/TFD under the checking scheme proposed in Chapter 4 is shown in Fig. 6.1(b).

And an EMPDC graph(Fig. 6.1(c)) is constructed by introducing redundant data elements and mapping checks on either the primary processors or the redundant processors. According to the EMPDC graph, processors p_1, p_2, p_3, p_4 perform nominal computations to produce primary data elements $d_{11}, d_{12}, d_{21}, d_{31}, d_{41}$, whereas processors p_5, p_6, p_7, p_8, p_9 perform redundant computations to produce redundant data elements $d_{51}, d_{61}, d_{71}, d_{81}, d_{91}$ to be compared to the sum of primary data elements in each check. In this example, d_{51} is the redundant data element to be compared to d_{11} in checking operation of c_3, d_{61} is to be compared to d_{21} in checking operation of c_2, and so on. Also, checks c_1, c_2, c_3, c_4, c_5 are mapped to processors p_5, p_8, p_6, p_9, p_4, respectively, and such checks guarantee that the ABFT system is totally single-fault locatable.

Throughout this chapter, we use SID model as error occurrence/propagation model. Also, we assume that the faults in only the checking operations to be computed in the processors p_i and p_j are not necessary to be distinguished to other faults because the results for original computations are correct, but such faults have to be detected.

In the following, $Rch(d_{ml})$ denotes the set of data elements reachable from the data element d_{ml}. W_{mx} is one of all available unions of elements in D_m. d_x denotes the redundant data element to be compared to the sum of primary data elements in a check c_x. And $\tilde{c}_x$ denotes a variable regarded as adjacent element of a processor computing the checking operations for c_x. Also, M' denotes the number of processors(both primary processors and redundant processors), $M' \geq M$.

Lemma 6.1 *An extended ABFT system is single-fault detectable if for any W_{ix} which is one of unions of elements in D_i, $1 \leq i \leq M$, there is a check c_x such that*

1. $|c_x \cap W_{ix}| = 1$
2. $|c_x \cap (Rch(d_x) - \{d_x\})| = 0$
3. $|\{d_x, \tilde{c}_x\} \cap Adj(p_i)| = 0$
4. $|\{d_x, \tilde{c}_x\} \cap Adj(p_m)| \leq 1,\ 1 \leq m \leq M'$

Proof: Note that W_{ix} is one of all error patterns which can be occurred when a processor p_i fails. Thus, all available unions of elements in D_i become all error patterns induced by the faulty processor p_i. By Condition 1, the check c_x certainly outputs "1" for the x-th one of error patterns induced by the fault pattern $\{p_i\}$. Similarly, by Condition 2, c_x certainly outputs "1" when d_x is erroneous. By Condition 3 and Condition 4, the reliability of checking operation for c_x is guaranteed for the error pattern W_{ix} when the processor p_i fails and for the error patterns including d_x when the processor p_m fails, respectively. □

Lemma 6.2 *An extended ABFT system is single-fault locatable if for any pair of W_{ix} and W_{jy} which are one of unions of elements in D_i and D_j, $i \neq j$, $1 \leq i \leq M$, $1 \leq j \leq M$, respectively, there is a set of checks $\{c_x, c_y, c_u, c_v\}$ such that*

1. $\boldsymbol{c_x}$:

   1a. $|c_x \cap (W_{ix} \oplus W_{jy})| = 1$

   1b. $|c_x \cap (W_{ix} \cap W_{jy})| = 0$

   1c. $|c_x \cap (Rch(d_x) - \{d_x\})| = 0$

   1d. $|\{d_x, \tilde{c}_x\} \cap (Adj(p_i) \cup Adj(p_j))| = 0$

$1f.\ |\{d_x, \tilde{c}_x\} \cap Adj(p_m)| \leq 1,\ 1 \leq m \leq M'$

$1f1.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_x\},\ |c_r \cap Rch(d_x)| \neq 1$

$1f2.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_x, |c_r \cap Rch(d_x)| = 1\},\ |\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_x) \cap Rch(Adj(p_{m'})))| \geq 1,\ m' \neq m,\ 1 \leq m' \leq M'$

2. *for* $|c_x \cap W_{ix}| = 1$

$2a.\ \boldsymbol{c_y}$:

$2a1.\ |c_y \cap W_{jy}| = 1$

$2a2.\ |c_y \cap (Rch(d_y) - \{d_y\})| = 0$

$2a3.\ |\{d_y, \tilde{c}_y\} \cap Adj(p_j)| = 0$

$2a4.\ |\{d_y, \tilde{c}_y\} \cap Adj(p_m)| \leq 1,\ 1 \leq m \leq M'$

$2a4\text{-}1.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_y\},\ |c_r \cap Rch(d_y)| \neq 1$

$2a4\text{-}2.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_y, |c_r \cap Rch(d_y)| = 1\},\ |\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_y) \cap Rch(Adj(p_{m'})))| \geq 1,\ m' \neq m,\ 1 \leq m' \leq M'$

$2b.\ \boldsymbol{c_u}$:

$2b1.\ |c_u \cap (W_{ix} \oplus Rch(d_x))| = 1$

$2b2.\ |c_u \cap (W_{ix} \cap Rch(d_x))| = 0$

$2b3.\ |c_u \cap (Rch(d_u) - \{d_u\})| = 0$

$2b4.\ |\{d_u, \tilde{c}_u\} \cap (Adj(p_i) \cup Adj(p_x) \cup Adj(p_{c_x}))| = 0$

$2b5.\ |\{d_u, \tilde{c}_u\} \cap Adj(p_m)| \leq 1,\ 1 \leq m \leq M'$

$2b5\text{-}1.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_u\},\ |c_r \cap Rch(d_u)| \neq 1$

$2b5\text{-}2.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_u, |c_r \cap Rch(d_u)| = 1\},\ |\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_u) \cap Rch(Adj(p_{m'})))| \geq 1,\ m' \neq m,\ 1 \leq m' \leq M'$

$2c.\ \boldsymbol{c_v}$:

$2c1.\ |c_v \cap (W_{jy} \oplus Rch(d_y))| = 1$

$2c2.\ |c_v \cap (W_{jy} \cap Rch(d_y))| = 0$

$2c3.\ |c_v \cap (Rch(d_v) - \{d_v\})| = 0$

$2c4.\ |\{d_v, \tilde{c}_v\} \cap (Adj(p_j) \cup Adj(p_y) \cup Adj(p_{c_y}))| = 0$

$2c5.\ |\{d_v, \tilde{c}_v\} \cap Adj(p_m)| \leq 1,\ 1 \leq m \leq M'$

$2c5\text{-}1$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_v\},\ |c_r \cap Rch(d_v)| \neq 1$

$2c5\text{-}2$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_v, |c_r \cap Rch(d_v)| = 1\},\ |\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_v) \cap Rch(Adj(p_{m'})))| \geq 1,\ m' \neq m,\ 1 \leq m' \leq M'$

3. *for* $|c_x \cap W_{jy}| = 1$

$3a.\ \boldsymbol{c_y}$:

$3a1.\ |c_y \cap W_{ix}| = 1$

$3a2.\ |c_y \cap (Rch(d_y) - \{d_y\})| = 0$

$3a3.\ |\{d_y, \tilde{c}_y\} \cap Adj(p_i)| = 0$

$3a4.\ |\{d_y, \tilde{c}_y\} \cap Adj(p_m)| \leq 1,\ 1 \leq m \leq M'$

$3a4\text{-}1.$ *if* $^{\forall} c_r : \{c_r | c_r \neq c_y\},\ |c_r \cap Rch(d_y)| \neq 1$

3a4-1. *if* $^{\forall}c_r : \{c_r | c_r \neq c_y, |c_r \cap Rch(d_y)| = 1\}$, $|\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_y) \cap Rch(Adj(p_{m'})))| \geq 1$, $m' \neq m$, $1 \leq m' \leq M'$

3b. $\boldsymbol{c_u}$:

3b1. $|c_u \cap (W_{jy} \oplus Rch(d_x))| = 1$

3b2. $|c_u \cap (W_{jy} \cap Rch(d_x))| = 0$

3b3. $|c_u \cap (Rch(d_u) - \{d_u\})| = 0$

3b4. $|\{d_u, \tilde{c}_u\} \cap (Adj(p_j) \cup Adj(p_x) \cup Adj(p_{c_x}))| = 0$

3b5. $|\{d_u, \tilde{c}_u\} \cap Adj(p_m)| \leq 1$, $1 \leq m \leq M'$

3b5-1. *if* $^{\forall}c_r : \{c_r | c_r \neq c_u\}$, $|c_r \cap Rch(d_u)| \neq 1$

3b5-2. *if* $^{\forall}c_r : \{c_r | c_r \neq c_u, |c_r \cap Rch(d_u)| = 1\}$, $|\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_u) \cap Rch(Adj(p_{m'})))| \geq 1$, $m' \neq m$, $1 \leq m' \leq M'$

3c. $\boldsymbol{c_v}$:

3c1. $|c_v \cap (W_{ix} \oplus Rch(d_y))| = 1$

3c2. $|c_v \cap (W_{ix} \cap Rch(d_y))| = 0$

3c3. $|c_v \cap (Rch(d_v) - \{d_v\})| = 0$

3c4. $|\{d_v, \tilde{c}_v\} \cap (Adj(p_i) \cup Adj(p_y) \cup Adj(p_{c_y}))| = 0$

3c5. $|\{d_v, \tilde{c}_v\} \cap Adj(p_m)| \leq 1$, $1 \leq m \leq M'$

3c5-1. *if* $^{\forall}c_r : \{c_r | c_r \neq c_v\}$, $|c_r \cap Rch(d_v)| \neq 1$

3c5-1. *if* $^{\forall}c_r : \{c_r | c_r \neq c_v, |c_r \cap Rch(d_v)| = 1\}$, $|\{\tilde{c}_r\} \cap Adj(p_m)| = 1$ *or* $|c_r \cap (Rch(d_v) \cap Rch(Adj(p_{m'})))| \geq 1$, $m' \neq m$, $1 \leq m' \leq M'$

4. *for any pair* s(*one of* x *and* u) *and* s'(*one of* y *and* v), $|\{d_s, d_{s'}, \tilde{c}_s, \tilde{c}_{s'}\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$, $m' \neq m$, $1 \leq m \leq M'$, $1 \leq m' \leq M'$

Proof: Note that the unions W_{ix} and W_{jy} are one of error patterns induced by fault patterns $\{p_i\}$ and $\{p_j\}$, respectively. We assume that the faults in only the checking operations to be computed in the processors p_i and p_j are not necessary to be distinguished to other faults because the results for original computations are correct. By Condition 1, the check c_x detects one error pattern $W_{ix}(W_{jy})$, and also distinguishes between W_{ix} and W_{jy} because c_x certainly outputs "1" for exactly one of such two error patterns. Furthermore, the reliability for the checking operation of c_x is guaranteed by Condition 1 and Condition 4. By Condition 2a(Condition 3a) and Condition 4, c_y detects the other error pattern $W_{jy}(W_{ix})$, and the reliability for the checking operation is guaranteed. Also, by Condition 2b(Condition 3b) and Condition 4, c_u distinguishes between W_{ix} and $Rch(d_x)$, and the the reliability for the checking operation is guaranteed. Similarly, by Condition 2c(Condition 3c) and Condition 4, c_v distinguishes between W_{iy} and $Rch(d_y)$, and the the reliability for the checking operation is guaranteed. □

As an example, let us consider the EMPDC graph illustrated in Fig. 6.1(c). For p_1 and p_2, $D_1 = \{\{d_{11}\}, \{d_{12}, d_{21}, d_{31}\}\}$, $D_2 = \{\{d_{21}\}\}$. If four checks c_x, c_y, c_u and c_v, which satisfy Lemma 6.2, are defined as follows,

1. $D_{11} = \{d_{11}\}$ and $D_{21} = \{d_{21}\}$:
   $c_x = c_3$, $d_x = d_{51}$, $p_x = p_5$, and $p_{c_x} = p_6$
   $c_y = c_2$, $d_y = d_{61}$, $p_y = p_6$, and $p_{c_y} = p_8$
   $c_u = c_4$, $d_u = d_{81}$, $p_u = p_8$, and $p_{c_u} = p_9$
   $c_v = c_5$, $d_v = d_{91}$, $p_v = p_9$, and $p_{c_v} = p_4$

2. $D_{12} = \{d_{12}, d_{21}, d_{31}\}$ and $D_{21} = \{d_{21}\}$:
   $c_x = c_1$, $d_x = d_{71}$, $p_x = p_7$, and $p_{c_x} = p_5$
   $c_y = c_2$, $d_y = d_{61}$, $p_y = p_6$, and $p_{c_y} = p_8$
   $c_u = c_5$, $d_u = d_{91}$, $p_u = p_9$, and $p_{c_u} = p_4$
   $c_v = c_5$, $d_v = d_{91}$, $p_v = p_9$, and $p_{c_v} = p_4$

as a result, any pair of error patterns, one is induced by p_1 and the other is induced by p_2, can be distinguished by each other, and each of error patterns induced by p_1 andp_2 is detected. Since there are such four checks c_x, c_y, c_u and c_v, for any pair of fault patterns, the ABFT system is single-fault locatable.

In the next section, we will discuss a method for constructing checks for single-fault locatable ABFT systems based on Lemma 6.2 by using the EMPDC graph model.

6.3 A Basic Algorithm MAP-EMPDC

Now, we introduce a basic algorithm to construct checks and redundant data elements for single-fault locating ABFT system, and to map such checks and redundant data elements to the system processors. Let c_x, c_y, c_u and c_v be checks satisfying Lemma 6.2. Algorithm 6.1 finds a set of checks C and a set of data elements $ptr(c_s)$ which have to be checked by $c_s \in C$. And the algorithm selects(or adds) a processor p_k computing redundant data element d_s to be compared to the sum of the primary data elements in $c_s \in C$ and a processor p_l computing the checking operation $\tilde{c}_s$ for $c_s \in C$. Finally, the algorithm always return $P'(\supseteq P)$, $D'(\supseteq D)$, C and $ptr(c_s)$ if there exists a set of checks given by Lemma 6.2. The outline of the algorithm for a pair of Q_I and Q_J is illustrated in Fig. 6.2.

For a pair of Q_I and Q_J, the algorithm MAP-EMPDC consists of four construction steps: (1) c_x(XMAP), (2) c_u(UMAP), (3) c_y(YMAP), (4) c_v(VMAP). The subroutine XMAP finds $c_x \in C$ given by Condition 1 of Lemma 6.2 by using the subroutine CONSTRUCT. If c_x satisfying the condition is not in C, c_x is newly added by the subroutine ADD. And then the processor p_k computing the redundant data element d_x and the processor p_l computing the checking operation $\tilde{c}_x$ for c_x are properly selected or newly added by the subroutine ADDP. Similarly, the subroutines YMAP, UMAP and VMAP return c_s, d_s, p_k and p_l which satisfy the conditions $2a$(or $3a$), $2b$(or $3b$), $2c$(or $3c$) and 4 of Lemma 6.2. If the redundant data element $d_u(d_v)$ and the checking operation $\tilde{c}_u(\tilde{c}_v)$ in the subroutine UMAP(VMAP) are computed in the same processor, then to guarantee the reliability of $c_u(c_v)$, $c_u(c_v)$ has to be examined whether $c_r \neq c_u(c_v)$ given by the condition $2b5(2c5)$(or $2b5(2c5)$) exists or not. It is executed by the subroutine RMAP. The subroutine RMAP returns c_r, d_r, p_r and $p_{r'}$, and is repeated until both p_s and p_r are not in P to limit the hardware resource to at most three times. If $|\{p_s, p_r\} \cap P| = 0$, the checking operation $\tilde{c}_r$ for c_r is mapped such that $|\{\tilde{c}_r\} \cap (Adj(p_s) \cap Adj(p_r))| = 0$.

On the other hand, the "*while*" loop in main routine MAP-EMPDC is executed until either Q_I or Q_J is empty set. In the next iteration, if c_x in Q_I, then Q_I is only updated. And if c_x in Q_J, then Q_J is only updated. Also, the "*while*" loop in the subroutine QMAP is executed until Q is empty set.

Algorithm 6.1

MAP-EMPDC

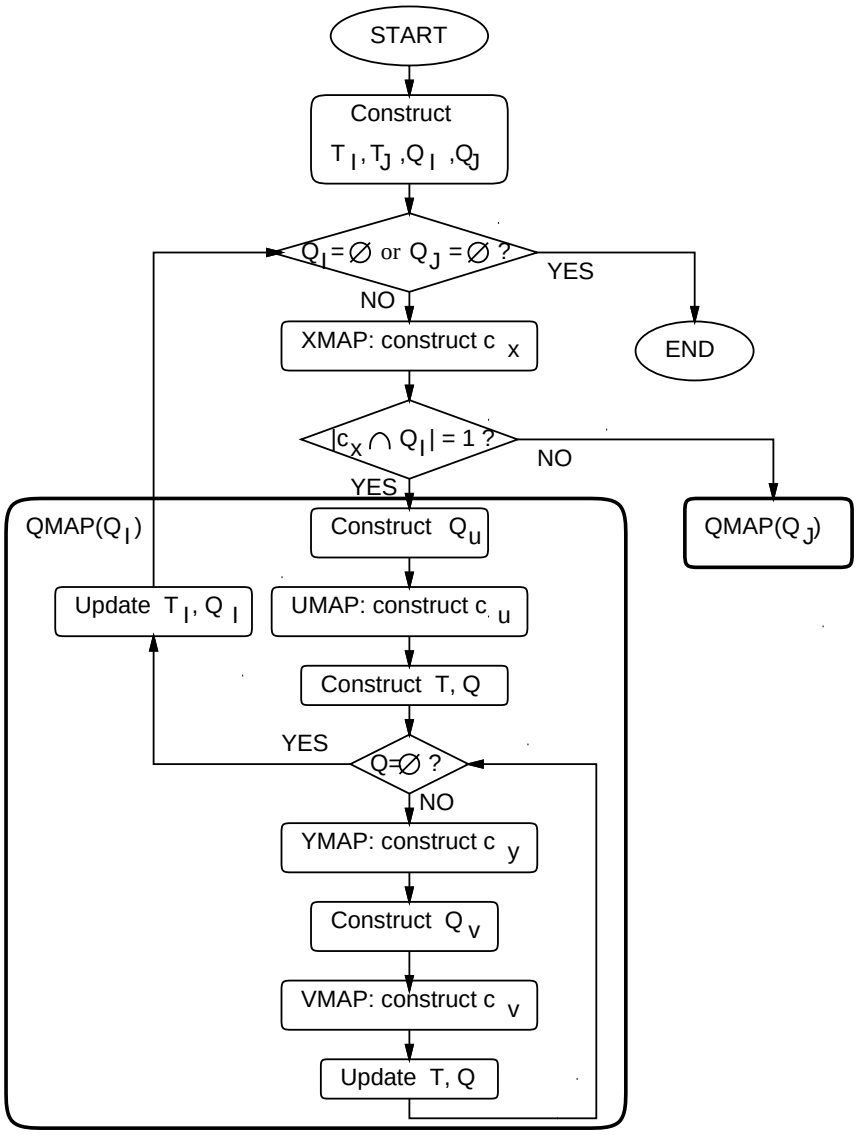


Figure 6.2: Flow diagram for a pair of Q_I and Q_J in Algorithm 6.1.

input : MPD graph $G(V, E)$
$M' \leftarrow M$, $P' \leftarrow P$, $D' \leftarrow D$, and $C \leftarrow \emptyset$
for $1 \leq i < j \leq M$
{
$T_I \leftarrow D_i$ and $T_J \leftarrow D_j$
$Q_I \leftarrow \bigcup_{w=1}^{|T_I|} D_{iw}$ and $Q_J \leftarrow \bigcup_{z=1}^{|T_J|} D_{jz}$
while$(Q_I \neq \emptyset$ & $Q_J \neq \emptyset)$
{
XMAP(c_x, Q_I, Q_J)
if$(|c_x \cap Q_I| = 1)$, QMAP(Q_I)
if$(|c_x \cap Q_J| = 1)$, QMAP(Q_J)

}
}
return P', D', C and $\{ptr(c)|c \in C\}$

QMAP(Q_*)
{
$Q_u \leftarrow \bigcap_{|c_x \cap D_{*w}|=1} D_{*w}$
UMAP($c_u, Q_u, Rch(d_x)$)
$T \leftarrow T_{\bar{*}}$ and $Q \leftarrow Q_{\bar{*}}$
while($Q \neq \emptyset$)
{
YMAP($c_y, Q, \emptyset$)
$Q_v \leftarrow \bigcap_{|c_y \cap D_{\bar{*}z}|=1} D_{\bar{*}z}$
VMAP($c_v, Q_v, Rch(d_y)$)
$T \leftarrow T - \bigcup_{|c_y \cap D_{\bar{*}z}|=1} \{D_{\bar{*}z}\}$ and $Q \leftarrow \bigcup_{D_{\bar{*}z} \in T} D_{\bar{*}z}$
}
$T_* \leftarrow T_* - \bigcup_{|c_x \cap D_{*w}|=1} \{D_{*w}\}$ and $Q_* \leftarrow \bigcup_{D_{*w} \in T_*} D_{*w}$
return
}

XMAP(c_x, Q_I, Q_J)
{
find $c_x \in C$ such that $|\{d_x, \tilde{c}_x\} \cap (Adj(p_i) \cup Adj(p_j))| = 0$, $|c_x \cap (Q_I \cup Q_J)| = 1$,
$|ptr(c_x) \cap (Q_I \oplus Q_J)| = 1$ and $|c_x \cap (Rch(d_x) - \{d_x\})| = 0$
if fail, *find* $c_x \in C$ such that $|\{d_x, \tilde{c}_x\} \cap (Adj(p_i) \cup Adj(p_j))| = 0$ and
$|c_x \cap (Rch(d_x) - \{d_x\})| = 0$
if succeed, CONSTRUCT(c_x, Q_I, Q_J)
if fail, ADD(c_x, d_x, Q_I, Q_J)
select p_k, $d_x \in Adj(p_k)$ such that $|\{d_x\} \cap (Adj(p_i) \cup Adj(p_j))| = 0$
if $|\{p_k\} \cap P'| = 0$, ADDP(p_k)
$Adj(p_k) \leftarrow Adj(p_k) \cup \{d_x\}$
select p_l, $\tilde{c}_x \in Adj(p_l)$ such that $|\{\tilde{c}_x\} \cap (Adj(p_i) \cup Adj(p_j))| = 0$
if $|\{p_l\} \cap P'| = 0$, ADDP(p_l)
$Adj(p_l) \leftarrow Adj(p_l) \cup \{\tilde{c}_x\}$
return
}

YMAP($c_y, Q, \emptyset$)
{
find $c_y \in C$ such that $|\{d_y, \tilde{c}_y\} \cap Adj(p_{\bar{*}})| = 0$,
$|\{d_x, d_y, \tilde{c}_x, \tilde{c}_y\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$,
$|\{d_u, d_y, \tilde{c}_u, \tilde{c}_y\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$,
$m \neq m'$, $1 \leq m \leq M'$, $1 \leq m' \leq M'$,
$|c_y \cap Q| = 1$, $|ptr(c_y) \cap Q| = 1$ and $|c_y \cap (Rch(d_y) - \{d_y\})| = 0$

if fail, $find\ c_y \in C$ such that $|\{d_y, \tilde{c}_y\} \cap Adj(p_*)| = 0$, $|c_y \cap (Rch(d_y) - \{d_y\})| = 0$
$|\{d_x, d_y, \tilde{c}_x, \tilde{c}_y\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$ and
$|\{d_u, d_y, \tilde{c}_u, \tilde{c}_y\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$,
$m \neq m'$, $1 \leq m \leq M'$, $1 \leq m' \leq M'$
if succeed, CONSTRUCT$(c_y, Q, \emptyset)$
if fail, ADD$(c_y, d_y, Q, \emptyset)$
$select\ p_k$, $d_y \in Adj(p_k)$ such that $|\{d_y\} \cap Adj(p_*)| = 0$
$if\ |\{p_k\} \cap P'| = 0$, ADDP$(p_k)$
$Adj(p_k) \leftarrow Adj(p_k) \cup \{d_y\}$
$select\ p_l$, $\tilde{c}_y \in Adj(p_l)$ such that $|\{\tilde{c}_y\} \cap Adj(p_*)| = 0$ and
$|\{\tilde{c}_y\} \cap (Adj(p_m) \cup Adj(p_{m'}))| = 0\ if$
$|\{d_x, d_y, \tilde{c}_x\} \cap (Adj(p_m) \cup Adj(p_{m'}))| = 3$ or
$|\{d_u, d_y, \tilde{c}_u\} \cap (Adj(p_m) \cup Adj(p_{m'}))| = 3$,
$m \neq m'$, $1 \leq m \leq M'$, $1 \leq m' \leq M'$
$if\ |\{p_l\} \cap P'| = 0$, ADDP$(p_l)$
$Adj(p_l) \leftarrow Adj(p_l) \cup \{\tilde{c}_y\}$
$return$
}

UMAP$(c_u, Q_u, Rch(d_x))$
{
$T \leftarrow Q_u$
$if(|\{d_x, \tilde{c}_x\} \cap Adj(p_m)| = 2, 1 \leq m \leq M'), T \leftarrow \emptyset$
$find\ c_u \in C$ such that $|\{d_u, \tilde{c}_u\} \cap (Adj(p_*) \cup Adj(p_x) \cup Adj(p_{c_x}))| = 0$,
$|c_u \cap (T \cup Rch(d_x))| = 1$, $|ptr(c_u) \cap (T \oplus Rch(d_x))| = 1$, and
$|c_u \cap (Rch(d_u) - \{d_u\})| = 0$
if fail, $find\ c_u \in C$ such that $|\{d_u, \tilde{c}_u\} \cap (Adj(p_*) \cup Adj(p_x) \cup Adj(p_{c_x}))| = 0$ and
$|c_u \cap (Rch(d_u) - \{d_u\})| = 0$
if succeed, CONSTRUCT$(c_u, T, Rch(d_x))$
if fail, ADD$(c_u, d_u, T, Rch(d_x))$
$select\ p_k$, $d_u \in Adj(p_k)$ such that
$|\{d_u\} \cap (Adj(p_*) \cup Adj(p_x) \cup Adj(p_{c_x}))| = 0$
$if\ |\{p_k\} \cap P'| = 0$, ADDP$(p_k)$
$Adj(p_k) \leftarrow Adj(p_k) \cup \{d_u\}$
$select\ p_l$, $\tilde{c}_u \in Adj(p_l)$ such that
$|\{\tilde{c}_u\} \cap (Adj(p_*) \cup Adj(p_x) \cup Adj(p_{c_x}))| = 0$
$if\ |\{p_l\} \cap P'| = 0$, ADDP$(p_l)$
$Adj(p_l) \leftarrow Adj(p_l) \cup \{\tilde{c}_u\}$
$if(|\{d_u, \tilde{c}_u\} \cap Adj(p_m)| = 2, 1 \leq m \leq M')$, RMAP$(d_r, c_r, d_u, c_u)$
$return$
}

VMAP$(c_v, Q_v, Rch(d_y))$
{
$T \leftarrow Q_v$
$if(|\{d_x, \tilde{c}_x\} \cap Adj(p_m)| = 2, 1 \leq m \leq M'), T \leftarrow \emptyset$
$find\ c_v \in C$ such that $|\{d_v, \tilde{c}_v\} \cap (Adj(p_*) \cup Adj(p_y) \cup Adj(p_{c_y}))| = 0$,

$|\{d_x, d_v, \tilde{c}_x, \tilde{c}_v\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$,
$|\{d_u, d_v, \tilde{c}_u, \tilde{c}_v\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$,
$m \neq m'$, $1 \leq m \leq M'$, $1 \leq m' \leq M'$,
$|c_v \cap (T \cup Rch(d_y))| = 1$, $|ptr(c_v) \cap (T \oplus Rch(d_y))| = 1$
and $|c_v \cap (Rch(d_v) - \{d_v\})| = 0$

if fail, *find* $c_v \in C$ such that $|\{d_v, \tilde{c}_v\} \cap (Adj(p_*) \cup Adj(p_y) \cup Adj(p_{c_y}))| = 0$,
$|\{d_x, d_v, \tilde{c}_x, \tilde{c}_v\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$,
$|\{d_u, d_v, \tilde{c}_u, \tilde{c}_v\} \cap (Adj(p_m) \cup Adj(p_{m'}))| \leq 3$, $m \neq m'$,
$1 \leq m \leq M'$, $1 \leq m' \leq M'$, and $|c_v \cap (Rch(d_v) - \{d_v\})| = 0$

if succeed, CONSTRUCT$(c_v, T, Rch(d_y))$

if fail, ADD$(c_v, d_v, T, Rch(d_y))$

select p_k, $d_v \in Adj(p_k)$ such that
$|\{d_v\} \cap (Adj(p_*) \cup Adj(p_y) \cup Adj(p_{c_y}))| = 0$

if $|\{p_k\} \cap P'| = 0$, ADDP(p_k)

$Adj(p_k) \leftarrow Adj(p_k) \cup \{d_v\}$

select p_l, $\tilde{c}_v \in Adj(p_l)$ such that
$|\{\tilde{c}_v\} \cap (Adj(p_*) \cup Adj(p_y) \cup Adj(p_{c_y}))| = 0$
and $|\{\tilde{c}_v\} \cap (Adj(p_m) \cup Adj(p_{m'}))| = 0$
if $|\{d_x, d_v, \tilde{c}_x\} \cap (Adj(p_m) \cup Adj(p_{m'}))| = 3$ or
$|\{d_u, d_v, \tilde{c}_u\} \cap (Adj(p_m) \cup Adj(p_{m'}))| = 3$,
$m \neq m'$, $1 \leq m \leq M'$, $1 \leq m' \leq M'$

if $|\{p_l\} \cap P'| = 0$, ADDP(p_l)

$Adj(p_l) \leftarrow Adj(p_l) \cup \{\tilde{c}_v\}$

if$(|\{d_v, \tilde{c}_v\} \cap Adj(p_m)| = 2, 1 \leq m \leq M')$, RMAP$(d_r, c_r, d_v, c_v)$

return

}

RMAP(d_r, c_r, d_s, c_s)

{

find $c_r \in C$ such that for $c_r \neq c_s$, $|c_r \cap Rch(d_s)| = 1$, $|ptr(c_r) \cap Rch(d_s)| = 1$,
$|c_r \cap (Rch(d_s) \cap Rch(Adj(p_k)))| = 0$, $k \neq s$, $1 \leq k \leq M'$,
and $|\{d_r, \tilde{c}_r\} \cap Adj(p_s)| = 0$

if fail, *select* one $d_n \in ((Rch(d_s) \cap Adj(p_s)) - Rch(Adj(p_k)))$, $k \neq s$, $1 \leq k \leq M'$

select d_r's

$D_r \leftarrow \bigcup \{d_r\}$

$c_r \leftarrow D - ((Rch(d_s) - \{d_n\}) - Rch(d_r)) \cup D_r$

$ptr(c_r) \leftarrow \{d_n\} \cup D_r$

for each d_r: $|\{d_r\} \cap D'| = 0$, $D' \leftarrow D' \cup \{d_r\}$

$C \leftarrow C \cup \{c_r\}$

select p_r, $d_r \in Adj(p_r)$ such that $|\{d_r\} \cap Adj(p_s)| = 0$

if $|\{p_r\} \cap P'| = 0$, ADDP(p_r)

$Adj(p_r) \leftarrow Adj(p_r) \cup \{d_r\}$

select $p_{r'}$, $\tilde{c}_r \in Adj(p_{r'})$ such that $|\{\tilde{c}_r\} \cap Adj(p_s)| = 0$ or
$|\{\tilde{c}_r\} \cap Adj(p_s) \cap Adj(p_r)| = 0$ if $|\{p_s, p_r\} \cap P| = 0$

if $|\{p_{r'}\} \cap P'| = 0$, ADDP$(p_{r'})$

$Adj(p_{r'}) \leftarrow Adj(p_{r'}) \cup \{\tilde{c}_r\}$

if$(|\{d_r, \tilde{c}_r\} \cap Adj(p_m)| = 2, 1 \leq m \leq M')$, RMAP$(d_t, c_t, d_r, c_r)$

return
}

CONSTRUCT(c_s, T_1, T_2)
{
find $c_s \in C'$ such that $|c_s \cap (T_1 \cup T_2)| \geq 2$, $|ptr(c_s) \cap (T_1 \oplus T_2)| = 1$ and
$|ptr(c_s) \cap (T_1 \cap T_2)| = 0$
if succeed, $c_s \leftarrow c_s - ((T_1 \cup T_2) - ptr(c_s))$ and *return* "succeed"
find $c_s \in C'$ such that $|c_s \cap (T_1 \oplus T_2)| \geq 1$ and $|ptr(c_s) \cap (T_1 \cup T_2)| = 0$
if succeed, *select* $d_n \in (c_s \cap T_1 \oplus T_2)$
$c_s \leftarrow c_s - ((T_1 \cup T_2) - \{d_n\})$
$ptr(c_s) \leftarrow ptr(c_s) \cup \{d_n\}$
return "succeed"
return "fail"
}

ADD(c_s, d_s, T_1, T_2)
{
select one $d_n \in (T_1 \oplus T_2)$
add d_s
$c_s \leftarrow D - (((T_1 \cup T_2) - \{d_n\}) - Rch(d_s)) \cup \{d_s\}$
$ptr(c_s) \leftarrow \{d_n, d_s\}$
$D' \leftarrow D' \cup \{d_s\}$
$C \leftarrow C \cup \{c_s\}$
return
}

ADDP(p_m)
{
$P' \leftarrow P' \cup \{p_{M'+1}\}$
$M' \leftarrow M' + 1$
$m \leftarrow M' + 1$
$Adj(p_m) \leftarrow \emptyset$
}

As a design example of single-fault locatable ABFT system for Algorithm 6.1, we will consider a MPD graph illustrated in Fig. 6.1(a). From this MPD graph, $D_1 = \{\{d_{11}\}, \{d_{12}, d_{21}, d_{31}\}\}$, $D_2 = \{\{d_{21}\}\}$, $D_3 = \{\{d_{31}\}\}$ and $D_4 = \{\{d_{31}, d_{41}\}\}$. According to the algorithm MAP-EMPDC, we can construct checks for each pair of D_i and D_j, $1 \leq i < j \leq 4$. There are various solutions which depend on the method for choosing d_n(ADD), p_k and p_l(XMAP, YMAP, UMAP, VMAP and RMAP), and finding c_x, c_y, c_u, c_v and c_r in each "*find*" state within the subroutines XMAP, YMAP, UMAP, VMAP and RMAP, respectively. The procedure for constructing checks by Algorithm 6.1 is described as follows.

1. D_1 and D_j, $2 \leq j \leq 4$:
   $c_1 = \{\underline{d_{31}}, \underline{d_{71}}\}$: p_5
   $c_2 = \{\underline{d_{21}}, \underline{d_{61}}\}$: p_8

$c_3 = \{\underline{d_{11}}, d_{12}, \underline{d_{51}}\}$: p_6
$c_4 = \{\underline{d_{11}}, d_{12}, d_{21}, \underline{d_{41}}, \underline{d_{81}}\}$: p_9
$c_5 = \{d_{12}, \underline{d_{21}}, \underline{d_{31}}, d_{41}, \underline{d_{91}}\}$: p_4

2. D_2 and D_j, $3 \leq j \leq 4$: no updated

3. D_3 and D_j, $4 \leq j \leq 4$: no updated

Where, the underlined data elements of each check c_q, $1 \leq q \leq 5$, denote that such data elements are in $ptr(c_q)$. The result for the above execution is illustrated in Fig. 6.1(c). The line between data elements and checks denotes that the data element is in $ptr(c)$ for a check $c \in C$.

In the next section, we will describe a checking scheme for single-fault locatable FIR filter based on the EMPDC graph model.

6.4 Design of Fault Tolerant FIR Filter

High-speed FIR filtering has been attracted by many researchers in spite of less efficient than infinite impulse response(IIR) filtering in analog equivalence and cost effectiveness because it is always *stable* and can always be made to have *linear phase* response which is characteristic that makes it extremely attractive in audio, image and sonar applications. Since the probability of one or more PEs to become faulty in VLSI architectures such as systolic array is quite large, it is desirable to build some *on-line* fault tolerance features at lower cost into them.

6.4.1 FIR Filter

Let $x(n)$ and $y(n)$ be input and output, respectively. FIR filtering with filter coefficients h_i, $0 \leq i \leq N-1$, is defined as,

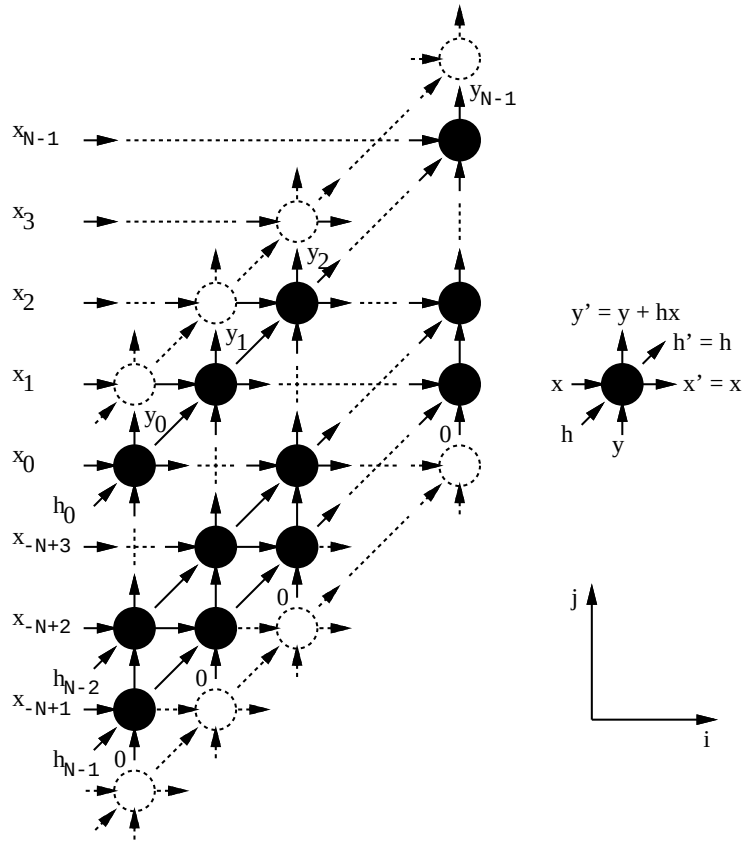
$$y(n) = \sum_{i=0}^{N-1} x(n-i)h_i \tag{6.1}$$

An important feature of FIR filtering is the iterative computation with respect to inputs $x(n)$, $n \geq 0$, to produce outputs $y(n)$, $n \geq 0$. Now, the k-th block computation $y_k(l)$, $0 \leq l \leq B-1$, with block length B, will be considered.

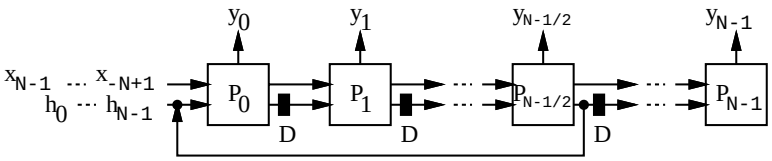
$$y_k(l) = \sum_{i=0}^{N-1} x_k(l-i)h_i \tag{6.2}$$

Where, $y_k(l) = y(kB+l)$ and $x_k(l-i) = x(kB+l-i)$. The form of *matrix-vector multiplication* for *eq.* (6.2) with $B = N$ is shown in *eq.* (6.3).

$$\begin{bmatrix} y_k(0) \\ y_k(1) \\ y_k(2) \\ \vdots \\ y_k(N-1) \end{bmatrix} = \begin{bmatrix} h_{N-1} & h_{N-2} & \cdots & h_0 & 0 & \cdots & 0 \\ 0 & h_{N-1} & \cdots & h_1 & h_0 & \cdots & 0 \\ 0 & 0 & \cdots & h_2 & h_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & h_{N-1} & h_{N-2} & \cdots & h_0 \end{bmatrix} \begin{bmatrix} x_k(-N+1) \\ x_k(-N+2) \\ \vdots \\ x_k(0) \\ \vdots \\ x_k(N-1) \end{bmatrix} \tag{6.3}$$



(a) Dependence Graph for FIR filtering.



(b) A systolic FIR filter.

Figure 6.3: Original FIR filter.

The DG for FIR filtering and a particular implementation on systolic array are illustrated in Fig. 6.3(a) and (b) with $B = N$, respectively. Each node in DG consists of one addition and one multiplication, and each PE consists of one adder, one multiplier, and three registers.

6.4.2 Fault Tolerant FIR Filter

In the following, we consider the only computational fault of each PE. We assume that the only updated ones of data elements which leave the processor may become erroneous. On the other hand, faults on communication links or registers are assumed to be treated by other techniques such as error detecting/correcting codes. Therefore, the only output string $\boldsymbol{y}$ will be considered in constructing MPD graph. The MPD graph for the FIR filter is illustrated in Fig. 6.4(a), where y_i denotes the i-th output of the k-th block.

We now consider two basic checking schemes based on the EMPDC graph model for locating single-fault in a particular architecture of FIR filter(Fig 6.3(b)).

Theorem 6.1 *For a given system which is* $Adj(p_i) = \{y_i\}$ *and* $Rch(y_i) = \{y_i\}$, $0 \le i \le N-1$, *an extended ABFT system is single-fault locatable if for each pair of data elements* y_i *and* y_j, $0 \le i < j \le N-1$, *there is a set of checks* $\{c_i, c_j, c_t\}$ *such that*

1. $1a.\ c_i = \{y_i, y_i'\}$
   $1b.\ c_j = \{y_j, y_j'\}$
   $1c.\ c_t = \{y_i, y_j, y_t'\}$
2. $2a.\ |\{\tilde{c}_i, \tilde{c}_j\} \cap (Adj(p_i) \cup Adj(p_j))| \le 1$
   $2b.\ |\{\tilde{c}_i, \tilde{c}_j\} \cap (Adj(p_i') \cup Adj(p_j'))| \le 1$
3. $3a.\ |\{\tilde{c}_i, \tilde{c}_t\} \cap (Adj(p_i) \cup Adj(p_i') \cup Adj(p_t))| = 0$
   $3b.\ |\{\tilde{c}_j, \tilde{c}_t\} \cap (Adj(p_j) \cup Adj(p_j') \cup Adj(p_t))| = 0$

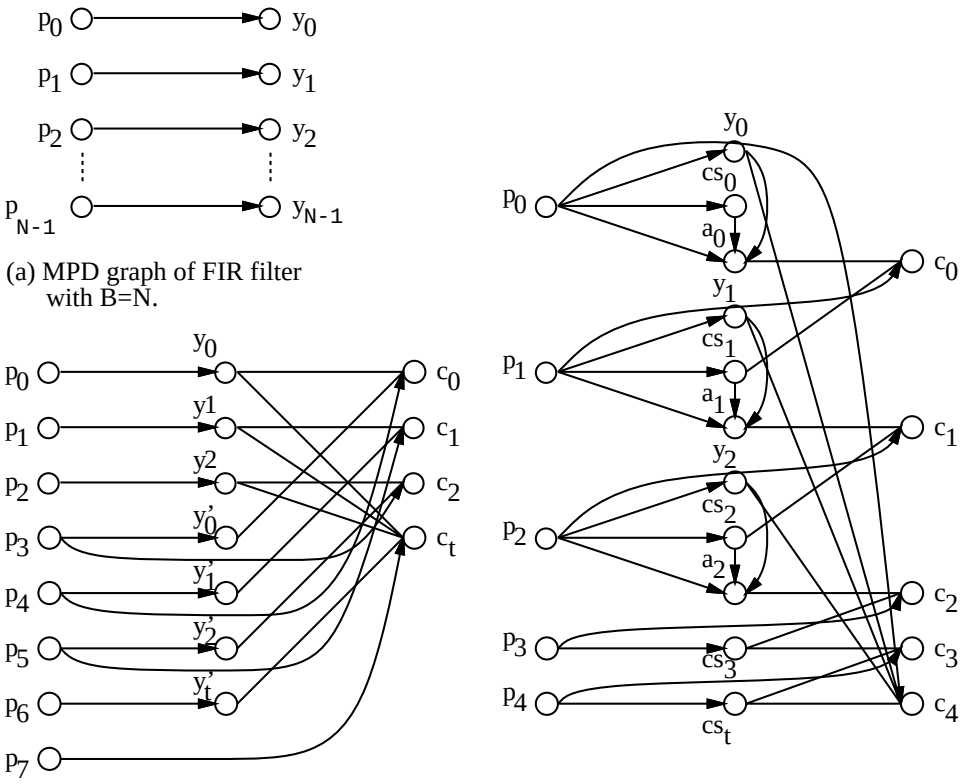
where y_*' *is a redundant data element to be compared to the primary output* y_* *or the sum of the primary outputs in* c_*, *and is computed on a redundant processor* p_*'.

Proof: It is straightforward from Lemma 6.2. There is a set of checks $\{c_x, c_y, c_u, c_v\}$ such a way that $c_x = c_i$, $c_y = c_j$, and $c_u = c_v = c_t$, for any pair of $W_{ix} = \{y_i\}$ and $W_{jy} = \{y_j\}$, $0 \le i < j \le N-1$. □

Theorem 6.2 *For a given system which is* $Adj(p_i) = \{y_i\}$ *and* $Rch(y_i) = \{y_i\}$, $0 \le i \le N-1$, *if an extended ABFT system is constructed as follows,*

1. $1a.\ Adj(p_i) = \{y_i, cs_i, a_i\}$, $0 \le i \le N-1$
   $1b.\ Adj(p_N) = \{cs_N\}$
   $1c.\ Adj(p_{N+1}) = \{cs_t\}$
2. $2a.\ Rch(y_i) = \{y_i, a_i\}$, $0 \le i \le N-1$
   $2b.\ Rch(cs_i) = \{cs_i, a_i\}$, $0 \le i \le N-1$
   $2c.\ Rch(a_i) = \{a_i\}$, $0 \le i \le N-1$
   $2d.\ Rch(cs_N) = \{cs_N\}$
   $2e.\ Rch(cs_t) = \{cs_t\}$
3. $3a.\ c_i = \{a_i, cs_{i+1}\}$, $0 \le i \le N-1$
   $3b.\ c_N = \{cs_N, cs_t\}$
   $3c.\ c_{N+1} = \{y_0, y_1, \cdots, y_{N-1}, cs_t\}$
4. $4a.\ \tilde{c}_i \in Adj(p_{i+1})$, $0 \le i \le N$
   $4b.\ \tilde{c}_{N+1} \in Adj(p_0)$

then the system is single-fault locatable.



(a) MPD graph of FIR filter with B=N.

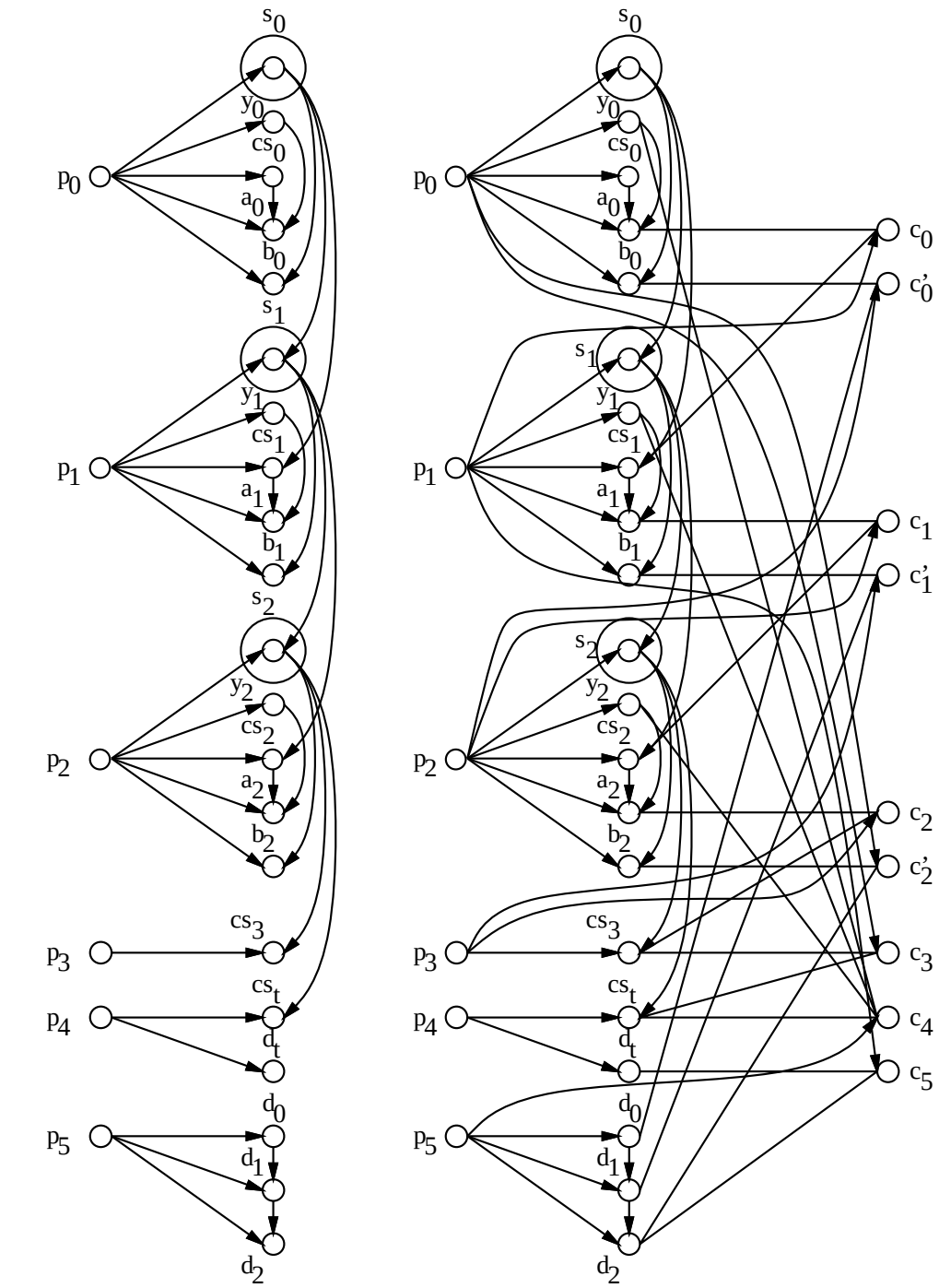
(b) An extended MPDC graph using Theorem 6.1 with N=3.

(c) An extended MPDC graph using Theorem 6.2 with N=3.

Figure 6.4: Two basic EMPDC graphs for the original FIR filter.

Proof: It is straightforward from Lemma 6.2. Note that a_i is always erroneous when p_i is faulty. Hence, a_i is included in all of the available error patterns induced by p_i. Therefore, we will show whether for each pair of $W_{ix} = \{a_i\}$ and $W_{jy} = \{a_j\}$, $0 \leq i < j \leq N-1$, a set of checks which satisfy Lemma 6.2 exists or not. There is a set of checks $\{c_x, c_y, c_u, c_v\}$ such a way that $c_x = c_j$, $c_y = c_i$, $c_u = c_{j+1}$, and $c_v = c_{i+1}$. Also, it is clear that for any pair of processors which are not considered in the above situations, there is a set of checks which are given by Lemma 6.2. Therefore, the extended ABFT system is single-fault locatable. □

For the case of $N = 3$ in the MPD graph(Fig. 6.4(a)), examples of an EMPDC graph using Theorem 1 and Theorem 2 are illustrated in Fig. 6.4(b) and (c), respectively. The redundant data elements in Fig. 6.4(b) are computed in several redundant processors, while such redundant data elements in Fig. 6.4(c) are well distributed to the primary processors. Therefore, the basic checking scheme of single-fault locatable FIR filter to be designed is based on Theorem 6.2. It may be very difficult to design MPD graph including redundant data elements without degrading the performance of the system nor increasing the number of processing components or processors. Hence, we will concentrate on minimizing the degradation of performance and the number of processing components or processors by mapping redundant data elements and checking operations to the system processors.



(a) MPD graph with redundancies. (b) An extended MPDC graph for (a).

Figure 6.5: An EMPDC graph with $N = 3$.

To efficiently compute the redundant data elements on systolic array, first we introduce updated data elements cs_i such a way that $cs_i = y_{i-1} + cs_{i-1}$ computed on p_i, $0 \leq i \leq N$. And then $a_i = y_i + cs_i$ computed on p_i, $0 \leq i \leq N-1$, is introduced to meet the scheme in Theorem 6.2. Unfortunately, a number of updated data elements $s_i(l)$, $0 \leq l \leq N-1$, to compute cs_i, are computed on p_i, $0 \leq i \leq N-1$. Therefore, all of available error patterns due to these data elements have to be additionally considered to design single-fault locatable FIR filter. To detect and locate such data elements, we introduce $b_i = \sum_{l=0}^{N-1} s_i(l)$ computed on p_i, $0 \leq i \leq N-1$, d_i computed on p_{N+2} to compare to b_i, $0 \leq i \leq N-1$, and d_t computed on p_{N+1} to compare to d_{N-1}. Furthermore, $N+1$ checks $c'_i = \{b_i, d_i\}$, $0 \leq i \leq N-1$, and $c_{N+2} = \{d_{N-1}, d_t\}$ are newly introduced. Finally, the MPD graph which is included all of these redundant computations is shown in Fig. 6.5(a).

According to the MPD graph, each of redundant data elements has the properties as follows.

$$
\begin{aligned}
a_i &= cs_{i+1} && (6.4)\\
b_i &= d_i && (6.5)\\
cs_t &= \sum_{i=0}^{N-1} y_i && (6.6)\\
cs_t &= cs_N && (6.7)\\
d_t &= d_{N-1} && (6.8)
\end{aligned}
$$

Where,

$$
\begin{aligned}
a_i &= cs_i + y_i,\\
b_i &= \sum_{l=0}^{N-1} s_i(l),\\
d_i &= d_{i-1} + (x_{(-N+1+i)} + x_{(-N+2+i)} + \cdots + x_i),\\
cs_t &= h_{N-1}(x_{(-N+1)} + x_{(-N+2)} + \cdots + x_0) +\\
&\quad h_{N-2}(x_{(-N+2)} + x_{(-N+3)} + \cdots + x_1) + \cdots +\\
&\quad h_0(x_0 + x_1 + \cdots + x_{N-1}),\\
d_t &= (x_{(-N+1)} + x_{(-N+2)} + \cdots + x_0) +\\
&\quad (x_{(-N+2)} + x_{(-N+3)} + \cdots + x_1) + \cdots +\\
&\quad (x_0 + x_1 + \cdots + x_{N-1}),\\
s_i(l) &= x_{(-l)} + x_{(-l+1)} + \cdots + x_{(i-l-1)}.
\end{aligned}
$$

From these properties, we can construct a set of checks as follows.

(1) c_i: $a_i = cs_{i+1}$? on p_{i+1}, $0 \leq i \leq N-1$

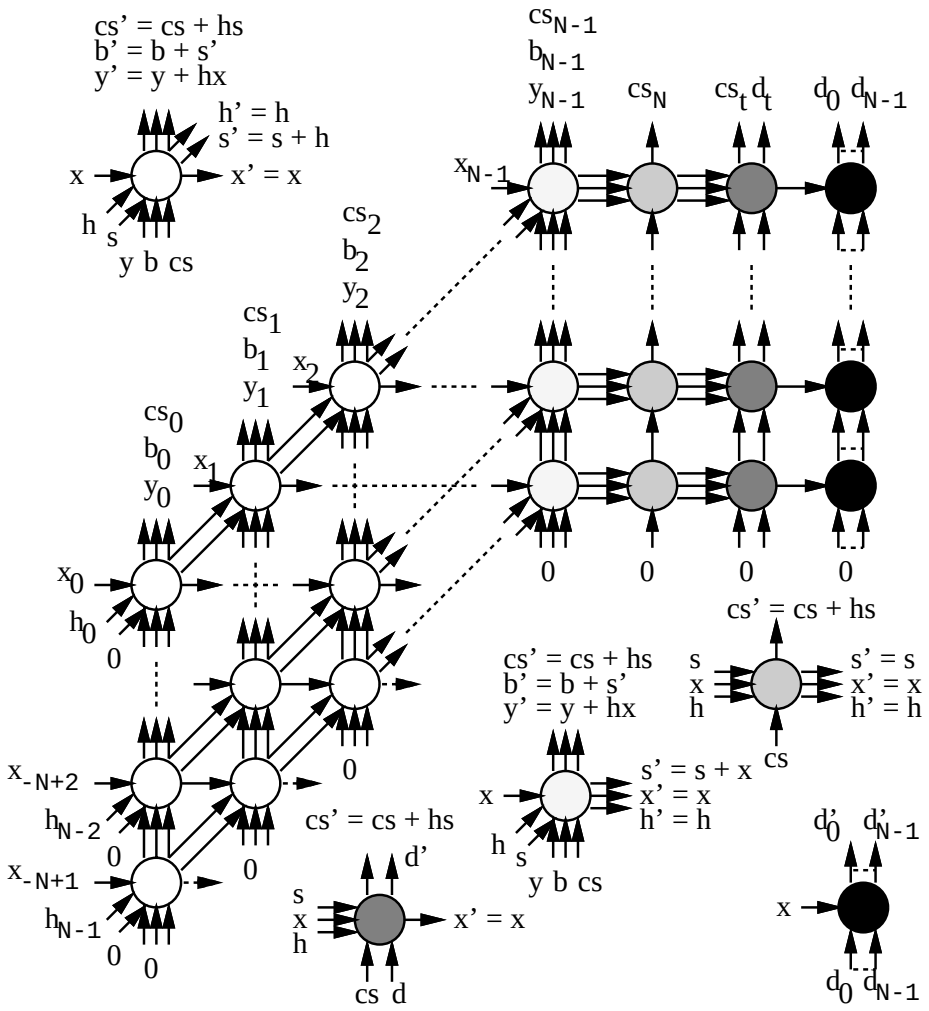
(2) c'_i: $b_i = d_i$? on p_{i+2}, $0 \leq i \leq N-2$

(3) c'_{N-1}: $b_{N-1} = d_{N-1}$? on p_0

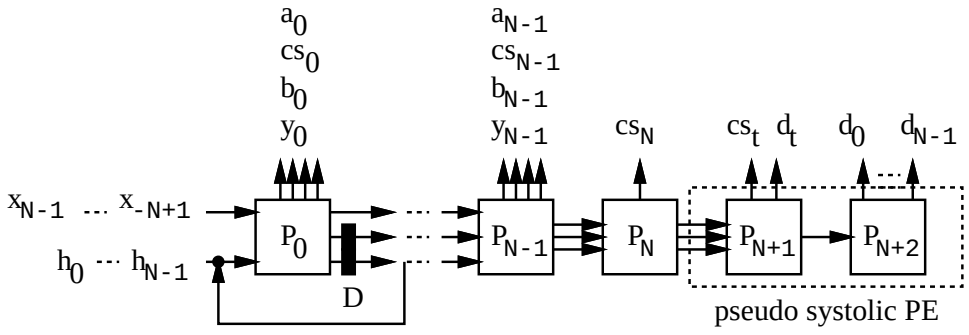
(4) c_N: $cs_N = cs_t$? on p_0

(5) c_{N+1}: $y_0 + y_1 + \cdots + y_{N-1} = cs_t$? on p_{N+2}

(6) c_{N+2}: $d_{N-1} = d_t$? on p_1



(a) Dependence graph for FIR filtering with fault tolerance.



(b) A fault tolerant FIR filter.

Figure 6.6: Single-fault locatable FIR filter implemented on systolic array.

There are totally $2N+3$ checks for locating single-fault.

Now, we will examine whether this checking scheme satisfies Lemma 6.2 or not. In the following, $\boldsymbol{s}_i$ denotes the set which consists of $s_i(l)$, $0 \leq l \leq N-1$.

Correctness of checking scheme For each pair of unions W_{ix} and W_{jy}, W_{ix} is one of all available unions of elements in $D(p_i)$, W_{jy} is one of all available unions of elements in $D(p_j)$, $0 \leq i < j \leq N-1$,

- $\mathcal{S}1$. if $|\boldsymbol{s}_i \cap W_{ix}| \geq 1$ and $|\boldsymbol{s}_j \cap W_{jy}| \geq 1$, then $c_x = c'_{j-1}$, $c_y = c'_j$, and $c_u = c_v = c_{N+2}$,
- $\mathcal{S}2$. if $|\boldsymbol{s}_i \cap W_{ix}| \geq 1$, $|\{a_j\} \cap W_{jy}| = 1$, and $|\boldsymbol{s}_j \cap W_{jy}| = 0$, then $c_x = c'_{j-1}$, $c_y = c_j$, $c_u = c_{N+2}$, and $c_v = c_{j+1}$,
- $\mathcal{S}3$. if $|\{a_i\} \cap W_{ix}| = 1$, $|\boldsymbol{s}_i \cap W_{ix}| = 0$, and $|\boldsymbol{s}_j \cap W_{jy}| \geq 1$, then $c_x = c_i$, $c_y = c_j$, $c_u = c_{i+1}$, and $c_v = c_{N+2}$,
- $\mathcal{S}4$. if $|\{a_i\} \cap W_{ix}| = 1$, $|\boldsymbol{s}_i \cap W_{ix}| = 0$, $|\{a_j\} \cap W_{jy}| = 1$, and $|\boldsymbol{s}_j \cap W_{jy}| = 0$, then $c_x = c_j$, $c_y = c_i$, $c_u = c_{j+1}$, and $c_v = c_{i+1}$,
- $\mathcal{S}5$. if $|\{b_i\} \cap W_{ix}| = 1$, $|(\boldsymbol{s}_i \cup \{a_i\}) \cap W_{ix}| = 0$, $|\{b_j\} \cap W_{jy}| = 1$, and $|(\boldsymbol{s}_j \cup \{a_j\}) \cap W_{jy}| = 0$, then $c_x = c'_i$, $c_y = c'_j$, $c_u = c_v = c_{N+2}$.

Thus, for any pair of error patterns, one is induced by p_i and the other is induced by p_j, $0 \leq i < j \leq N-1$, there are always four checks c_x, c_y, c_u, and c_v which satisfy the conditions of Lemma 6.2. And it is clear that for any pair of processors which are not considered in the above situations, there are always such four checks. Therefore, the extended ABFT system based on the proposed checking scheme is single-fault locatable. The EMPDC graph based on this checking scheme is shown in Fig. 6.5(b).

Finally, DG including all of the redundant computations is illustrated in Fig. 6.6(a), where a_i is not shown because it is computed once after y_i and cs_i are computed on p_i, $0 \leq i \leq N-1$. The FIR filter obtained by projecting the DG in j-direction is illustrated in Fig. 6.6(b). As a result, the original FIR filter(Fig. 6.3(b)) consists of N adders and N multipliers, while the fault tolerant FIR filter consists(Fig. 6.6(b)) of $5N+5$ adders, $2N+2$ multipliers, $2N+3$ comparators, and additional N registers due to the circular buffer for storing input data $\boldsymbol{x}$.

6.4.3 System Evaluation

Now, we discuss some properties for error detection and correction of the proposed fault tolerant FIR filter.

Error Detection

Several efforts have been made for CED schemes in the area of signal processing applications. Gupta and Bayoumi [11] proposed a novel CED scheme termed as logarithm based on-line error detection which is based on the use of logarithmic coding for inputs and results in a self-testing systolic cell. And Vinnakota and Jha [6] proposed a method for synthesizing single-fault detectable ABFT system from DG of FIR filtering by introducing an useful checking scheme. In general, these schemes assume that the monitoring circuit to compare two results is fault-free or has self checking properties. And the schemes require almost twice in silicon area or in time.

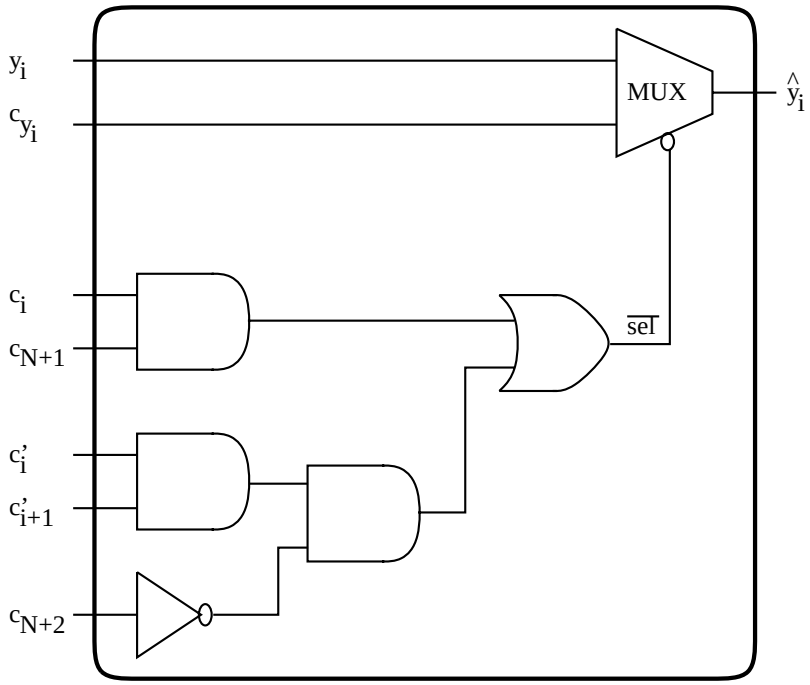


Figure 6.7: The circuit for correcting the erroneous output y_i.

On the other hand, the proposed fault tolerant FIR filter does not assume that the checking operation to compare two results is fault-free, and it can be achieved systolic array implementation without causing any loss in throughput rate. Also, this scheme can detect two-fault except for the faults included at least one checking operation, of course, all of single-fault can be detected. Unfortunately, since our main target is to locate single-fault, the scheme requires nearly three times in hardware resources.

Error Correction

Kung [10] presented error detection and correction based on interleaved DG. The idea is to perform the same computation twice in adjacent PEs at two different but close enough time periods and then compare the results. If they match there is no fault. Otherwise a roll-back is necessary to correct the fault. However, a fault in checking operation which is to compare two results: one is *primary* output computed in a PE, the other is *redundant* output computed in adjacent PE, was not considered, that is, checking operations were assumed to be fault-free. Cosentino [12] proposed a scheme of concurrent error detection and correction in systolic architecture of FIR filtering at a cost of halving the maximum throughput rate by performing the same computation twice in adjacent PEs and comparing such two results.

On the other hand, for the output y_i, $0 \leq i \leq N-1$, of FIR filtering with order N, the proposed fault tolerant FIR filter can concurrently correct the error induced by any single-fault, without retrying the computations. For any error pattern including the output y_i, $0 \leq i \leq N-1$, which is induced by the faulty processor p_i, the checks c_i and c_{N+1} always output 1, or the checks $c'_i, c'_{i+1}, \cdots, c'_{N-1}$ always output 1 and the check c_{N+2}

always outputs 0. According to the properties of the checking scheme,

$$\begin{aligned} cs_i &= y_{i-1} + cs_{i-1}, \\ cs_{i+1} &= y_i + cs_i. \end{aligned}$$

Assuming single-fault(p_i is faulty), cs_{i+1} and a_{i-1} equivalent to cs_i are error-free, which are computed on p_{i+1} and p_{i-1}, respectively. Therefore, the corrected output c_{y_i} can be obtained as follows.

$$c_{y_i} = cs_{i+1} - a_{i-1} \tag{6.9}$$

The circuit for correcting the erroneous output y_i is shown in Fig. 6.7. The correcting circuit is assumed to fault-free. Where, $\hat{y}_i = y_i$ if $\overline{sel} = 0$, and $\hat{y}_i = c_{y_i}$ if $\overline{sel} = 1$. Also, MUX denotes a multiplexer.

Fault Location

The conventional schemes has been concentrated on concurrent error detection and correction in the area of signal processing applications. There is little approach in systematically synthesizing single-fault locatable FIR filter which is implemented on systolic array. However, the fault location plays an important role in correcting errors or reconfiguring the system to bypass the faulty processor when a fault is permanent. To achieve this objective, we proposed a scheme based on the EMPDC graph model so that single-fault locatable FIR filter is systematically synthesized on systolic array.

6.5 Conclusion

In this chapter, we present a method in designing single-fault locatable FIR filter based on EMPDC graph model. The ABFT system which has been introduced in previous chapters for designing more efficient fault tolerant systems was extended by introducing some redundancies to be compared to the sum of data elements in checks and mapping such checks to the system processors such a way that the system still maintains the desired fault tolerance. As an application of our theory to a practical problem, a fault tolerant FIR filter was implemented on systolic array. As a result, while the fault tolerant FIR filter required nearly three times in hardware resources, it achieved CED capability without causing any loss in throughput rate and assuming that checking operations are fault-free. Furthermore, for any primary output of FIR filtering, the fault tolerant FIR filter could be applied to *on-line* error correction without retrying the computation by introducing the correcting circuit being fault-free or being of the self-checking property.

Chapter 7

Conclusions

In this thesis, we present the analysis and synthesis of ABFT systems based on MPD graph model, and proposed checking schemes for single-fault detection and location on the simple error model(SID model) and the sophisticated error model(MID model). Also, we present two basic algorithms SFL-TFD I and SFL-TFD II for constructing checks of SFL/TFD ABFT systems for SID model and MID model, respectively. As a result, the set of error patterns to be considered with MID model becomes a superset of the one with SID model, but still a subset of the one from the conventional PDC graph models. In general, the computational complexity and the number of checks in analyzing and designing ABFT systems increase as the number of error patterns increases. In this sense, SID model based analysis is preferable among three analysis models. However, it assumes that erroneous inputs yield always erroneous output, and error masking by plural erroneous inputs or by faulty computation using an erroneous input is not treated. In contrast with SID model based analysis, MID model based analysis can treat such error masking phenomenon. While the complexity and the number of checks tend to increase compared to the MPD graph with SID model in compensation for improving the accuracy of error propagation model, the effectiveness of MPD graph model with MID model holds good compared to the conventional PDC graph models.

Our approach will be applied to linear algebra based computation algorithms without feedback loops, and both the number of error patterns to be considered and the number of checks are reduced compared with the ones for conventional PDC graph models. Furthermore, there exist some ABFT systems which are single-fault locatable from our MPD graph model but not from the conventional PDC graph models, because our MPD graph with the appropriate error propagation model can exclude redundant error patterns which are included for considerations in the analysis by the conventional PDC graph models. Analysis and checking schemes for cyclic MPD graph for applying our method to computation algorithms with feedback loops, algorithms for designing minimum number of checks for a given MPD graph and the design of mapping from a set of operations to a set of processors with regarding checking scheme are remained as future problems.

In this thesis, we also extended the MPD graph model based ABFT system by introducing some redundancies to be compared to the sum of data elements in checks and mapping these checks to the system processors such a way that the system still maintains the designated fault tolerance. The proposed EMPDC graph model includes the relationships between checks and processors for computing checks, and plays important role in designing cost-effective ABFT systems in the sense that the number of checks and

redundant computations are reduced by considering data dependency between primary data elements and redundant data elements. Also we proposed a method in designing single-fault locatable FIR filter based on EMPDC graph model under SID error model, and a fault tolerant FIR filter was implemented on systolic array. The scheme will be used as a basis of systematic synthesis for locating faults in signal processing applications.

Throughout this research, we focused our attention on two important features in analyzing and designing cost-effective ABFT systems, *i.e.*, (1) detection of errors due to a fault at some processor output, and (2) location of the faulty processor, by introducing a specified error occurrence/propagation model using data dependency between computation results. The results obtained in this thesis will provide important bases for highly reliable parallel computing systems.

Bibliography

[1] K.H. Huang and J.A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations",*IEEE Trans. on computers*, Vol.33, pp.518-528, June 1984.

[2] P. Banerjee and J.A. Abraham, "Concurrent Fault Diagnosis in Multiple Processor Systems", *Proc. 16th Symp. Fault Tolerant Comput.*, pp.298-303, 1986.

[3] R. Sitaraman and N.K. Jha, "Optimal design of checks for error detection and location in fault tolerant multiprocessor systems", in *Proc. Int. Symp. Fault Tolerant Comput.*, pp.396-406, 1992.

[4] V.S.S. Nair, J.A. Abraham, and P. Banerjee, "Efficient techniques for the analysis of ABFT schemes", *IEEE Trans. on computers*, Vol.45, No.4, April 1996.

[5] P. Banerjee and J.A. Abraham, "Bounds on ABFT in multiple processor systems", *IEEE Trans. on computers*, Vol.35, No.4, April 1986.

[6] B. Vinnakota and N.K. Jha, "Synthesis of ABFT systems from dependence graphs", *IEEE Trans. on Parallel and Distributed Systems*, Vol.4, No.8, August 1993.

[7] S. Yajnik and N.K. Jha, "Graceful degradation in ABFT multiprocessor systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol.8, No.2, February 1997.

[8] M. Kaneko and H.Miyauchi, "Fault Tolerant Non-regular Digital Signal Processing Based on Computation Tree Block Decomposition", *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E77-A, No.9, pp.1535-1545, September 1994.

[9] M. Kaneko and H.Miyauchi, "A Systematic Design of Fault Tolerant Systolic Arrays Based on Triple Modular Redundancy in Time-Processor Space", *IEICE Trans. on Inf. and Syst.*, Vol.E79-D, No.12, pp.1676-1689, December 1996.

[10] S.Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.

[11] S.R. Gupta and M.A. Bayoumi, "Concurrent Error Detection in Systolic Arrays for Real-Time DSP Applications" VLSI Signal Processing III.

[12] R.J. Cosentino, "Concurrent Error Detection in Systolic Architectures", *IEEE Trans. on Computer-Aided Design*, Vol.7, No.1, pp.117-125, January 1988.

[13] H.T. Kung, "Why Systolic Architectures?", *IEEE Computer*, pp.37-46, January 1982.

[14] J.A. Abraham, P. Banerjee, C.-Y. Chen, W.K. Fuchs, S.-Y. Kuo, and A.L.N. Reddy, "Fault Tolerance Techniques for Systolic Arrays", *IEEE Computer*, pp.65-75, July 1987.

[15] S.-W. Chan and C.-L. Wey, "The Design of Concurrent Error Diagnosable Systolic Arrays for Band Matrix Multiplications", *IEEE Trans. on Computer-Aided Design*, Vol.7, No.1, pp.21-37, January 1988.

[16] M. Renfors and Y. Neuvo, "The Maximum Sampling Rate of Digital Filters under Hardware Speed Constraints", *IEEE Trans. on Circuits and Systems*, Vol.CAS-28, No.3, pp.196-202, March 1981.

[17] J.-Y. Jou and J.A. Abraham, "Fault-Tolerant FFT Networks", *IEEE Trans. on Computers* Vol.37, No.5, pp.548-561, May 1988.

[18] Y.-H. Choi and M. Malek, "A Fault-Tolerant FFT Processor", *IEEE Trans. on Computers*, Vol.37, No.5, pp.617-621, May 1988.

[19] A.L.N. Reddy and P. Banerjee, "Algorithm-Based Fault Detection for Signal Processing Applications", *IEEE Trans. on Computers*, Vol.39, No.10, pp.1304-1308, October 1990.

[20] S.-J. Wang and N.K. Jha, "Algorithm-Based Fault Tolerance for FFT Networks", *IEEE Trans. on Computers*, Vol.43, No.7, pp.849-854, July 1994.

[21] P. Banerjee, J.T. Rahmeh, C. Stunkel, V.S. Nair, K. Roy, V. Balasubramanian and J.A. Abraham, "Algorithm-Based Fault Tolerance on a Hypercube Multiprocessor", *IEEE Trans. on Computers*, Vol.39, No.9, pp.1132-1145, September 1990.

[22] S.-Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors", *Proceedings of The IEEE*, Vol.72, No.7, pp.867-884, July 1984.

[23] C.-M. Liu and C.-W. Jen, "Design of algorithm-based fault-tolerant VLSI array processor", *IEE Proceedings*, Vol.136, No.6, pp.539-547, November 1989.

[24] S. Dutt abd J.P. Hayes, "Some Practical Issues in the Design of Fault-Tolerant Multiprocessors", *IEEE Trans. on Computers*, Vol.41, No.5, pp.588-598, May 1992.

[25] V. Balasubramanian and P. Banerjee, "Compiler-Assisted Synthesis of Algorithm-Based Checking in Multiprocessors", *IEEE Trans. on Computers*, Vol.39, No.4, pp.436-446, April 1990.

[26] V.S.S. Nair and J.A. Abraham, "Real-Number Codes for Fault-Tolerant Matrix Operations on Processor Arrays", *IEEE Trans. on Computers*, Vol.39, No.4, pp.426-435, April 1990.

[27] A. Orailoglu and R. Karri, "Coactive Scheduling and Checkpoint Determination During High Level Synthesis of Self-Recovering Microarchitectures", *IEEE Trans. on VLSI Systems*, Vol.2, No.3, pp.304-311, September 1994.

[28] A. Orailoglu and R. Karri, "Automatic Synthesis of Self-Recovering VLSI Systems", *IEEE Trans. on Computers*, Vol.45, No.2, pp.131-142, February 1996.

[29] J.-L. Sung and G.R. Redinbo, "Algorithm-Based Fault Tolerant Synthesis for Linear Operations", *IEEE Trans. on Computers*, Vol.45, No.4, pp.425-438, April 1996.

[30] V.S.S. Nair, Y.V. Hoskote and J.A. Abraham, "Probabilistic Evaluation of On-Line Checks in Fault-Tolerant Multiprocessor Systems", *IEEE Trans. on Computers*, Vol.41, No.5, pp.532-541, May 1992.

[31] D.M. Blough and A. Pelc, "Almost Certain Fault Diagnosis Through Algorithm-Based Fault Tolerance", *IEEE Trans. on Parallel and Distributed Systems*, Vol.5, No.5, pp.532-539, May 1994.

[32] S. Dutt and F.T. Assaad, "Mantissa-Preserving Operations and Robust Algorithm-Based Fault Tolerance for Matrix Computations", *IEEE Trans. on Computers*, Vol.45, No.4, pp.408-424, April 1996.

[33] D.L. Tao, C.R.P. Hartmann and Y.S. Han, "New Encoding/Decoding Methods for Designing Fault-Tolerant Matrix Operations", *IEEE Trans. on Parallel and Distributed Systems*, Vol.7, No.9, pp.931-938, September 1996.

[34] A. Roy-Chowdhury and P. Banerjee, "Algorithm-Based Fault Location and Recovery for Matrix Computations on Multiprocessor Systems", *IEEE Trans. on Computers*, Vol.45, No.11, pp.1239-1247, November 1996.

[35] H. Kim and K.G. Shin, "Design and Analysis of an Optimal Instruction-Retry Policy for TMR Controller Computers", *IEEE Trans. on Computers*, Vol.45, No.11, pp.1217-1225, November 1996.

[36] A. Roy-Chowdhury, N. Bellas and P. Banerjee, "Algorithm-Based Error-Detection Schemes for Iterative Solution of Partial Differential Equations", *IEEE Trans. on Computers*, Vol.45, No.4, pp.394-407, April 1996.

[37] C. Gong, R. Melhem and R. Gupta, "Loop Transformations for Fault Detection in Regular Loops on Massively Parallel Systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol.7, No.12, pp.1238-1249, December 1996.

[38] B.W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, 1989.

[39] A. Chatterjee, "Concurrent Error Detection and Fault-Tolerance in Linear Analog Circuits Using Continuous Checksums", *IEEE VLSI Systems*, Vol.1, No.2, pp.138-150, June 1993.

[40] M.T. O'Keefe, J.A.B. Fortes and B.W. Wah, "On the Relationship Between Two Systolic Array Design Methodologies", *IEEE Trans. on Computers*, Vol.41, No.12, pp.1589-1593, December 1992.

[41] Y. Tamir and M. Tremblay, "High-Performance Fault-Tolerant VLSI Systems Using Micro Rollback", *IEEE Trans. on Computers*, Vol.39, No.4, pp.548-554, April 1990.

[42] V.P. Roychowdhury, J. Bruck and T. Kailath, "Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays", *IEEE Trans. on Computers*, Vol.39, No.4, pp.480-489, April 1990.

Publications

[1] C. Park and M. Kaneko, "An Efficient Scheme Based on EMPDC Graph Model in Synthesizing Fault Tolerant FIR Filter", *ISCAS 2000*, May 2000. (to appear)

[2] C. Park and M. Kaneko, "Checking Scheme for ABFT Systems Based on Modified PD Graph under an Error Generation/Propagation Model", *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, VOL.E82-A, No.6, June 1999.

[3] C. Park and M. Kaneko, "An Efficient Technique for Design of ABFT Systems Based on Modified PD Graph", *Third International Conference on Massively Parallel Computing Systems*, April 6-9, 1998.

[4] C. Park and M. Kaneko, "Checking Scheme for ABFT Systems Based on Modified PD Graph under an Error Generation/Propagation Model", *Proc. The 1998 International Technical Conference on Circuits/Systems, Computers and Communications*, VOL.II, pp.1703-1706, 1998.