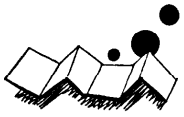
解　説

フォールトトレラント論理回路の設計技法†



南　谷　　崇††

1. はじめに

フォールトトレラント (Fault-tolerant) システム[1),2)]に関連する論理回路レベルの冗長構成技術は，その目的が誤り訂正か誤り検出かによって，次の二つに分けられる．

i) 故障マスク (Fault-masking)

ii) 自己検査 (Self-checking)

i) は，回路内部に故障が生じても外部へは常に正しく出力するように構成するもので，マスク冗長系の基本技術である．ii) は，内部に故障が生じると必ず外部へ知らせるように構成するもので，待機冗長系の基本技術である．

本稿では，i), ii) に関する論理回路設計技法を系統的に整理してみる．なお，i), ii) では故障が生じたとき，出力が“正しいか否か”を価値尺度とするのに対して，別に“安全か否か”を尺度とするフェイルセイフ技術[3)]があるが，ここでは割愛する．本稿に関連する良い教科書として文献 4)～7) がある．

2. 故障マスク技術

基本的には金物量の冗長性を利用し，①多重化による方法と，②誤り訂正符号[8)]を利用する方法，に分けられる．①は考え方が簡単で部品レベルからシステムレベルまで各段階に適用できるため，古くから実用化されている[1)]．一般に，信頼性だけを比較すると②より①の方が良いが，信頼性／金物量の比較では，多数決素子復号可能な符号[8)]を用いる場合に，①より②の方が良くなるという検討結果がある[9)]．

2.1 多重化による方法

1956年に発表された二つの論文[10),11)]が要素の多重化による信頼性の向上という概念をもたらし，現在までの計算機の高信頼化設計に大きな影響を与えている．

その第一は Moore と Shannon[10)] による冗長化リレーのアイデアである．**図-1** に5個のリレーによって単一リレーの機能を実現する例を示す．正常ならば，各リレーは閉じるべき時に一斉に閉じ，開くべき時に一斉に開くとする．個々のリレーがある時刻に閉じている確率を一様に p とすると，全体が閉じている確率は，

$$h(p)=2p^2(1-p)^3+8p^3(1-p)^2+5p^4(1-p)+p^5$$
$$=2p^2+2p^3-5p^4+2p^5$$

となる．この式から，① $p>1/2$ ならば $h(p)>p$，② $p<1/2$ ならば $h(p)<p$ である．しかるに，個々のリレーの信頼度が 1/2 以上であれば，本来閉じているべき時は $p>1/2$，開いているべき時は $p<1/2$ である．したがって，①，②より，全体の信頼度は個々のリレーより必ず向上する．

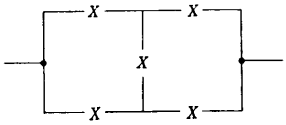
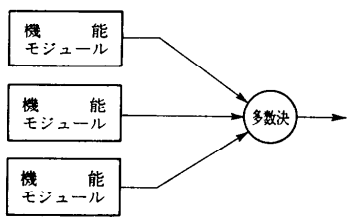
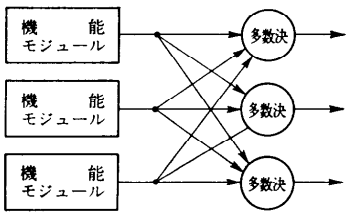


図-1 冗長化リレー



(a)



(b)

図-2 Triple Modular Redundancy

† Design Techniques for Fault-Tolerant Logic Circuitry by Takashi NANYA (Department of Computer Science, Tokyo Institute of Technology).

†† 東京工業大学工学部情報工学科

第二は，Von Neumann[11] の “restoring organ” の概念である．同一の論理信号を多重化された信号線で伝え，その冗長性を利用して信号値の誤りを修復する機構であり，この考えに基づくものとして，**図-2**(a) に示す TMR (Triple Modular Redundancy) がある[12]．機能モジュールを三重化し，“restoring organ” として多数決素子を用いれば，単一モジュールの故障はマスクされる．もし多数決素子が理想的ならば，モジュールの信頼度を R_m とした時，TMR 出力の信頼度 R は

$$R=R_m^3+3(1-R_m)R_m^2$$
$$=3R_m^2-2R_m^3$$

となる．したがって，$R_m>1/2$ ならば $R>R_m$ となり，単体より TMR の方が信頼度は向上する．多数決素子が理想的でない場合の TMR 構成を**図-2**(b) に示す．この場合の信頼度の改善効果は，多数決素子の信頼度とモジュールの信頼度の関係で定まる[13]．

“restoring organ” は誤り訂正機能を持つだけで，それ自体は論理機能を持たない．これに対して，両機能を同時に果す論理回路構成法として，4重化論理 (Quadded logic)[14] と編合せ論理 (Interwoven logic)[5] がある．

4重化論理は，AND と OR を交互に配置した論理回路において各素子及び信号線を 4 重化し，**図-3** のように相互接続したものである．これによって，4重化信号 a_1～a_4 の単一誤りは素子二段通過した 4 重化信号 b_1～b_4 において訂正され，所定の論理機能を果す．

編合せ論理は 4 重化論理を一般化したもので，多重化信号線を適当に編み合わせた素子間相互接続によって誤り訂正論理を実現する．しきい値 θ を持つ n 入力しきい値素子で構成された任意の論理回路を考えよう．各しきい値素子の代わりに，しきい値 $\theta_k=k\theta-t$ を持つ kn 入力の冗長化しきい値素子を用いると，入力の t 重誤りは冗長化しきい値素子一段で訂正できる．ただし，t は $k/2$ より小さい最大整数である．たとえば，**図-4**(a) の回路は冗長化しきい値素子を用いると同図(b) になり，すべての単一誤りは素子一段で訂正される．同様に，冗長化 NAND (NOR) 素子による誤り訂正論理も可能である[5]．

“restoring organ” は 4 重化論理や編合せ論理の特別な場合として実現できる，たとえば，図-3 はそれ自体で “restoring organ” である．また，TMR における多数決素子は，冗長化しきい値素子の特別の場合である．

2.2 誤り訂正符号を利用する方法

誤り訂正符号の計算機への適用は記憶装置を除けばきわめて少ない[15]．一つの大きな理由は，計算機内で生じるすべてのデータ変換操作に対して保存される符号が存在しないことによる．このため，ディジタル回路の信頼性向上に誤り訂正符号を利用するという一般的概念[16],[17] が示されて以来，主として，①順序回路・カウンタの状態割当への適用[18]と，②算術符号による演算回路の誤り訂正[19]が研究されているが，実用化された例は少ない[20]．これらの話題は，本特集で別に述べられているのでここでは省略する．

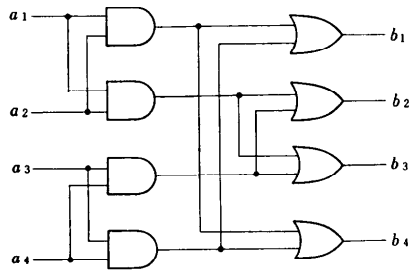
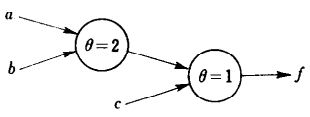
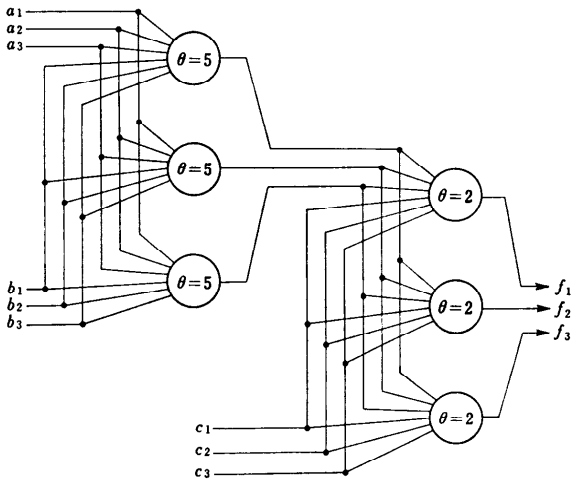


図-3　4 重化論理



(a)



(b)

図-4　しきい値素子による編合せ論理 ($f=ab+c$)

3. 自己検査性設計

3.1 自己検査性の概念

システムに故障が生じた場合，診断，スペア切換，再構成，プログラム再走等，待機冗長系における一連の自己修復動作の出発点は動作中の論理回路レベルの故障検出である．このため，各論理機能ブロックの出力に誤り検出符号を用い，通常動作中に故障が生じると，一定の動作サイクル内にそれを自動的に検出するというのが自己検査性の概念である[21]．検出対象とする故障の集合を F，通常動作サイクル中に回路へ加えられる入力の集合を N，回路の出力符号空間を S，故障 $f\in F$ がある時入力 $i\in N$ に対する出力を $G(i, f)$，故障がない時の出力を $G(i, \lambda)$ で表わすと，自己検査性回路は形式的に次のように定義される[22]．

〔定義 1〕 任意の $f\in F$ と任意の $i\in N$ に対して，$G(i, f)=G(i, \lambda)$ または $G(i, f)\notin S$ である時，回路は fault-secure (以下 FS と記す) であるという．

〔定義 2〕 任意の $f\in F$ に対してある $i\in N$ が存在して，$G(i, f)\notin S$ である時，回路は self-testing (以下 ST と記す) であるという．

〔定義 3〕 回路が FS かつ ST である時，totally self-checking (以下 TSC と記す) であるという．

すなわち，TSC 回路では，故障が生じても誤った符号語を出力せず (FS 性)，通常の動作サイクル中に非符号語出力によって故障が検出される (ST 性)．TSC 回路が有効であるためには，動作サイクルが平均故障間隔 (MTBF) に比べて小さいことが前提であることに注意を要する．さもないと，第一の故障が未検出のまま第二の故障が発生し，それによって TSC 性が失われる可能性があるからである．したがって，故障検出までの潜伏期間を評価する必要がある[23]．

実際にシステムの診断・修復機能を起動させるには，TSC 回路の出力を監視して「正常か故障か」の判定出力を出す検査回路が必要であるが，この検査回路自身も TSC でなければならない．正常時に検査回路自身の故障も検出するには判定出力は二線信号 (c_1, c_0) で表わされ，通常動作時に c_1 と c_0 はともに 0 と 1 の両方の値をとる必要がある[21]．

結局，TSC 回路は，**図-5** のように，TSC 機能回路と TSC 検査回路で構成される．TSC 検査回路の出力 (c_1, c_0) は，一方向性多重誤りを考慮して，**表-1** のように定められる．ただし，機能回路と検査回路が同時に故障することはないと仮定している．

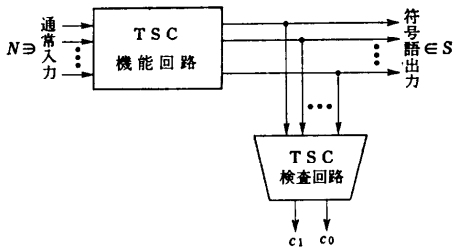


図-5 TSC 回路の構成

表-1 TSC 検査回路の出力 (c_1, c_0)

機能回路出力＼検査回路	正　常	故　障
符　号　語	(0 1), (1 0)	(0 0), (1 1)
非 符 号 語	(0 0), (1 1)	不　定

3.2 誤り検出符号

TSC 機能回路の出力符号空間 S (=TSC 検査回路への通常入力集合) として用いられる主な誤り検出符号には以下のものがある．

i) パリティ検査符号 (Parity check code)[8),16)]

機能回路がビットスライス構成の場合は 1 ビットパリティ符号，バイトスライス構成の場合は b-隣接符号[24)]が用いられる．いずれも情報部と検査部の分離した組織符号 (Systematic code)[8)] である．

ii) 剰余符号 (Residue code)[19)]

情報部の表わす整数 X を整数 A で割った剰余 $|X|_A$ を検査部とする組織符号 $[X, |X|_A]$ を剰余符号という．A を適当に選ぶと剰余符号は算術演算に対して保存されるので，機能回路が算術回路の場合に適している．特に $A=2^b-1$ の場合には $|X|_A$ を modulo (2^b-1) 加算で容易に生成できるので低コスト (Low cost) 剰余符号[25)]と呼ばれる．また，検査部に剰余の補数を用いた符号 $[X, \overline{|X|_A}]$ は逆 (inverse) 剰余符号と呼ばれ，繰返し演算 (たとえば乗算) 実行中の故障検出に有用である[25)]．

iii) M-out-of-N 符号[7)]

符号語長 N ビットの内，値 1 をとるビットが M 個である符号を M-out-of-N 符号という．任意の二符号語間に大小関係がない (この性質を持つ符号を非順序符号という) ので，すべての一方向性多重誤りを検出できる．組織符号ではないので，データ系には不便であるが，後で述べるように，制御系，特に順序回路に適している．なお，表-1 で定義した TSC 検査回路の出力 (c_1, c_0) は 1-out-of-2 符号である．

iv) バーガー符号 (Berger code)[26),27)]

kビットの情報部に含まれる0の個数の2進表示を$\log_2(k+1)$ ビットの検査部とした組織符号である．M-out-of-N 符号と同様に非順序符号である．

v) チェックサム符号 (Check sum code)[28)]

バイト長bビットのnバイトを情報部とし，n個のバイトの modulo 2^b 加算の総和を検査部とした (n+1) バイト組織符号である．b-隣接符号と同様，バイトスライス構成に対する1バイト誤り検出に用いられる．

3.3 TSC 検査回路の構成法

用いられる誤り検出符号によって TSC 検査回路の構成法は異なる．

i) 1ビットパリティ符号検査回路

図-6 に，9ビットの奇数パリティ符号の TSC 検査回路の構成[21)]を示す．一般に，奇数パリティの場合は入力ビットを2グループに分けて，各々のパリティを Ex-OR トリーで生成すると出力は 1-out-of-2 符号になる．偶数パリティの場合は，片方のグループの出力の否定をとればよい．

ii) 二線式符号 (Two-rail code) 検査回路

n組の対の各々が 1-out-of-2 符号をなしている $2n$ ビット符号を二線式符号という．n=2 の場合に対する TSC 検査回路の構成[21)]を **図-7**(a) に示す．これは同図(b) のカルノー図で Morphic 論理[29)]を実現したものである．この検査回路への入力 ($a_0b_0a_1b_1$) の符号空間 s_2 は

$$s_2 = \{(0101), (0110), (1001), (1010)\}$$

である．この4個の符号語が通常入力として加えられれば，検査回路中のすべての一方向性多重故障に対して TSC であることは容易に確かめられる．

一般のnに対する TSC 検査回路は，図-7(a) の回路を基本ブロックとしたトリー接続で **図-8** のように構成できる．各ブロックが TSC であることから，全体が FS であることは明らかである．一方，全体が ST であるためには上記 s_2 の4個の符号語が各ブロックに加えられねばならない．しかるに，$2n$ ビット

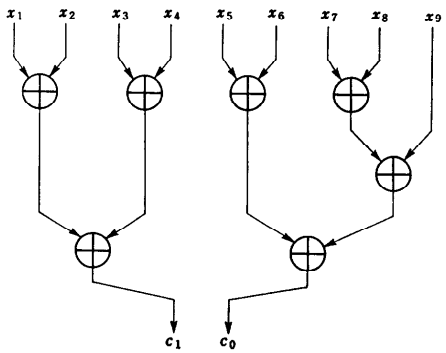
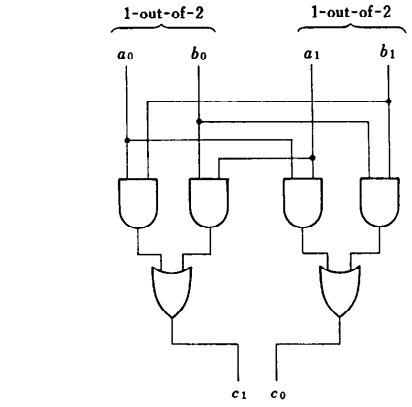
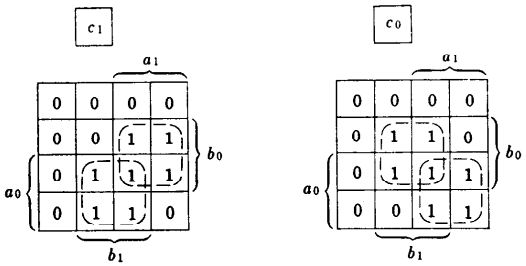


図-6 単一ビットパリティ符号の TSC 検査回路



(a)



(b)

図-7 二線式符号の TSC 検査回路 (n=2)

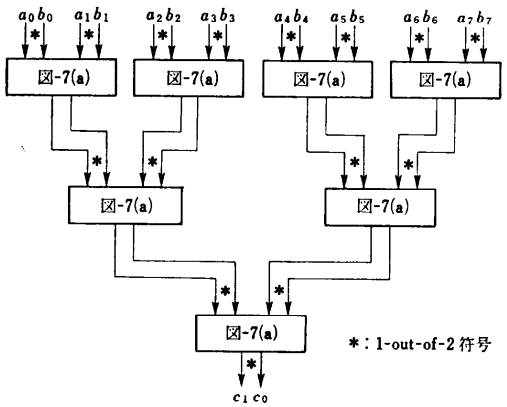


図-8 二線式符号の TSC 検査回路 (n=8)
(比較回路)

の二線式符号 s_n の符号語数は 2^n 個であるが，s_n の中のわずか4個の符号語を適当に選べば各ブロックへ s_2 が加わるようにすることができる[7]．したがって，図-8 は TSC である．

二つの n ビットベクトル $(a_1a_2\cdots a_n)$ と $(b_1b_2\cdots b_n)$ が等しい時のみ，$2n$ ビットベクトル $(a_1\bar{b}_1a_2\bar{b}_2\cdots a_n\bar{b}_n)$ は二線式符号の符号語になる．したがって，図-8 の二線式符号検査回路は，二重化信号に対する TSC 比較回路として用いられる．

iii) M-out-of-N 符号検査回路

M と N の組合せに対して種々の構成法があり得るが，しきい値関数を基礎とするものが主流である[21),22),30),31)]．たとえば，k-out-of-$2k$ の場合[22)]，$2k$ ビットの入力線を k ビットずつ2グループ A, B に分け，任意の入力パターンに対して A と B に生じる1の個数をそれぞれ k_A と k_B で表わす．不等式 $X \geqq Y$ が成立する時のみ値1をとるしきい値関数を $T(X \geqq Y)$ で表わすと，TSC の検査回路 (c_1, c_0) は次式で得られる．

$$c_1 = \sum_{i=0}^{k} T(k_A \geqq i) \cdot T(k_B \geqq k-i) \qquad i: \text{奇数}$$

$$c_0 = \sum_{i=0}^{k} T(k_A \geqq i) \cdot T(k_B \geqq k-i) \qquad i: \text{偶数}$$

具体的に，たとえば 3-out-of-6 の場合，入力線 $(a_1a_2a_3b_1b_2b_3)$ を $A=\{a_1, a_2, a_3\}$ と $B=\{b_1, b_2, b_3\}$ に分け，

$$c_1 = T(k_A \geqq 1) \cdot T(k_B \geqq 2) + T(k_A \geqq 3) \cdot T(k_B \geqq 0)$$
$$c_0 = T(k_A \geqq 0) \cdot T(k_B \geqq 3) + T(k_A \geqq 2) \cdot T(k_A \geqq 1)$$

となる．しきい値関数を積和形式で実現すると，

$$c_1 = (a_1+a_2+a_3) \cdot (b_1b_2+b_1b_3+b_2b_3) + a_1a_2a_3$$
$$c_0 = b_1b_2b_3 + (a_1a_2+a_1a_3+a_2a_3) \cdot (b_1+b_2+b_3)$$

これを AND, OR 素子で実現すればよい．

iv) 組織符号検査回路

誤り検出符号として組織符号を用いる場合，**図-9** に示すように，TSC 検査回路は検査部生成回路と図-8 の TSC 比較回路で構成される[7]．この場合，①検査部生成回路の故障をテストするのに十分な入力組合せが情報部 X に現れることと，②比較回路をテストするのに十分な入力組合せが $\{C(X), C'(X)\}$ に現れること，が前提である．組織符号の k ビットの情報部にはすべての組合せ $(2^k$ 個$)$ が現れ得ることと，前述のように 図-8 の比較回路のテストに要する入力組合せ数はわずか4個であることから，通常，①，②は容易に成立すると考えられる[7]．①と②さえ満たされれば，

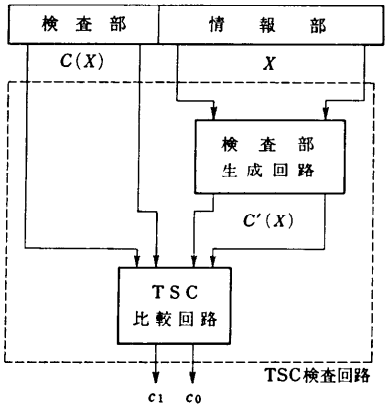


図-9 組織符号の TSC 検査回路

表-2 各種機能回路に対する符号と検査回路

機能回路			符号	検査回路
データ系	非変換系	データパス，メモリ	組織符号	図-9
	変換系	算術演算	剰余符号	図-9
			他の組織符号	図-10
		論理演算	組織符号	図-10
			二線式符号	図-8
制御系		組合せ回路	組織符号	図-9
			非順序符号	図-5
		順序回路	非順序符号	図-5

検査部生成回路は TSC である必要がない．したがって，組織符号に対する TSC 検査回路の構成問題は，多くの場合，単なる検査部生成回路の構成問題に帰着する．

3.4 各種機能回路の自己検査性設計

機能回路の種類とそれに適した符号及び検査回路を**表-2** に示す．与えられた故障集合に対して機能回路が TSC か否かは回路構造にも依存する．

i) データ非変換系

データパスやメモリでは，データは変換操作を受けないので，任意の符号は保存される．したがって，ビットスライス構成ならば1ビットパリティ符号，バイトスライス構成ならば b-隣接符号またはチェックサム符号を用いれば，単一スライス誤りに対して TSC になる．TSC 検査回路は 図-9 の構成でよい．ここで検査部生成回路は，パリティ検査符号の場合には Ex-OR の木構造で，チェックサム符号の場合は modulo 2^b 加算器の木構造で構成される[28)]．

ii) データ変換系

算術演算回路と論理演算回路に分けられる．

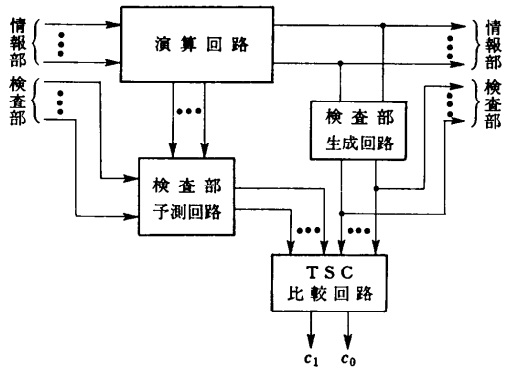


図-10 組織符号の検査部予測

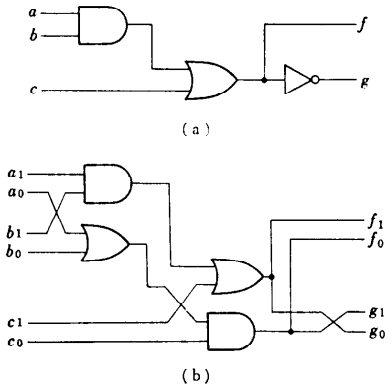


図-11 二線式論理

算術演算には剰余符号が適している．たとえば，情報部 n ビット，検査部 b ビットの低コスト剰余符号 $[X, |X|_A]$ (ただし $A=2^b-1$) を用いると，b が n の因数であれば，加減算及び巡回操作に対して保存されるので[19] 図-9 の TSC 検査回路が適用できる．この場合，検査部生成回路は modulo (2^b-1) 加算器の木構造で構成される[25]．

剰余符号（一般には算術符号[19]）以外の符号は算術演算に対して保存されないので，図-9の検査回路は無効である．しかし，組織符号の場合，**図-10** に示すように，入力の検査部と演算回路内で生ずる信号を用いて出力の検査部を予測し，それを出力の情報部から生成した検査部と比較することによって誤り検出が可能になる．たとえば1ビットパリティ符号を用いた加算の場合[6]，入力符号語を $(a_1a_2\cdots a_n, p_a), (b_1b_2\cdots b_n, p_b)$ とする．ここで，

$$p_a=\sum_{i=1}^{n} a_i,\quad p_b=\sum_{i=1}^{n} b_i \qquad \Sigma: \text{modulo 2 加算}$$

である．このとき，出力 $(s_1s_2\cdots s_n)$ の予測パリティ p_s は，

$$p_s=\sum_{i=1}^{n} s_i=\sum_{i=1}^{n}(a_i\oplus b_i\oplus c_{i-1})=p_a\oplus p_b\oplus p_c$$

ただし，$p_c=\sum_{i=0}^{n-1} c_i$

上式より，$p_a\oplus p_b\oplus p_c$ を図-10 の予測回路として実現すればよい．この場合，キャリー c_i $(i=0\sim n-1)$ の生成回路 $c_i=Maj(a_i, b_i, c_{i-1})$ は演算回路用と予測回路用で二重化される．

一方，論理演算に関しては，パリティ検査符号が Ex-OR 演算に対して保存されることを除けば，二重化（二線式符号）より冗長度の小さい誤り検出符号で保存されるものは存在しない[32]．したがって，一般に図-10 の方式になるが，実質的には，二重化機能回路の出力の一致を 図-8 の TSC 比較回路で検出するのと同じになる．二重化の特別な場合である二線式論理 (Two-rail logic)[6] も用いられる．たとえば，**図-11(a)** の回路の二線式論理を同図 (b) に示す．二線式論理は NOT 素子が不要のため，単純な二重化よりスピード及び金物量の点で有利な場合がある．

実際には，算術演算回路と論理演算回路は ALU として一つにまとめられる．したがって，検査部予測回路を算術演算と論理演算で共用したり[6]，剰余符号による算術演算の結果を論理演算の検査に利用する[33]等の金物量を小さくする工夫がなされている．また，データ非変換系に比べて，ALU を TSC 化することは金物の冗長性が大きくなり経済的に引き合わないという立場から，符号が保存される演算に対してのみ 図-9 の方式によって TSC を保証し，保存されない演算に対しては，その時だけ FS 性が失われても止むを得ないとする考え方 (Partially self-checking)[34] もある．たとえば，ALU に剰余符号を用いて図-9 の方式を適用し，論理演算実行時のみ (c_1, c_0) を制御入力で強制的に (1, 0) または (0, 1) に固定すると，その時だけ FS 性が失われるが，それ以外の時は FS かつ ST である．

ⅲ) 制 御 系

組合せ回路と順序回路に分けることができる．

データ系とは異なり，制御系の組合せ回路では符号の保存性は重要ではない．与えられた符号化出力に対して個々に論理関数が定義されるからである．通常，①ビットスライス構成による単一ビットパリティ符号，②バイトスライス構成による b-隣接符号，チェックサム符号，③一方向性多重故障に有効な，M-

out-of-N 符号，バーガー符号，二線式符号，等が用いられる．①，②は単一スライス誤りに対して FS になる．入力符号空間に③を用いると NOT 素子を含まない形式で回路を構成できるので[35),36)]，入力と出力の両方に③を用いると，すべての一方向性多重故障に対して FS になる．①～③のいずれの場合も，回路が ST 性を持つかどうかはその構造に依存する[37)]．

順序回路は状態遷移関数と出力関数によって記述されるので，定義 1～3 の定義は必ずしもそのまま適用できない．今のところ，順序回路の設計目標として十分意味を持つ TSC の概念に関して大方の合意を得た定義はなく，定義 1～3 を“常識的”に順序回路へ拡張した解釈がなされている[7),38)]．順序回路に FS 性を持たせる設計法としては，現在まで，状態割当に M-out-of-N 符号もしくはその他の非順序符号を用いることにより状態遷移回路を NOT 素子なしで構成し[35)]，外部入出力線は二線式とする[21),38)]方法以外に有効なものはない．この方法によれば，一方向性多重故障が生じると順序回路を非符号語状態へ“trapping”することができ，FS 性が達成される．一方，ST 性を持つ設計は必ずしも容易ではない[39)]．このため，TSC 順序回路の設計に関してはまだ十分な解決は与えられていない[21),38)~41)]．また，ST 性の条件を緩和した概念の提案もある[37),42)]．

制御系としては他に，マイクロプログラム制御回路の自己検査性に関する検討もなされている[43)~46)]．

4. むすび

フォールトトレラントシステムを支援する論理回路レベルの二つの基本技術，故障マスクと自己検査性，について現状を概観した。最近では，LSI 技術の進歩を背景に，これまで個別に確立されている自己検査性設計技術を LSI プロセッサに適用するための評価，検討がなされている[47)~49)]．PLA 構造の利用[50)]を含めて，今後は LSI/VLSI によるインプリメントが重要であろう．

参考文献

1) Avizienis, A.: Fault-tolerance: the survival attribute of digital systems, Proc. IEEE, 66, 10, pp. 1109-1125 (1978).
2) 当麻，南谷：フォールトトレラントシステム，電子通信学会誌，63, 10, pp. 1031-1041 (1980).
3) 奥村幾正：フェイルセイフシステムの構成，電子通信学会誌，62, 2, pp. 198-205 (1979).
4) Wilcox, R. H., Mann, W. C.: Redundancy Techniques for Computing Systems, Spartan Books (1962).
5) Pierce, W. H.: Failure-Tolerant Computer Design, Academic press, New York (1965).
6) Sellers, F. F. et al.: Error Dtecting Logic for Digital Computers, McGraw-Hill, New York (1968).
7) Wakerly, J.: Error Detecting Codes, Self-Checking Circuits and Applications, North-Holland, New York (1978).
8) 宮川，岩垂，今井：符号理論，昭晃堂 (1973).
9) Larsen, R. W., Reed, I. S.: Redundancy by coding versus redundancy by replication for failure-tolerant sequential circuits, IEEE Trans, C-21, 2, pp. 130-137 (1972).
10) Moore, E. F., Shannon, C. E.: Reliable circuits using less reliable relays, J. Franklin Inst. 262, 9 and 10, pp. 191-208 and pp. 281-297 (1956).
11) Von Neumann, J.: probabilistic logics and the synthesis of reliable organisms from unreliable components, Automata Studies, 34, pp. 43-98 (1956).
12) Brown, W. G., Tierney, J.: Improvement of electronic computer reliability through the use of redundancy, IRE Trans., EC-10, 9, pp. 407-416 (1961).
13) Gurzi, K. J.: Estimates for best placement of voters in a triplicated logic network, IEEE Trans., EC-14, 10, pp. 711-717 (1965).
14) Tryon, J. G.: Quadded logic, 文献 4), pp. 205-228.
15) Pradhan D. K., Stiffler, J. J.: Error-correcting codes and self-checking circuits, Computers, 13, 3, pp. 27-37 (1980).
16) Hamming, R. W.: Error detecting and correcting codes, Bell Syst. Tech. J., Vol. 29, pp. 147-160 (1950).
17) Armstrong, D. B.: A general method of applying error-correcting to synchronous digital systems, Bell Syst. Tech. J., 40, pp. 572-593 (1961).
18) 当麻喜弘：フォルトトレラントとその関連技術，電子通信学会誌，59, 4, pp. 359-368 (1976).
19) Rao, T. R. N.: Error Coding for Arithmetic Processors, Academic Press, New York (1974).
20) 藤原英二：誤り訂正符号とその高信頼化ディジタルシステムへの応用に関する研究，東工大学位論文 (1981).
21) Carter, W. C., Schneider, P. R.: Design of dynamically checked computers, IFIP 68, pp. 878-883 (1968).
22) Anderson, D. A., Metze, G.: Design of totally self-checking check circuits for m-out-of-n codes, IEEE Trans., C-22, 3, pp. 263-269 (1973).

23) Shedletsky, J. J. et al.: The error latency of a fault in combinational digital circuits, FTC-5, pp. 210-214 (1975).
24) Bossen, D. C.: b-Adjacent error correction, IBM J. 14, 4, pp. 402-408 (1970).
25) Avizienis, A.: Arithmetic codes: coct and effectiveness studies for application in digital systems design, IEEE Trans., C-20, 11, pp. 1322-1331 (1971).
26) Ashjaee, M. J., Reddy, S. M.: On totally self-checking checkers for separable codes, IEEE Trans., C-26, 8, pp. 737-744 (1977).
27) Marouf, M. A., Friedmann, A. D.: Design of self-checking checker for Berger codes, FTCS-8, pp. 179-184 (1978).
28) Wakerly, J. F.: Checked binary addition with checksum codes, J. DA. FTC., 1, 1, pp. 18-27 (1976).
29) Carter, W. C. et al.: Computer error control by testable Morphic Boolean functions—a way of removing hardcore, FTC-2, pp. 154-159 (1972).
30) Reddy, S. M.: A note on self-checking checker, IEEE Trans., C-23, 10, pp. 1100-1102 (1974).
31) Marouf, M. A., Friedman, A. D.: Efficient design of self-checking checker for any m-out-of-n code, IEEE Trans., C-27, 6, pp. 482-490 (1978).
32) Peterson, W. W., Rabin, M. O.: On codes for checking logical operations, IBM J. 3, 2, pp. 163-168 (1959).
33) Rao, T. R. N., Monteiro, P.: A residue checker for arithmetic and logical operations, FTC-2, pp. 8-13 (1972).
34) Wakerly, J. F.: Partially Self-checking circuits and their use in performing logical operations, IEEE Trans., C-23, 7, pp. 658-666 (1974).
35) Tohma, Y. et al.: Realization of fail-safe sequential machines by using a k-out-of-n code, IEEE Trans., C-20, 11, pp. 1270-1275 (1971).
36) Mago, G.: Monotone functions in sequential circuits, IEEE Trans., C-22, 10, pp. 928-933 (1973).
37) Smith, J. E., Metze, G.: Strongly fault secure logic networks, IEEE Trans., C-27, 6, pp. 491-499 (1978).
38) Diaz, M.: Design of totally self-checking and fail-safe sequential machines, FTC-4, pp. 19-24 (1974).
39) Pradhan, D. K.: Asynchronous state assignments with unateness properties and fault-secure design, IEEE Trans., C-27, 5, pp. 396-404 (1978).
40) Özgüner, F.: Design of totally self-checking asynchronous and synchronous sequential machines FTCS-7, pp. 124-129 (1977).
41) Nanya, T., Tohma, Y.: Design of self-checking asynchronous sequential circuits,FTCS-10, pp. 278-280 (1980).
42) Viaud, J., David, R.: Sequentially self-checking circuits, FTCS-10, pp. 263-268 (1980).
43) Toy, W. N.: Modular LSI control logic design with error detection, IEEE Trans., C-20, 2, pp. 161-166 (1971).
44) Cook, R. W. et al.: Design of a self-checking microprogram control, IEEE Trans, C-22, 3, pp. 255-262 (1973).
45) Chang, H. Y. et al.: The design of a microprogrammed self-checking processor of an electronic switching system, IEEE Trans. C-22, 5, pp. 489-500 (1973).
46) Diaz, M. et al.: Design of self-checking microprogram controls, FTC-5, pp. 137-142 (1975).
47) Carter, W. C. et al.: Cost-effectiveness of self checking computer design, FTCS-7, pp. 117-123 (1977).
48) Sedmak, R. M., Liebergot, H. L.: Fault-tolerance of a general purpose computer implemented by very large scale integration, FTCS-8, pp. 137-143 (1978).
49) Crouzet, Y., Landrault, C.: Design of self-checking MOS LSI circuits, application to a four bit microprocessor, FTCS-9, pp. 189-192 (1979).
50) Wang, S. L., Avizienis, A.: The design of totally self-checking circuits using porgrammable logic arrays, FTSC-9, pp. 173-180 (1979).

(昭和57年1月5日受付)