Research Reports of the National Institute
of Industrial Safety, NIIS-RR-95 (1996)
UDC 62-192:519.681:681.3.05:681.5.09

符号化によるマイクロプロセッサのエラー検知手法と評価

池田　博康*

Error Detection by Coded Microprocessor and Its Evaluation

by Hiroyasu IKEDA*

Abstract: A microprocessor is indispensable to complex control system such as railways, aircraft and atomic power plants. However, standard mono-microprocessors are obviously not designed fail-safe, and they require protection against programming errors and hardware failures when used in safety-related systems or critical control fields.

In order to realize such protection, multiple microprocessors with a voter (for example, a majority element for 2-out-of-3) have been widely used to give high dependability, which equates to high-safety, high-security, high-availability and high-reliability. However, the redundant hardware system may become complex and be expensive, thus this kind of system would imply being used only in a limited process.

On the other hand, various coding techniques of information sources and channels have been developed in communication systems to improve transmission reliability or fidelity. A redundant arithmetic coding has been specially considered for vital data processing. It is required reducing hardware complexity to improve speed and to ensure reliability in computation. Above all, redundant residual codes have been used as error-detecting or error-correcting codes in data processing.

The coded microprocessor, which produces and deals with such residual codes to protect against hardware failures, has redundancy not of hardware and software but only of information. This unique innovative technique has been designed to detect errors occurring in data processing.

In this paper, the architecture of the coded microprocessor to detect any potential error during the processing of data was proposed by means of the coding principle. Further, the error detectability of the coded microprocessor was discussed.

The coded microprocessor is a standard mono-microprocessor running one program with external fail-safe monitoring or checking hardware (such as code generator, clock generator, dynamic comparator), and uses all coded redundant information (data) and added signatures based on residual arithmetic (remainder theorem). Therefore, it has the safety level that depends on only information redundancy, not on the total number of redundant codes and the processor or its reliability, provided that the global signature streams generated by the microprocessor by comparison with a predefined sequence.

The error detection capabilities is presented as well as the assessment of the non-detection probability of errors. For example, such error detectability can be set up 10^{-12} in terms of probabilities, if the check word length is 40 bits. The safety demonstration of the coded microprocessor has been investigated and accepted so that any error which may occur in the processor is detectable by the code and the probability of non-detection of error is defined by choice in a random manner.

Keywords; Fault tolerance, Error detection, Self check, Coding, Microprocessor

*機械システム安全研究部 Mechanical and System Safety Research Division

1. 緒　　言

機械の制御システムは，電子技術の進歩に伴い，ここ近年より複雑化・高速化・小型化しており，あらゆる機能がマイクロプロセッサの管理に委ねられるようになってきた。そのような傾向は，インターロックのような安全に係わる機能や制御にまで及ぼうとしており，プロセッサの故障や処理エラーが機械の誤動作に至る危険性が懸念されているとともに，実際に事故も起きている[1]。

マイクロプロセッサを機械の制御システムに利用するには，ソフトウェア及びハードウェアの安全性，すなわち前者はプログラムあるいは特定のエラーに対する防護を，後者はマイクロプロセッサとその周辺のハードウェア故障に対する防護について考慮しなければならない。

これらの問題に対処するため，様々な対策・手法が採られてきており，鉄道の信号制御[2]や航空機の管制システムの制御[3]の分野では対策の実用化が進んでいる。そして，これらの対策のほとんどがプロセッサの耐故障性を向上させる高信頼性設計手法によるものであり，欧州規格によれば，信頼性設計の目標（例えば，障害発生確率等）をリスク分析により具体的に設定して設計するようになっている[4]。

一方，近年，マイクロプロセッサ内蔵のプログラマブル・ロジック・コントローラが産業プロセスの広い範囲に普及されるに伴い，異常や予期せぬ事象に対処する高信頼化を図るための能力のチェックが容易ではなくなってきている。これは，上記のリスク分析に基づく高信頼性設計の適用が難しく，安全性レベルの評価が適正にできないことを意味している。しかし，今後プロセッサの利用は避けることのできない情勢にあるため，安全性レベルが高いと判断されるプロセッサの高信頼化手法が求められている。

本報は，ハードウェアやソフトウェアの冗長化という従来の信頼性向上手法によらない，情報の冗長性に基づいた符号化プロセッサに関するものである。符号化プロセッサは，プロセッサの運用中にエラーが起こっても出来る限り運用を続けるような高稼働性を目的とするのではなく，起こり得る全てのエラーを確実に検出してプロセッサの運用を安全に停止する目的で導入される。その実現のため，エラー検出のための符号化原理を利用して新たに記号を付加した符号構成を検討し，この符号を適用できる自己診断機能を有する符号化マイクロプロセッサシステムの基本構成を明らかにした。また，提案したプロセッサシステムのエラー検出能力の理論的な検証を行い，安全性レベルの評価を行った。

2. 符号化によるプロセッサエラーの検出手法原理

2.1 符号化によるエラー検出

一般に，マイクロプロセッサを利用する上で安全の観点から問題とされているのは，

(1) 障害発生確率が高い要素と見なされている（代表的な障害発生率は 50×10^{-4}/h とされる[5]），
(2) 起こり得る障害モードが明確でない（安全側障害率は50%とされる[5]），
(3) 完全な検査が困難，
(4) 実環境下での障害データが少ないため分析が十分行われていない，

からであり，現状ではこれらの問題に多重冗長化で対処する場合が多い。例えば，プロセッサを含めた全ての構成ブロックを3重化して，プロセス処理結果を照合する 2 out of 3 手法[6]が一般的であるが，このような手法の信頼性の評価は，複雑なハードウェア構造に依存するため解析的に解くことが困難であり，数値計算による評価ツール[7]が提案されている程度で実用的ではない。

一方では，プロセッサで起こり得る障害を早期に検出するため，障害に至る前のエラーを検出，あるいは訂正してプロセッサの作動状態を自己診断する方法が広く用いられており，ウォッチドッグタイマー[8]や符号化法が古くから用いられている。後者の手法は，ハードウェアに負担をかけずにエラー検出を行うことができ，パリティチェック[9]等が汎用性のある技術として知られている。

このように，バイナリ表現されたデータに特別なビットを付加する符号化法は，プロセッサが扱う全てのデータや変数を符号化することにより，ハードウェアあるいはプログラム上のあらゆるエラーを予め把握しておかなくとも，それらのエラーが何かしら符号の違いとなって現れることを期待するものである。したがって，符号の工夫により診断が容易にでき，さらに情報数学上で符号を確率で論ずることができるため，符号化の効果を定量的に評価しやすい。

以上のことより，符号化によるエラー検出の特徴をまとめると次のようになる。

1) 符号の数学的性質により，リスク分析等から得られる信頼性設計の目標を実現できるかという評価，すなわちエラーを検出できずに危険側の障害に至る確率を導出可能である。
2) ハードウェアに依存しないため，マイクロプロセッサの種類や構造に無関係に適用できる。

3) エラーモデルについての初期仮定を不要にする。

つまり，全てのエラーモデルとその発生確率を完全に書き出すのは不可能であるため，エラーモデルを想定する限り，ハードウェア及びソフトウェアの完全な検査と分析は不可能である。また，冗長化ハードウェアであっても共通要因故障（すなわち，同じ原因で複数のハードウェアに起こる故障）の可能性があり，これらを把握できない以上，評価において不確定さが残る。したがって，エラーモデルを必要としない符号が適用される。

2.2 エラー検出目標

エラー検出符号を選択するために確率的エラーモデルを必要とする符号化手法では，もし，実際のプロセッサの動作（エラー）が予測していたものと異なるならば，符号によるエラー検出能力は無効となる。したがって，ここでは検出すべきエラーの対象をランダム故障により引き起こされるものとした。そして，制御対象機器のソースプログラムの記述からその実行までに起こるかもしれないあらゆる故障とエラーを検出することを目標とした。

プログラムのコンパイル中，あるいは実行中にランダム故障が起こると，これらの故障やエラーは次の3つの場合の組み合わせになる。

① オペレーション（演算）エラー
キャリーオーバーエラー，シフトエラー等の計算エラーであり，演算結果が誤りとなる。

② オペランド（演算項）エラー
不正なアドレスエラーやメモリのリフレッシュエラー等。

③ オペレータ（演算子）エラー
不十分なアドレスによるオペレータの置き換え等。

これらの内，オペレーションエラーについては，どんな符号であれ符号化の算術を実行することにより容易に検出可能であるが，オペランドエラーとオペレータエラーに対しては，符号化オペランドを符号化の結果に直接変換するオペレータとこのオペレータの容易な実行ができる符号化プロセスが必要である。

2.3 ランダム剰余符号の構成

2.3.1 符号化能力の指標

オペレーションを可能な限り簡易に行い，また符号化・復号化を容易にするため，変数のデータ値と符号が独立した分離形式符号語が有効である。よって，プログラムで与えられた変数は，符号化プロセスの中で機能部と制御部からなる符号化の値（すなわち符号化オペランド）Xで表される。このオペランドの非符号化値を機能データとしてN_fビットを持つものとし，付加された符号を制御データとしてN_cビットを持つものとすると，符号化言語Xは$2^{N_f+N_c}$となる。

このとき，エラーの未検出の確率Pi，すなわち誤りの符号語が現れると考えられる確率は，符号語がランダムに選ばれる場合，

$$Pi = \frac{2^{N_f}}{2^{N_c+N_f}} = \frac{1}{2^{N_c}} \tag{1}$$

Piは符号のエラー検出能力を示す指標となるため，プロセッサシステムの安全レベルを評価する因子の一つとなる。したがって，機能データに関係なく，符号すなわち制御データを十分大きく選ぶことによって，Piは理論的には希望通りの値が得られるので，エラー未検出確率を設定することで符号の大きさを決定することが可能である。例えば，$Pi = 10^{-12}$を目標とする場合，$N_c \geqq 40$となる。

2.3.2 剰余符号化によるオペレーションエラー検出

符号の種類は様々であるが，前述したようにオペレーションが容易に実行できる符号としては算術符号がある。さらに，上記の例のように40ビット以上の大きな符号の演算を迅速に行うためには，整数の剰余演算の利用が有利であり，この剰余符号を利用する演算エラーの検出原理が明らかにされている[10),11),12)]。

この剰余符号化の原理に基づくと，符号化言語Xの機能データx_fは整数を持つものとして次式で表される。

$$x_f = q_x A + r_x \tag{2}$$

ここで，Aは符号のキーと呼ばれる変数であり後述の理由から素数が選ばれ，除算x_f/Aの商がq_x，剰余がr_x $(0 \leqq r_x < A)$である。また，式(2)はAを除数とする算術演算として次式のようにも表現される。

$$r_x = x_f \bmod A \tag{3}$$

一方，整数$k(>0)$を導入すると式(2)と同様に，

$$2^k = q_k A + r_k \tag{4}$$

となり，式(2)，(4)をまとめると，

$$r_{kx} = 2^k x_f \bmod A \tag{5}$$

となる。ただし，$r_{kx} = r_k r_x \bmod A$である。

いま，kを$2^k > A$である整数にとれば，$2^k x_f$は変数Xの機能データx_fがkビット重み付けられて上位のN_fビットとして置かれ，$-r_{kx}$の値はkビット分の制御データとして下位のN_cビット上に置かれて剰余符号となる。

このようにして，$2^k x_f - r_{kx}$がAの倍数，すなわち剰余ゼロを確認することにより，この算術は計算のエ

ラーを $1-1/A$ の確率で検出することを可能にする。ただし，この符号化により全ての符号語は A の倍数となるから，オペランドエラーやオペレータエラーは検出されない。

2.3.3 記号の付加によるオペランドエラー及びオペレータエラーの検出

アドレシングエラー（オペランドエラー，オペレータエラー，変数の混乱等）を検出するため，プログラム中のどんな変数に対しても，その機能データとは独立した記号を符号に付加することができる。この記号は，算術符号の一部として利用されるため既知の整数とし，予め1から $A-1$ の範囲からランダムに選ばれる。

いま，符号化された変数 X の制御データに記号 B_X が付加されると，X は次式で表される。

$$X = 2^k x_f - r_{kx} + B_X \qquad (6)$$

ここで，$-r_{kx} + B_X$ は制御データ x_c に相当する。

次に，加算 $X = Y + Z$ を例にしてエラー検出の原理を説明する。変数 Y, Z 共に式 (6) と同様に表されるものとし，加算が正しく実行されると，符号化された値 X は次式となる。

$$X = 2^k(y_f + z_f) - r_{k(y+z)} + F_{plus}(B_Y, B_Z) \qquad (7)$$

ただし，F_{plus} は $(B_Y + B_Z) \bmod A$ の機能である。オペランドエラーがある場合，オペランド T が Y の代わりに用いられると，X の制御データ x_c は $-r_{k(t+z)} + (B_T + B_Z) \bmod A$ となるため，エラーは $B_T \neq B_Y$ ならば検出される。

次に，オペレータエラーにより F_{plus} の代わりに F_{op} が用いられるとすると，符号化された値 X は次式となる。

$$X = 2^k(y_f \text{ op } z_f) - r_{k(y \text{ op } z)} + F_{op}(B_Y, B_Z) \qquad (8)$$

ここに，F_{op} は $(B_Y \text{ op } B_Z) \bmod A$ の機能であり，エラーは，$F_{op}(B_Y, B_Z) \neq F_{plus}(B_Y, B_Z)$ ならば検出される。

さらに，オペランドエラーとオペレータエラーが両方ある場合，$F_{op}(B_T, B_Z) \neq F_{plus}(B_Y, B_Z)$ ならば，たとえ B_Z が B_U となる2重のオペランドエラーがあったとしても $F_{op}(B_T, B_U) \neq F_{plus}(B_Y, B_Z)$ ならば検出される。

一方，乗算の場合は，A に素数を選んで乗法演算を行えば，乗法の逆要素が存在することから，乗法オペレータを直接実行可能にすることが示されている[13]。

2.3.4 動的記号の導入

前述の記号 B_X によりアドレシングエラーが検出されても，プログラムの1サイクル中に B_X が変わらなければエラーがマスクされる恐れがある。そのため，新たに日付（時間）を示す動的な記号 D を導入して，例えばプログラムループ数の誤りや不要なデータの記憶等を検出するものとした。この記号は，一つの同じサイクル中に実行される全ての符号化されたデータに共通し，各サイクル毎に修正させるため，これらのデータの一時的な有効性を保証するものとなる。

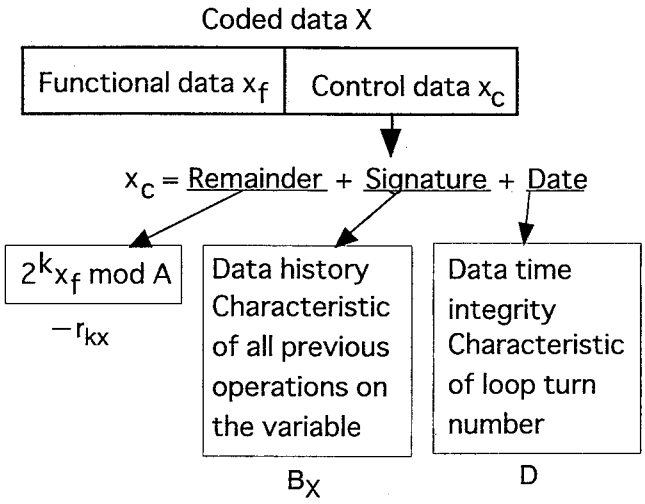

Fig. 1　Structure of code word with two separable data.
2つの分離データを持つ符号語の構成

したがって，動的記号 D は記号 B_X と同様に選ばれて，結局変数 X は **Fig. 1** に示すような構成となる。全てのオペランドが同じ動的記号 D を用いるため，符号化オペレータはこれを必ず利用しなければならない。

2.4 主な符号化オペレータ

加算と乗算については直接各々のオペレータにより実行される。

他に基本的なオペレータとして条件付き分岐があるが，これには比較演算を含めた条件付き分岐テストを行うことになる。

いま簡単に，次のアルゴリズムをとる。ただし，記号「：=」は右辺項を左辺項へ代入するという意味である。

IF	$T \geqq 0$
Then	$Z := X + Y$
Else	$Z := W$
ENDIF	

テスト終了時，Z の記号は分岐エラーにより影響されるが，もしエラーがなければ記号は2つの条件付き分岐 (Then と Else) の収束後，同一のものでなければならない。この条件は最終結果の記号をチェックできることを要求している。

$T \geqq 0$ の比較の記号を得るためには，符号化された T (この記号を B_t とする) を A を除数として算術演算を行う。その剰余は，$T \geqq 0$ ならば $R_1 = B_t$ であり，$T < 0$ ならば n をビット数とすると $R_2 = B_t + r_n$ (すなわち，$r_n = 2^n \bmod A$) である。したがって，記号 B_t と比較の結果により，比較オペレータとしてのマーカーが実現されることになる。

次に，2 つの記号の差が全てのテストで等しくなる $(R_1 - R_2 = r_n)$ ことを避けるため，マーカーのマーク R に各テストで異なる機能 F_i を割り当てることにより，比較の記号 S を計算する。そして値 S_{i1} か S_{i2} をとる記号 S_i を得て，比較結果による記号を得るために変数を付加しなければならない。そして，テストの else 分岐内で予め計算された定数 C_{iz} を変数 Z に加えている限り，分岐エラーがなければ選択された分岐に無関係な変数 Z の最終記号 B_Z を得ることができる。

結局，分岐構造アルゴリズムは次のように表され，変数 Z の記号 B_Z は予め計算され記憶された値 $C_{iz}(= B_X + B_Y - B_W + S_{i1} - S_{i2})$ を用いて算出される。

プログラム		記号 B_Z
$R := T \bmod A$		
$S := F_i(R)$		
IF $T \geqq 0$		
then	$Z := X + Y$	$B_Z = B_X + B_Y$
else	$Z := W$	$B_Z = B_W$
	$Z := Z + C_{iz}$	$B_Z = B_W + C_{iz}$
ENDIF		
$S := Z + S_i$		$B_Z = B_X + B_Y + S_{i1}$ $= B_W + S_{i2} + C_{iz}$

以上の 3 つの基本的なオペレータ（加算，乗算，条件付分岐）を用いて，以下のオペレータが実現可能である。

- 整数オペレータ（減算，ビットの桁落ち）
- ブール代数オペレータ (AND, OR, NOT, Exclusive OR 等)

なお，繰り返し構造（ループ）のテストは，動的記号 D と符号化条件付き分岐を直接利用することにより実現される。

3. 符号化プロセッサの構成と機能

フェールセーフなマイクロプロセッサが実用化されていない現状では，プロセッサで扱うデータを全て符号化することによりあらゆるエラーを検出しようとする手段は，プロセッサと独立した要素を伴わざるを得ない。データの符号化及びエラー検出の重要な機能については，多くのビットを並列処理するのでなければ，

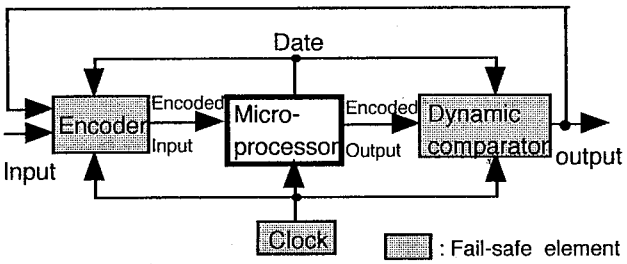

Fig. 2 Basic architecture of coded microprocessor system.
符号化マイクロプロセッサシステムの基本構成

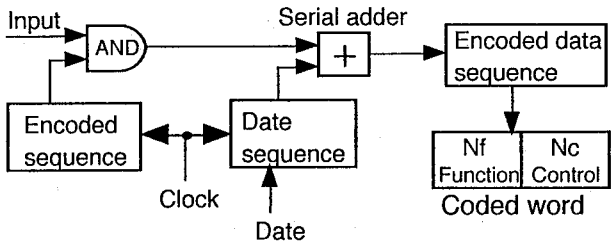

Fig. 3 Principle of serial encoding of input data.
入力データの逐次符号化原理

実用的なフェールセーフ要素[14),15)]に依存することが可能である。このような要素を利用することによって，プロセッサ自体の構造・種類に依存せず，プロセッサも含めた冗長ハードウェアを必要としない符号化プロセッサシステムが実現できる。

フェールセーフ要素を含んだ符号化プロセッサシステムのハードウェア構成は **Fig. 2** で示される。基本的には，入力符号器による入力データの符号化機能と，ダイナミックコンパレータによる符号化出力データの持つ記号を参照記号と比較する機能が備われば，全てのサイクルにおけるエラー検出が可能となる。

3.1 入力符号器による符号化

符号化されるべき入力データは，センサ等の外部からのデータと後述する出力のフィードバックデータがあり，これらのデータをフェールセーフ要素を利用して誤りなく符号化するには，全入力データの 1 ビット毎に逐次符号化のためのデータを付加する方法が有効である。つまり，入力データビットが「1」(高レベル) のときだけ特定の記号データが用意されて，符号化データ列に変換されるよう操作される。このような逐次処理ではダイナミックな自己診断が容易であり，既存のフェールセーフ技術が適用できる。

Fig. 3 は予め設定されたシーケンスに従って入力データを符号化する原理を示す。このような逐次変換方式は，入力データの状態が入力サンプリング中に遷

移しないような配慮が必要となる。また，符号化されたデータには，マイクロプロセッサにより処理される前に動的記号（日付）が付けられる。

3.2 ダイナミックコンパレータによる符号検査

入力の符号化された値を受け取ったマイクロプロセッサは，データの機能データと制御データ（符号化部）のサイクルを実行し，**Fig. 4** に示すように動的記号の付加されたシーケンスデータをダイナミックコンパレータへ送る。

ダイナミックコンパレータは受信されたシーケンスを解読し，それが正しいことを検証してはじめて出力許可の信号をエネルギーとして供給する。**Fig. 4** に示すように，このエネルギーは，動的記号の付いたシーケンスからこれを解除したデータがダイナミックコンパレータの PROM に予め記憶させてある参照記号データと一致するときのみ供給される。記号の検証を行う比較器は，入力符号器と同様に，ビット毎に排他的論理和演算 (Exclusive OR) を逐次実行するようになれば，既存のフェールセーフ技術が適用できる。このような符号検査方式は，確定的なプログラムにより参照記号を用意しておくことで，特別なハードウェアなしに容易に実現可能である。

3.3 出力状態のチェックサイクル

一般に，マイクロプロセッサシステムの出力は，たとえ安全リレーを介してアクチュエータを駆動する場合でも，出力インタフェースの段階ではフェールセーフが保証されない。したがって，マイクロプロセッサから実際に生成される「0」（低レベル）出力がインタフェースボード上でも「0」であり，誤って「1」すなわち許可出力に遷移しないことを検証する手段が必須である。そのため，出力をフィードバックして再入力させて，出力状態の確認をする監視機能が常套手段として設けられる。この機能を実現するには，反転したインタフェース出力「1」を再び入力符号器を通して読み込み，新たに符号語の制御データとして使用すればよい。このとき，「0」が入力されるようならば，ダイナミックコンパレータで制御データの記号が一致せず，出力許可のエネルギーが供給されなくなるため，直ちにインタフェース出力は全て遮断される。

Fig. 5 はインタフェース出力の再読み込みによる確認の原理を示しており，フィードバック用反転出力はリレー接点で得られることが明らかにされている[16]。このようなリレー出力の場合，リレー出力の検証のためある程度の時間が必要となり，プロセッサのプログラムサイクル終了前にチェックする手順を組み込んでおく。**Fig. 6** は，全入力読み込み，データ処理，全出力セットというプログラムシーケンスの１サイクルを示しており，通常，出力セットのサイクルでダイナミックコンパレータによる出力照合が行われる。リレー出力の場合は，データ処理プロセスの結果，「0」を出力すべきときは，出力照合サイクルにフィードバックの再読み込みが間に合うよう早めに出力され，「1」を出力すべきときは，リレー故障が「0」側であるからリレー出力自体の照合は必要ない。

なお，出力照合はプログラムシーケンスの１サイクル終了毎に行われるが，入力符号器，ダイナミックコンパレータ共にビット毎の逐次処理であるから，**Fig. 6** の１サイクルは，**Fig. 2** のクロックに従ってＩ番目入力ビットが読み込まれてプロセッサで処理され，Ｉ番目出力ビットが照合される。すなわち，符号語の全ビットがサイクルを終了する前にエラーが検出される可能

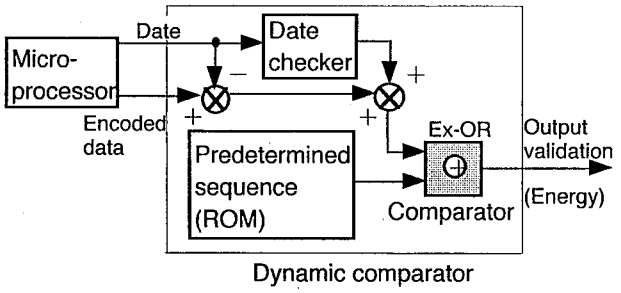

Fig. 4　Principle of code checking by dynamic comparator.
ダイナミックコンパレータによる符号チェックの原理

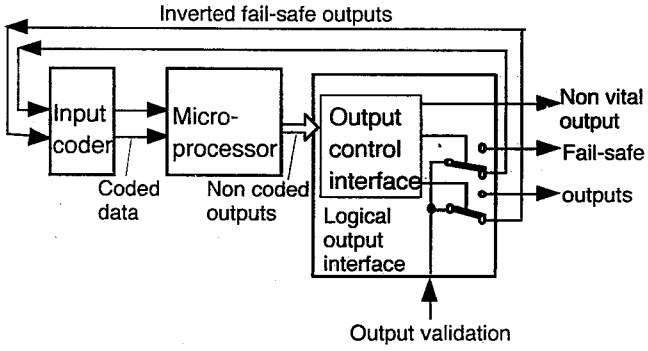

Fig. 5　Output validation by rereading output condition.
出力状態の再読み込みによる出力確認

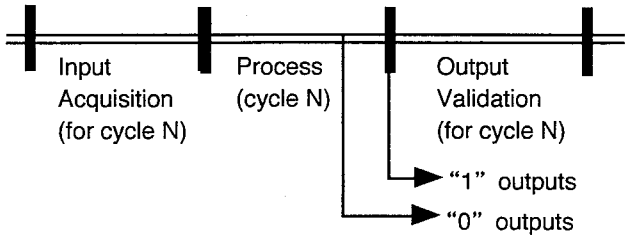

Fig. 6　Program sequence for output check cycle.
出力チェックサイクルのためのプログラムシーケンス

性が非常に高いことになる。

3.4 エラー照合用参照テーブルの作成

符号化プロセッサにおける符号語の優れたエラー検出機能は，処理された制御データを照合するための記号参照テーブルを予め準備していることに大きく依存している。確定的なプログラムが設計されるならば，特別なハードウェアを用いなくとも，符号化されたソースプログラムを解析することにより，符号語の制御データ部の記号を計算しておくことが可能である。これらの記号をダイナミックコンパレータ内のROMに書き込んでおくことにより，リアルタイムでのエラー検出が実現でき，なおかつ，プログラムと並行に記号を計算することで問題となる共通故障（エラー）モードも回避している。

また，符号化オペレータも予めライブラリとして準備しておくことができる。

4. エラー検出能力の検証

4.1 エラー未検出確率

各符号化変数は各命令後に制御されると仮定する。エラー (Eの値を持つ) が，符号化変数 Xを $X' = X + E$ (ただし $X' \neq X$) に変えるものとすると，エラー Eは，X'が **Fig. 1** の構造の符号語であれば検知できない。例えば，伝送通信路で符号化データ伝送の場合，メッセージ X内のエラー Eを検知する確率の決定はエラーモデルすなわちメッセージ X内の発生確率 $P_X(E)$ が既知ならば可能である。未検知確率 $P_{nd}(x)$ は次式で与えられる。

$$P_{nd}(x) = \sum (X + E) P_X(E) \tag{9}$$

ここで，$E \in E_X$であり，E_Xは符号によって検知できないエラーの集合である。$P_X(E)$ の内容は，伝送通信路の特性と環境からの妨害に由来する。

符号化プロセッサの場合，故障のモードと発生確率は未知なので，$P_X(E)$ を決定することは不可能とされてきた。そこで，エラーモデルによらない未検出確率を決定するため，符号語をランダムに選ぶものとすると，エラーの未検出確率の計算は次のように考えられる。

いま，(Mの要素数)/A の要素を含む全ての機能データ c_fと制御データ c_cの集合 M の中からランダムに選ばれた c_fと c_cの集合を C_Xとおく。よって，M のどの符号語 $X' = x_f' + x_c'$ にも $x' \in C_X$となる確率は次式で与えられる。

$$P = \frac{C_X\text{の要素数}}{M\text{の要素数}} = \frac{1}{A} \tag{10}$$

次に，符号化プロセッサで用いられる語の集合を G_Xとすると，

$$G_X = 2^k c_f - c_c + B_X + D \tag{11}$$

であり，G_Xがランダムに選ばれた語の集合 C_Xとしてとられたら，全ての $X' = X + E$で符号語 ($X' \in G_X$) となる Xの確率は $P_1 = 1/A$ である。換言すれば，変数 Xでのエラー Eの結果は何であれ，符号化プロセッサにより検出できないエラー Eの確率が $1/A$ となる。

なお，G_Xの全ての語はランダムな性質を持つものである。

4.2 エラー補償損失確率

次に，各命令実行前にエラーが発生して符号化変数が符号外となるとき，そのエラーを命令終了後に補償しようとして結果的に検出できない場合を想定する。すなわち，1つあるいはそれ以上の符号化変数がプロセシング中に誤っているとき，プログラム終端でダイナミックコンパレータに送られる最終記号上のエラー E_gは，多項式 $E_g = Q(E_1, E_2, \ldots, E_m)$ として表すことができる。この多項式がゼロ，すなわち $Q(0, 0, \ldots, 0) = 0$ となるのは，Q の係数がエラーの起こる瞬間から最終記号の送出までの間に実行される命令数に依存することである。すなわち，ダイナミックコンパレータがビット毎の逐次処理を行うため，上述のように符号語がランダムな性質を持ち，多項式 Q の係数がランダムな多項式においては，エラー補償のための検出能力の損失 P_2がどんなエラー $E \neq (0, 0, \ldots, 0)$ に対しても $P_2 = 1/A$ となる。

以上の考察より，符号化プロセッサにおける全体のエラー未検出確率 P_{nd}は，符号のキー A を用いて次式で表されるため符号の大きさにのみ依存することが分かった。

$$P_{nd} = P_1 + P_2 = \frac{2}{A} \tag{12}$$

なお，エラー補償損失確率の想定では符号語のランダムな性質にのみ着目したが，実際にはプログラムの長さや同時多重エラーの発生を考慮する必要が生じるかもしれない。今後，厳密な検討が必要となると思われる。

5. 結　　言

本報で得られた結果をまとめると次の通りである。

(1) マイクロプロセッサでの不可避なデータ処理エラーや故障に対して，誤りなくエラー検出するにはエラー未検出確率を極小にする必要があるが，全データに除算操作後の剰余を付加したランダム符号を

利用すれば，プロセッサの形態に依存せずエラーモデルの想定を不要とするため，確率的に極小値を算出できることが明らかになった。

(2) ランダム剰余符号に時間や処理過程に応じて変わるような記号を付加することにより，起こり得るエラーが符号の変化として現れるため，この符号を予め算出された正しい符号表とフェールセーフな手段によって照合することによりあらゆるエラーが最終的に検知可能となることが判明した。

(3) 通常の単一マイクロプロセッサに，既存のフェールセーフ技術に基づく入力符号器や符号照合回路を付加した簡単な符号化プロセッサシステム構成を明らかにした。

(4) エラーを見分けられない確率とエラー補償のための検出能力の損失確率で表されるエラー未検出確率が，符号化プロセッサのエラー検出能力を示す指標となり，これらは符号化操作時の剰余と記号のビット数から算出できることが分かった。

機械等の制御装置へのプロセッサの適用は拡大していくと予想されるが，プロセッサシステムの故障や異常に対する安全性が確保されなければ安全に係わる用途には使用が難しい。今後，符号化プロセッサをはじめとする新しい技術を実際の機械の制御装置へ適用する方法はもとより，その結果得られる安全性レベルの評価・検証が重要になると思われる。

謝　辞

符号化の原理やプロセッサシステムの基本構成を検討するに当たり，ご助言をいただいた英国 Health and Safety Laboratory の A.M. Wray 博士に深く謝意を表します。

参考文献

1) Neumann, P., Computer-Related Risks, 12–93, ACM Press (1995).
2) 奥村，中村，鉄道信号制御に見るフォールト・トレラント・システムの設計，日経エレクトロニクス，152–174 (1982).
3) Avizienis, A. and Ball, D., On the achievement of a highly dependable and fault-tolerant air traffic control system, Computer, 20–2 (1987).
4) Redmill, F., Dependability of Critical Systems 2, Elsevier Applied Science, 44–7 (1989).
5) Functional safety: safety-related systems, IEC65A Draft Part 2: Requirements for Electrical/Electronic/Programmable electronic systems, 97–113 (1994).
6) 井原，フォールトトレラントコンピュータ，電学誌，105–8, 745 (1995).
7) Makam, S. and Avijienis, A., ARIES: A reliability and life-cycle evaluation tool for fault-tolerant system, 12th Int. Symp. Fault-Tolerant Comput. (1982).
8) 並木，古賀，自己検査性ウォッチドッグタイマの一構成法，信学論，J68–D–8, 1543–1544 (1985).
9) 岩尾，コンピュータ・エラー：チェックの方法と実際，76–84，創元社 (1974).
10) Gösel, M. and Graf S., Error Detection Circuits, 35–41, McGRAW-HILL BOOK Co. Europe (1993).
11) Jenkins, W., The Design of Error Checkers for Self-Checking Residue Number Arithmetic, IEEE Trans. on Computers, C–32–4, 388–396 (1983).
12) Barsi, F. and Maestrini, P., Error Correcting Properties of Redundant Residue Number Systems, IEEE Trans. on Computers, C–22–3, 307–315 (1973).
13) 汐崎，情報・符号理論の基礎，100–103，国民科学社 (1991).
14) Lohmann, H., URTL circuit system Ul with high safety and automatic fault diagnosis, Siemens Rev. XLII (1975).
15) 加藤，フェイル・セーフ理論積 IC とその応用，信学技法，FTS86–3 (1986).
16) Forin, P., Vital Coded Microprocessor Principles and Application for Various Transit Systems, IFAC Control Computers Communications in Transportation, 79–84 (1989).

(平成 8 年 7 月 16 日受理)