社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

耐故障性を有するLUTカスケード・エミュレータについて

中原　啓貴† 　笹尾　勤†

† 九州工業大学 情報工学部 〒820–8502 福岡県飯塚市大字川津 680–4

あらまし　LUT カスケード・エミュレータは任意の順序回路を実現する．まず, 与えられた順序回路の組合せ論理関数部分を LUT カスケードに変換し, LUT カスケードのセルデータをメモリに格納する．次に, セルデータを逐次読み出しながら, 多出力論理関数を評価する．耐故障性を向上させるために, 論理メモリに自己検査回路を付加する．論理メモリに格納するデータを Berger 符号語に置き換え, 自己検査回路でデータが符号語であるかを常時監視する．自己検査回路により, 論理メモリの多重単方向故障, デコーダの単一縮退故障, 接続回路の単一縮退故障を検出可能である．故障が検出された場合には, モニタは, 回路を停止し, 故障回避モードに移行する．さらに, LUT カスケードのメモリパッキングを利用して, 論理メモリの故障箇所を回避して回路を再構成する．これにより, 回路の稼働率 (アベイラビリティ) を向上させることが可能である．

キーワード　LUT カスケード, 再構成可能アーキテクチャ, 自己適応性アーキテクチャ, 関数分解

A Fault Tolerant Look-Up Table Cascade Emulator

Hiroki NAKAHARA† and Tsutomu SASAO†

† Department of Comoputer Science and Eloctronics, Kyushu Institute of Technology
kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan

Abstract　An LUT cascade emulator realizes an arbitrary sequential circuit. We convert the combinational part into multiple LUT cascades, and store LUT(cell) data into a memory in the LUT cascade emulator. It evaluates multi-output logic functions by reading cell data sequentially. To improve availability of the LUT cascade emulator, we add a self-checking circuit. Cell data stored in the memory are encoded into Berger codes. The self-checking circuit watches for whether memory outputs are Berger codes. The self-checking circuit can detect uni-directional errors of the memory for logic, a single stuck-at fault of a decoder, and a single stuck-at fault of the programmable connection circuit for rails. When a fault is detected, the monitor stops the LUT cascade emulator, and goes to the fault avoidance mode. First, it diagnoses the fault of the LUT cascade emulator. Next, it rewrites memory contents by memory-packing that avoids the failure parts of the memory for logic. Therefore, our method can improve the availability of the circuit.

Key words　LUT cascade, Reconfigurable architecture, Self-adaptability architecture, Functional decomposition

1. はじめに

プロセス微細化による高集積化に伴い, LSI 設計はますます複雑になっており, 論理機能設計の複雑度向上, 消費電力増加に加え, 回路のバラつきやランダム故障に伴う信頼性低下が問題となっている [1]．信頼性やアベイラビリティを確保するため, 回路に冗長性を持たせ, 動作のリアルタイム検証を行い, 故障に応じて回路を自己修復する手法が提案されている．本論文では, 耐故障性を有する LUT カスケード・エミュレータについて述べる．LUT カスケード [2] は, LUT(Look-Up Table) を直列多段に接続した構造を持ち, 多出力論理関数を実現する．LUT カスケード・エミュレータ [2] は LUT カスケードの LUT データを大規模なメモリに格納し, LUT データを逐次読み出すことで, LUT カスケードの模擬を行うアーキテクチャである．LUT カスケード・エミュレータでは LUT データを格納するメモリと信号線の接続情報メモリを書き換えることで, 種々の論理関数を実現可能である．本論文では, LUT カスケード・エミュレータの耐故障性を向上させる手法について述べる．まず, 第 2 章で, LUT カスケードと LUT カスケード・エミュレータの構造について述べる．第 3 章では, 自己検査回路について述べる．また, 自己検査に使用する符号語について述べる．第 4 章では, 故障が検出された場合, 故障を診断する手順, 及び故障を回避する手法について述べる．第 5 章では, 誤り訂正手法との比較, 及び耐故障性を有するために必要なハードウェア量を求めた実験結果について述べる．最後に, 第 6 章で本論文のまとめと今後の課題について述べる．

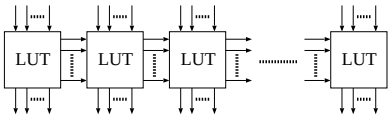

図 1　LUT カスケード

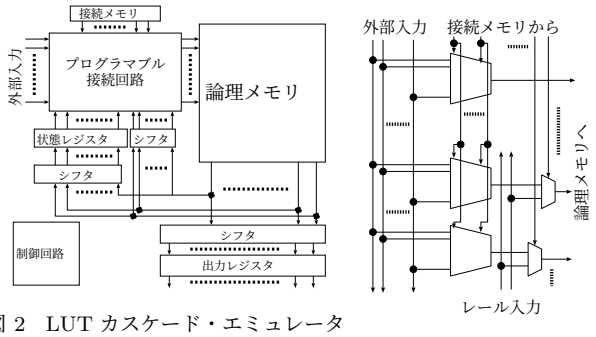

図 2　LUT カスケード・エミュレータ
(順序回路を実現)

図 3　プログラマブル接続回路

2. LUT カスケード・エミュレータ

2.1 LUT カスケード

多出力論理関数を実現する LUT カスケード [2] を図 1 に示す. LUT カスケードは LUT(**セル**) を直列多段に接続した回路である. LUT 間の接続線を**レール**と呼ぶ. LUT カスケードはセル間の接続線が前後のセルに限定されていることから, FPGA や ASIC と比較して配線が簡単であり, 遅延時間の見積もりが容易である. LUT カスケードは多出力論理関数を表現する BDD(BDD_for_CF) [3] に対して, 関数分解を繰り返し適用することにより得られる [4]. 従って, 論理合成が比較的容易である.

2.2 LUT カスケード・エミュレータ

LUT カスケードではレールの本数, LUT の段数を一旦決めてしまうと, 実現できる関数は限られてしまう. より柔軟性の高いアーキテクチャとして LUT カスケード・エミュレータが提案されている [2].

図 2 に示す LUT カスケード・エミュレータは順序回路を実現する. **論理メモリ**は LUT カスケードのセルデータを格納する. **プログラマブル接続回路**はセルを読み出すための信号を選択する. **接続メモリ**は接続情報を格納する. 読み出したデータは, **シフタ**を用いてレジスタの所定の位置に格納する. **出力レジスタ**は外部出力を保持する. **状態レジスタ**は状態変数を保持する. 状態レジスタは, 状態入力レジスタ, 状態出力レジスタの 2 つで構成される. 図 3 にプログラマブル接続回路の実現法の一例を示す. LUT カスケード・エミュレータの動作を以下に示す.

1. 外部入力の取り込み: 外部入力をレジスタに取り込む. また, 評価するセルの番号 ← 0 とする.
2. セルのアドレス生成: 接続メモリの接続情報に応じて, プログラマブル接続回路は外部入力とレール出力からセルのアドレスを生成する.
3. セルの読み出し: 論理メモリをアクセスし, セルデータを読み出す. 読み出したデータをシフタで出力レジスタの所定の位置に格納する. 同様にレール出力も取り出す. 最終段のセルでなければ, セルの番号 ← セルの番号 +1 とし, 2. へ戻る.
4. 出力レジスタの値を出力する. また状態出力レジスタの値を状態入力レジスタに取り込み, 1. へ戻る.

上記の各動作を 4 つの状態に割り当てた状態図を図 4 示す. 図の数字は状態変数の値を示す. また, **制御回路**を図 5 に示す. 制御回路はこれら一連の動作を制御する.

制御回路のメモリの内容を表 1 に示す. 制御回路のメモリは, 現在の状態と評価するセルの番号を入力とし, 次の状態と評価するセルの番号, 及び回路の制御信号を出力する. また, 現在の状態と評価中のセルの番号を保持するレジスタを持つ. メモリの内容を書き換えることにより, 様々なカスケードを模擬できる.

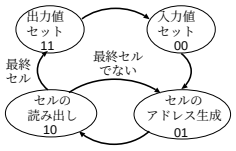

図 4　状　態　図

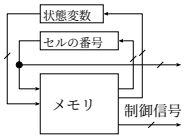

図 5　制 御 回 路

表 1　制御回路のメモリの内容

入力		出力		
現在の状態	セル番号	次の状態	セル番号	制御信号
00	0...00	01	0...00	···
01	0...00	10	0...00	···
10	0...00	01	0...01	···
⋮	⋮	⋮	⋮	⋮
11	1...11	00	0...00	···

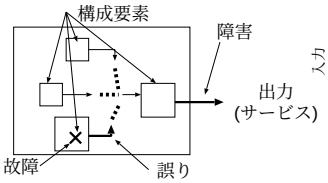

図 6　システムの概念図

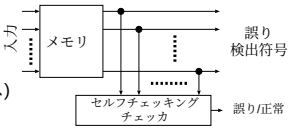

図 7　誤り検出回路

3. LUT カスケード・エミュレータの故障検査

3.1 用語の定義

まず, 本論文で使用する用語について述べる. システムの概念図を図 6 に示す.

[定義 3.1]　システムの出力の異常を**障害 (failure)** と呼ぶ.
[定義 3.2]　構成要素の異常な出力を**誤り (error)** と呼ぶ.
[定義 3.3]　誤りを引き起こす構成要素の物理的な欠陥 (破壊) を**故障 (フォールト)** と呼ぶ.

障害はシステムの構成要素の異常な出力によってもたらされる. 例として, メモリを考えよう. メモリとはアドレス信号を受け取り, アドレスで指定したワードの値を出力するシステムと考えることができる. メモリは, デコーダ, メモリセル, センスアンプなどの構成要素を持つ. メモリセルが故障して 0 値に固定したとする. 本来このセルに 0 値を格納する場合, このセルは故障であるが, 誤りではなく, 障害は発生しない. また, 代替セルを使用することで故障を回避できる. メモリセルに値 1 を保持しているとしよう. 外部からの α 線の影響で電荷を失いセルの値が 0 となった場合, このセルの値は誤りであるが, セル自体は壊れていないので故障とはいわない. この場合, このセルに再度 1 値を書きこむことで誤りを回避可能である. 本論文では誤りと故障を明確に区別し, それらを原因とする障害に対して個々に対処する.

3.2 検査の対象とする故障と誤り(注1)

本論文では検査の対象とする LUT カスケード・エミュレータの故障と誤りを

- 論理メモリの出力の単方向誤り
- 論理メモリのアドレスデコーダ入力線, 及び出力線の単一縮退故障
- 接続回路の出力誤り
- 外部入力信号線の単一縮退故障
- 接続回路のレール部の単一縮退故障

(注1)：メモリでは宇宙線や大きなノイズの影響でデータが変化する可能性がある. これは必ずしも故障ではない. ソフトエラーと呼ばれる.

表 2　Berger(7,3) 符号

情報部	検査部
0000000	111
0000001	110
⋮	⋮
1111111	000

. 接続メモリの出力誤り
. 制御回路の出力誤り

とする．**単方向誤り**とは, 影響を受けるいくつかのビットが 0 → 1, 又は 1 → 0 のいずれか一方向に値を変える単一あるいは多重ビット誤りである．**縮退故障**とは, なんらかの原因で, 論理素子の入出力線の値が 0 又は 1 に固定しまう故障である．**単一故障**とは, 全回路中で故障が一度に一つしか発生しないことを意味する．提案アーキテクチャで検査する故障は, すでに出荷・受け入れテストを通過した正常なシステムの動作中に生じる故障である．論理メモリのセル以外の部分では, 単一縮退故障のみを考える．本論文では, 入力レジスタ, 出力レジスタ, 状態レジスタ, 及び各シフタの故障と誤りは考慮しない．よって, 上記の 2 つの部分は十分にマージンをとって設計する．ただし, これらの部分が回路全体に占める割合はごくわずかであり, 十分動作マージンをとって設計しても回路全体の面積の増加はわずかである．

3.3　論理メモリの故障検査

3.3.1　誤り検出符号

論理メモリは, 誤り検出符号を用いて誤りを検出する．可能な全ての符号ベクトルの集合を V とし, V の部分集合を C とする．C に属する符号ベクトルを**符号語**, $V-C$ に属する符号ベクトルを**非符号語**と呼ぶ．メモリ回路が正常に動作する場合, 符号ベクトルが符号語であり, 誤りが起きた場合, 符号ベクトルが非符号語となるような符号 C を**誤り検出符号**と呼ぶ．

図 7 にメモリの誤りを検出する回路を示す．メモリの各ワードの内容を誤り検出符号で符号化し, その出力をチェッカで常時監視する．チェッカ自身の故障も検出できるように, セルフチェッキング・チェッカを用いる．

3.3.2　順序符号を用いた単方向誤りの検出

［定義 3.4］ (順序関係 $\leqq$) n ビットのベクトルを $\vec{a} = (a_1, a_2, \ldots, a_n)$,$\vec{b} = (b_1, b_2, \ldots, b_n)$ とする．$a_i \leqq b_i (i = 1, 2, \ldots, n)$ が成立するとき, $\vec{a} \leqq \vec{b}$ と記す．

［定義 3.5］ (非順序符号) 任意の異なる二つの符号語の間に順序関係 $\leqq$ が存在しない符号を**非順序符号**という．

［定理 3.1］ メモリの出力を非順序符号で符号化すれば, メモリの各ワードの全ての単方向誤りを検出可能である [5].

(証明) メモリの出力が非順序符号で符号化されているとする．符号語 $\vec{a}$ に 0 → 1 方向誤りが起きた場合, 観測される誤り出力 $\vec{e}$ に対して, 必ず $\vec{a} < \vec{e}$ が成立する．よって, $\vec{e}$ は非順序符号の非符号語であり, 誤り検出回路で検出可能である．1 → 0 方向の場合も同様に証明できる．　(証明終)

3.3.3　Berger 符号

［定義 3.6］ **Berger(k,r) 符号**は, k ビットの情報部と, r ビットの検査部からなり, 検査部は情報部の 1 の個数の 2 進表現の 1 の補数を表現する組織化符号である [5].

表 2 に Berger(7,3) 符号を示す．定義 3.6 より, 検査部のビット数 r は, $r = \lfloor \log_2(k+1) \rfloor$ である．また, 検査部を r ビットのとき, 情報部のビット数 k は, $k = 2^r - 1$ である．情報部のビット数が k とのとき, 2^k 個の符号語が存在するが, 情報部の個数が 2^k 未満の場合は, 検査部のビット数 r を削減できる場合がある．

［定理 3.2］ Berger 符号は非順序符号である [6].

定理 3.1, 3.2 より, メモリ回路の出力を Berger 符号で符号化することにより, 全ての単方向誤りの検出が可能である [11]. Berger 符号は全ての組織符号の中で最も符号長が短い [12].

3.3.4　セルフチェッキング論理回路

メモリ回路の符号化出力はチェッカが常時監視する．本論文では, 自分自身の故障をも検出できるセルフチェッキング論理回路を用いる．本論文で考える故障は, 製造時テストを通過した正常なシステムの動作中に生じる故障である．ここでは, チェッカの故障として以下の仮定を設ける．

［仮定 3.1］ 発生する故障は単一縮退故障のみであり, 故障が発生してから第二の故障発生までの間には, すべての符号語入力が回路に印加されるまでの十分な時間がある．

実際には, 故障が発生してから第二の故障発生までに, 回路を停止させることができれば, 仮定を実現可能である．この仮定の元で, セルフチェッキング回路の最も重要な定義を以下に述べる．

［定義 3.7］ (フォールトセキュア) 故障集合 F に関して**フォールトセキュア**な論理回路とは, F に属する任意の故障が存在するときに, 決して不正な符号語を出力しない論理回路である．

［定義 3.8］ (セルフテスティング) 故障集合 F に関して**セルフテスティング**な論理回路とは, F に属する任意の故障が存在するときに, 少なくとも 1 つの符号語入力に対して非符号語を出力する論理回路である．

［定義 3.9］ (トータリセルフチェッキング) 故障集合 F に関して**トータリセルフチェッキング (TSC: totally self-checking)** な論理回路とは, F に関してフォールトセキュアかつセルフテスティングな論理回路である．

フォールトセキュア論理回路では, 故障が生じても検出不能な誤りは決して出力されないことが保証される．また, セルフテスティング論理回路は, 第二の故障が発生する前に, 必ず検出可能な誤りを出力することで, 第一の故障が検出可能であることを保証する．従って, TSC チェッカを用いることで, 故障を必ず検出可能である．

［定理 3.3］ TSC チェッカの出力線は少なくとも 2 本必要である [6].

定理 3.3 より, チェッカの出力信号線を必要最小数の 2 本とし, c_0, c_1 とする．c_0, c_1 の縮退故障を検出するためには正常時に c_0 と c_1 は共に 0 と 1 の両方の論理値をとらなければならない．従って, チェッカの入力が符号語ならば, (c_0, c_1) は (0,1) 又は (1,0) であり, チェッカの入力が非符号語ならば, (c_0, c_1) は (0,0) 又は (1,1) となる．チェッカは以下に定義する性質を満たす必要がある．

［定義 3.10］ **コードディスジョイント**な論理回路とは, 入力及び出力を誤り検出符号で符号化した場合, 符号語入力に対しては出力も符号語であり, 非符号語入力に対しては出力も非符号語であるような論理回路である．

3.3.5　2 線式符号チェッカ

図 8 に 2 対 2 線式符号に対するセルフテスティング・チェッカを示す．2 対 2 線式符号の符号語では 2 個のビット対のそれぞれが (01) 又は (10) である．従って, 少なくとも 1 つの対が (00) 又は (11) であれば, そのベクトルは非符号語である．この回路の出力が符号語入力に対して (01) 又は (10) であり, 非符号語入力に対して (00) 又は (11) となることは明らかである．すなわち, この回路はコードディスジョイントである．また, この回路のゲート入出力線におけるどんな単一縮退故障に対しても, 4 つの符号語入力 $(x_0x_1, y_0y_1) = (10, 01)$,(01, 10),(10, 01),(10, 10) の少なくとも 1 つによって非符号語が出力される．従って, この回路はセルフテスティングである．2 対 2 線式符号チェッカを基本モジュールとし, それらを木構造に接続することにより, n 対 2 線式符号に対するセルフテスティングチェッカが得られる (図 9).

［定理 3.4］ n 対 2 線式符号チェッカは単一縮退故障に対して

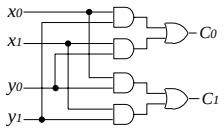

図 8　2 対 2 線式符号に対する
セルフテスティングチェッカ

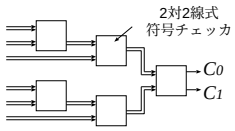

図 9　n 対 2 線式符号チェッカ

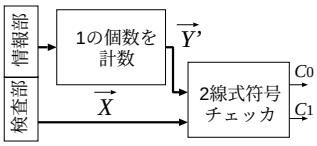

図 10　Berger(k,r) 符号チェッカ

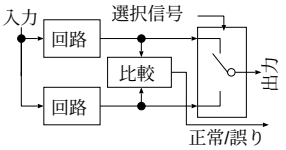

図 11　二重化方式

セルフテスティングである.

(証明) 単一縮退故障が存在するモジュールを M とする. この故障モジュール M は 4 つの符号語入力 $(10,01)$,$(01,10)$, $(10,01)$,$(10,10)$ が印加された場合, 少なくとも 1 つの非符号語 (00) 又は (11) を出力する. M 以外のモジュールはコードディスジョイントであるから, 非符号語はチェッカのモジュールを伝搬し, 非符号語が出力される. (証明終)

3.4　Berger 符号チェッカ

図 10 に Berger(k,r) 符号チェッカの構成を示す. Berger 符号チェッカは, あらかじめ格納した検査部入力ベクトル $\vec{X}$ と, 情報ベクトルから生成される検査部ベクトル $\vec{Y}$ を比較する. 定義 3.6 より, Berger(k,r) 符号の $\vec{Y}$ を生成するには, 情報部ベクトルの 1 の個数を計数し, その補数を求める回路が必要である. しかし実際には, 1 の個数を計数する回路を設計すれば十分である. 符号語入力に対して, 2 線式符号チェッカの入力には n 対の (01) 又は (01) が入力されなければならない. すなわち, 2 線式符号チェッカの入力には $\vec{Y}$ と, $\vec{X}$ を否定演算したベクトルとなるべきである. 図 10 の情報部の 1 の個数を計数する回路は Berger 符号語の検査部ベクトルの否定をとったベクトル $\vec{Y'}$ を生成する. よって, $\vec{X}$ を否定する操作と $\vec{Y}$ を生成する際, 1 の補数をとる操作が打ち消しあって不要になる.

3.5　制御回路と接続メモリの出力誤り検査

制御回路及び接続メモリの出力誤りは, 二重化方式を用いて検出する. 図 11 に二重化方式の概念図を示す. 同一の回路を 2 個用意し, 同じ入力を与え, 出力を比較する. 結果が一致すれば正常とし, 一致しなければ誤りとする. 誤りの時にはシステムを停止し, モニタが故障診断を行ない, 正常な回路に切替えることで故障を回避する (後述).

LUT カスケードのセル数を c, 制御信号数を m とする. 表 1 より, 制御回路のメモリ量 M_{ctrl} は

$$M_{ctrl} = 2^{2+\lceil log_2 c \rceil}(2 + \lceil log_2 c \rceil + m) \quad (1)$$

である. 外部入力数を n, LUT カスケードの最大レール数を l とする. 図 3 より, 外部入力を選択するための MUX の制御入力数は $\lceil log_2 n \rceil$ であり, レール信号を l 個の 1-MUX で選択する. このときの接続メモリ量 $M_{connect}$ は

$$M_{connect} = c(\lceil log_2 n \rceil + l) \quad (2)$$

である. 論理メモリのメモリ量を 1Mbit とし, 文献 [9] の手法を用いていくつかの MCNC ベンチマーク関数を LUT カスケード・エミュレータ上に実現した. そして, $M_{connect}$, M_{ctrl}, 及び論理メモリの必要メモリ量 M_{logic} (Mega bit) を求めた. 結果を表 3 に示す. なお, 制御信号線数は $m = 8$ とした.

LUT カスケードエミュレータでは論理メモリがチップ面積の大部分を占める. 表 3 より, 制御回路と接続メモリの大きさは論理メモリの高々2%程度である. よって, 回路を二重化しても全体の面積増加はごくわずかである.

表 3　各モジュールが使用するメモリの大きさ

関数名	入力数	出力数	c	l	$M_{connect}$ [bit]	M_{ctrl} [bit]	M_{logic} [Mbit]
C432	36	7	4	14	1197	4096	0.968
C1908	33	25	63	13	80	192	1.000
apex7	49	37	13	14	260	896	1.000
rot	135	107	201	10	3618	18432	1.000

c:セル数　l:最大レール数

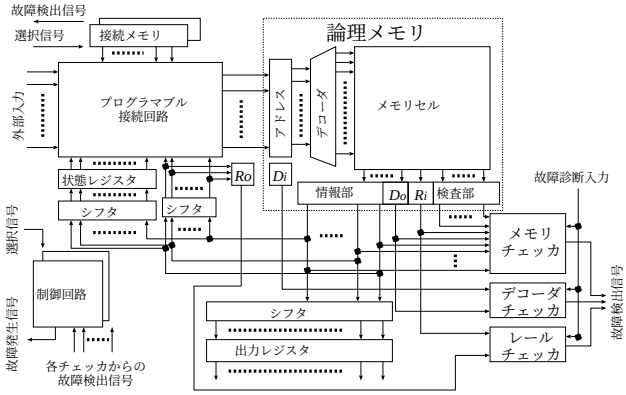

図 12　フォールト・トレラント LUT カスケード・エミュレータ

3.6　フォールト・トレラント LUT カスケード・エミュレータ

図 12 に**フォールト・トレラント LUT カスケード・エミュレータ**を示す. 論理メモリ用のデコーダの単一縮退故障を検出するため, 論理メモリのアドレスのパリティ D_i を計算する回路を付加し, 論理メモリのデータに参照アドレスのパリティ D_o を付加する. そして, **デコーダチェッカ**でチェックする. プログラマブル接続回路のレール部の単一縮退故障を検出するため, レール入力パリティ R_i とレール出力パリティ R_o を**レールチェッカ**でチェックする. R_i, D_o は論理メモリに格納する. 論理メモリの単方向誤りを検出するため, 論理メモリの情報部データと各パリティデータを Berger 符号化したデータに置き換え, **メモリチェッカ**でチェックする. メモリチェッカは図 10 に示した Berger 符号チェッカを用いる. その他のチェッカはパリティをチェックするので, 図 9 に示した 2 線式符号チェッカを用いる. 制御回路と接続メモリは図 11 に示した二重化方式を用いる. **選択信号**は二重化した回路の出力を選択する信号である. 制御回路は次のセルを処理する前に, 接続メモリ, 及び各チェッカからの**故障検出信号**と制御回路自身の故障をチェックする. 故障を検出した場合, 次のセルの処理を停止し, **故障発生信号**を送り, 故障回避モードに移る. **故障診断入力**は故障回避モードで用いる.

4.　故障箇所の特定と故障回避

各チェッカが故障を検出すると, 制御回路は回路を停止させ, 故障発生信号を送り, 故障回避モードに移る. モニタ (組込みプロセッサ) は, 故障診断を行って故障箇所の特定を行う. 次に, 故障箇所に応じて故障回避を行なう.

4.1　故障診断アルゴリズム

[アルゴリズム 4.1] (LUT カスケード・エミュレータの故障診断) 制御回路が動作を停止しているものとする.

1.　チェッカ自身の故障診断: 各チェッカに符号語を印加し, 出力を観測する. 非符号語が出力されれば, チェッカ自身の故障とする.

2.　メモリチェッカによる論理メモリの出力誤り診断: メモリチェッカでメモリ出力をチェックする. 非符号語が出力されればメモリ出力誤りとし, 異常ビットの位置を特定するため, 外部のデータと照合を行う. さらに, 不一致のビットを調べ, メモ

リセルの故障か誤りであるかを診断する.

3. 論理メモリのデコーダの入出力線の故障診断: デコーダチェッカでアドレスパリティ P_i とメモリに格納した P_o を比較する. 一致しなければ, デコーダの故障とする.

4. 制御回路と接続メモリの出力誤りの診断: それぞれの比較器の出力を観測する. 不一致出力が出力されれば, 誤りと診断する.

5. 接続回路の故障診断: まず, レールチェッカでレール出力の重み R_o とメモリに格納した R_i を比較する. 一致しなければ, 接続回路のレール部の故障とする. 上記のどれも該当しなければ, 接続回路の出力誤りと診断する.

チップ面積の大部分が論理メモリのため, 故障の大部分は論理メモリの故障と考えられる. 論理メモリの故障診断法としてフェイル・ビットマップを用いた [7] や [8] が挙げられる. 制御回路や接続メモリが出力誤りと診断された場合は, 正常に動作している回路に切替え, 誤りを回避する.

4.2 メモリパッキングによる故障回避

メモリセルの故障が検出された場合, 故障箇所を回避したメモリパッキングを行い, 故障を回避する. メモリの故障診断法を用いて, 故障が生じた論理メモリのアドレスを検出し, 故障箇所を回避したメモリパッキングを行う. この動作も組込みプロセッサが行う. 以下に故障箇所を回避したメモリパッキングの疑似コードを示す.

[アルゴリズム 4.2] (故障箇所を回避したメモリパッキング)
故障が生じた論理メモリのアドレスは既知とする. 論理メモリのメモリマップを M とし, ワード数を w とする. また, 与えられた論理関数は LUT カスケードで表現されており, LUT カスケードは c 個のセルで実現されているものとする.

1: セル $(i = 1, 2, \ldots, c)$ を入力数の降順にソートする. セルの入力数が同じ場合は, セル出力数の降順にソートする.
2: $k_{min} \leftarrow$ セルの最小入力数.
3: $for(i \leftarrow 1; i \leqq c; i \leftarrow i + 1)\{$
4: $\quad k \leftarrow$ セル $_i$ のセル入力数.
5: $\quad u \leftarrow$ セル $_i$ のセル出力数.
6: $\quad flag \leftarrow 0$.
7: $\quad for(j \leftarrow 1; j < \frac{M}{w} \& flag = 0; j \leftarrow j + 2^{k-k_{min}})\{$
8: $\quad\quad for(l \leftarrow 1; l < w \& flag = 0; l \leftarrow l + 1)\{$
9: $\quad\quad\quad M$ の j から $j + 2^k$ 番地間において, ワード l から $l + u$ 間のメモリセルに故障が無く, かつ他のセルが配置されてないか調べる. 上記の条件を満たせば, セル $_i$ を M にマッピングする. また, $flag \leftarrow 1$ とする.
10: $\quad\quad$ }
11: $\quad$ }
12: $\quad flag = 0$ ならば, M にセル $_i$ が配置できなかったので「メモリパッキング不可能」を返して終了.
13: }
14: メモリマップ M を返して終了.

文献 [9] は, 論理メモリ量制限下で, LUT カスケード・エミュレータの論理メモリマップを生成する手法について述べている. [9] のメモリパッキング部分を, アルゴリズム 4.2 に置き換えるだけで, 故障箇所を回避した論理メモリマップを生成できる. また, デコーダや接続回路の故障が発生した場合, 故障箇所に影響を受けるメモリセルを避け, メモリパッキングを行うことで, これらの故障を回避できる場合もある.

[例 4.1] 図 17 に MCNC ベンチマーク関数 C1908 を 1Mbit(32k×32bit) の論理メモリにメモリパッキングした結果を示す. 灰色で示した部分はセルデータを示す. 空白は未使用部を示す. また, 論理メモリの矩形はメモリセルが $2^{\text{最小セル入力数}} \times 1$ 個含まれている. この例では, 最小セル入力数が 10 個なので, 各矩形辺り $2^{10} \times 1 = 1024$ 個のメモリセルが存在する.

[定義 4.11] (セグメント) 例題 4.1 で説明した論理メモリの矩形を**セグメント**とする.

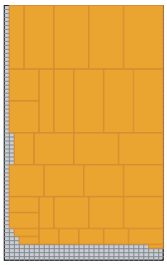

図 13 C1908 の論理メモリ
LUT カスケードのセル:34 個

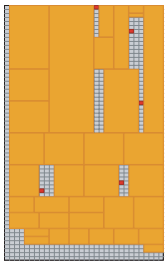

図 14 故障メモリセル:5 個
LUT カスケードのセル:35 個

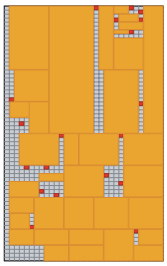

図 15 故障メモリセル:20 個
LUT カスケードのセル:35 個

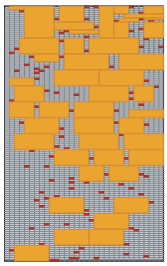

図 16 故障メモリセル:100 個
LUT カスケードのセル:46 個

[例 4.2] 図 17 に MCNC ベンチマーク関数 C1908 を 1Mbit の論理メモリにマッピングした結果を示す.

[定義 4.12] (故障セグメント) **故障セグメント**とは, 故障メモリセルを 1 つ以上有するセグメントである.

[例 4.3] 図 14, 15, 16 は, それぞれ, メモリセル故障を 5, 20, 100 個づつランダムに発生させ, ベンチマーク関数 C1908 を例 4.2 と同じ論理メモリにマッピングした結果を示す. 黒色は故障セグメントを示す. ただし, 同一セグメント内に故障セルが 2 個以上存在する場合, 故障メモリセル数と故障セグメント数が一致しないことに注意.

例 4.3 より明らかなように, 故障メモリセルが増加すると, LUT カスケードでの必要なセル数は増加する.

5. DED(k,q) 符号と Berger(k,r) 符号の比較

二重誤り検出 (Double Error Detecting) 符号はコンピュータのメモリの誤り検出に用いられている. 一方, 本論文では論理メモリの誤り検出に Berger 符号法を用いた. 本章では, DED(k,q) 符号と Berger(k,r) 符号の比較を行う. DED(k,q) 符号は, k ビットの情報部と q ビットの検査部から成る $k+q$ ビットの符号である.

5.1 ハミング符号を用いた二重誤りの検出

DED 符号 $\vec{w}$ は, 検査行列 $\vec{H}$ に対し $\vec{H}\vec{w}^T = 0$ を満足する符号である. ただし, $\vec{w}$ は $k+q$ ビットである. $\vec{k}$ を情報部とする. DED 符号は $\vec{H}\vec{G}^T = 0$ を満足する生成行列 $\vec{G}$ を用いて $\vec{w} = \vec{k}\vec{G}$ で得られる. DED 符号 $\vec{y}$ は $\vec{s} = \vec{y}\vec{H}$ で求められる**シンドローム** $\vec{s}$ を調べることにより故障の検出を行うことが可能である. 特に, 生成行列 $\vec{G}$ の行ベクトルの重みの最小値が 3 のとき, 二重故障の検出を行うことが可能である.

Berger(k,r) 符号や DED(k,q) 符号を用いて論理メモリの誤りを検出する場合, k ビットの情報部に加え, それぞれ r ビット, あるいは q ビットの検査部を必要とする. 表 4 に両符号に必要な検査部のビット数を示す. 表 4 より明らかなように, DED 符号よりも Berger 符号のほうが検査ビット数が少ない. Berger 符号による誤り検出法では, 任意個数の単方向誤りを検出可能である. 一方, DED 符号は高々2 個の双方向誤りしか検出できない.

図 17 に DED 符号を用いた誤り検出回路を示す. DED 符号

表 4　両符号に必要な検査部のビット数

情報部のビット数	Berger 符号	DED 符号
16	5	6
32	6	7
64	7	8
128	8	9

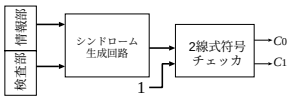

図 17　ハミング符号を用いた二重誤り検出回路

表 5　FPGA 上に実装した結果

	LUT カスケード エミュレータ	フォールトトレラント LUT カスケード・エミュレータ	
		DED 符号	Berger 符号
LE 数	3873(6.7%)	3969(6.9%)	4007(7.0%)
入出力ピン数	130(16.6%)	217(27.7%)	217(27.7%)
メモリ量 (kbit)	1057(20.7%)	1345(26.4%)	1281(25.1%)
動作周波数 (MHz)	44.24	40.64	40.14

の誤りを検出するには, シンドロームを求めればよい. シンドロームの計算は, EXOR ゲートを用いた回路で実行できる. 正しい符号語が入力された場合, シンドロームは全てのビットが 0 のベクトルとなる. シンドロームの各ビットと定数 1 を対にすることで, 2 線式符号チェッカの入力は, 正しい符号語入力の場合 (01) となり, 誤りを含む入力の場合 (11) となる. よって, 2 線式符号チェッカで誤りを検出可能である. また, DED 符号の 2 線式符号チェッカは定数 1 が入力されるので, Berger 符号の 2 線式符号チェッカと比較して, 回路が簡単である.

5.2　FPGA 上に実装した結果

通常の LUT カスケード・エミュレータとフォールト・トレラント LUT カスケード・エミュレータを FPGA 上に実装した結果を表 5 に示す. LUT カスケード・エミュレータとフォールト・トレラント LUT カスケード・エミュレータの入力数は 64, 出力数は 64, 状態変数の個数は 128, 論理メモリ量は 1 [Mega bit](32kbit×32bit), 及びセル数制限は 128 である. また, フォールト・トレラント LUT カスケード・エミュレータは Berger 符号語と DED 符号を用いた場合の二種類の論理メモリの誤り検出回路を実装した. 使用した FPGA は Altera 社の Stratix EP1S60F1020C5 (LE 数: 57120, 入出力ピン数: 782, メモリ量: 5093[kbit]) である. また, 合成ツールは Altera 社の QuartusIIver.5.0. である. FPGA の合成オプションは Standard Fit と Balanced(面積遅延最適化) を用いた.

表 5 に実現結果を示す. フォールト・トレラント LUT カスケード・エミュレータでは LE 数が若干増加した. 各符号のチェッカが全体の回路に占める割合はごくわずかだからである. 入出力ピン数は 53 本増加した. 増加した入出力ピンは, 論理メモリ (アドレス, データ, 出力, 読書き制御) 信号, 故障検出信号, 制御回路・接続メモリの選択信号, 及び各チェッカの出力監視信号である. また, Berger 符号を実装した場合, メモリ量が 21%増加し, DED 符号を実装した場合, メモリ量が 27%強増加した. 各符号の検査部がメモリ量増加のほとんどを占め, 接続メモリの 2 重化によるメモリ量増加はわずかであった. 動作周波数は若干落ちた. フォールト・トレランスを実現するために, 複雑にした制御回路と, メモリチェッカが回路の動作速度に影響を及ぼしたと考えられる. また, 入出力ピン数が増加し, FPGA の配置配線の自由度が落ちたため, 動作周波数が落ちたと考えられる.

表 4 より, DED 符号を用いたほうが必要メモリ量が増大する. FPGA への実装では, メモリ量の差の方が LE 数の差よりも大きかった. よって, チップ面積は Berger 符号を用いた方が DED 符号を用いるよりも小さくなると考えられる.

6.　まとめと今後の課題

本論文では, 耐故障性を有する LUT カスケード・エミュレータについて述べた.

提案アーキテクチャは Berger 符号語を用いて, 論理メモリの多重単方向誤りを検出する. また, デコーダの単一縮退故障, レール部の単一縮退故障, 接続回路の出力誤りを検出する. 制御回路, 接続メモリは多重化し, 誤りを検出する. また, チェッカ自身の故障を検出可能である. 論理メモリの故障, または誤りが検出された場合は, メモリパッキングを行い, 論理メモリの故障箇所を回避して回路を再構成する. 接続メモリ, または制御回路の誤りが検出された場合は, 正常な回路に切替えることで誤りを回避する. FPGA の実装結果から耐故障性を付加するために必要な冗長部分, 及び動作速度の低下は以下の通りである. LE 数はほとんど増加しない. メモリ量は 2 割ほど増加する. 動作速度は若干落ちる.

本論文で用いた, 論理メモリの故障を回避するメモリパッキングは, LSI の製造時に生じた論理メモリの初期不良を回避するのにも利用可能である. よって, 本手法は LSI の歩留りの改善にも利用可能である.

今後の課題として, 本アーキテクチャを用いた耐故障性をもつシステムを実際に設計することが残っている.

7.　謝　　辞

本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文部科学省・北九州地域知的クラスター創成事業の補助金による. 明治大学井口助教授には有益な助言を頂いた.

文　　献

[1] J. M. Rabaey, “Design at the end of the silicon roadmap,” *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, Shanghai, Jan., 2005, Volume 1, pp.18-21.

[2] T. Sasao, Y. Iguchi, and M. Matsuura, “LUT cascades and emulators for realizations of logic functions,” *RM2005*, Tokyo, Japan, Sept. 5-6, 2005, pp.63-70.

[3] P. Ashar, and S. Malik, “Fast functional simulation using branching programs,” *ICCAD’95*, Nov.1995, pp.408-412.

[4] T. Sasao, and M. Matsuura, “A method to decompose multiple-output logic functions,” *Proc. Design Automation Conference (DAC)*, San Diego, CA, USA, June 2-6, 2004, pp.428-433.

[5] 南谷 崇 著『フォールトトレラントコンピュータ』, オーム社, 1991.

[6] 当麻 喜弘, 南谷 崇, 藤原 秀雄 著『フォールトトレラントシステムの構成と設計』, 槇書店, 1991.

[7] J. T. Chen, J. Rajski, J. Khare, O. Kebichi, and W. Maly, “Enabling embedded memory diagnosis via test response compression,” *IEEE International Test Conference*, pp.292-298, 2002.

[8] A. Iseno, Y. Iguchi, and T. Sasao, “Fault diagnosis for RAMs using Walsh spectrum,” *IEICE Trans. Information and Systems*, Vol. E87-D, No.3, March 2004, pp. 592-600.

[9] H. Nakahara, T. Sasao, and M. Matsuura, “A design algorithm for sequential circuits using LUT rings,” *IEICE Trans. Fundamentals of Electronics*, Vol. E88-A, No.12, Dec. 2005, pp. 3342-3350.

[10] S. Yang, “Logic synthesis and optimization benchmark user guide version 3.0,” MCNC, Jan. 1991.

[11] C. Metra, M. Favalli, and B. Ricc’o, “Novel Berger code checker,” *IEEE Proceedings on Defect and Fault tolerance in VLSI systems*, 1995, pp. 287-295.

[12] C. V. Frieman, “Optimal error detection codes for completely asymmetric binary channels,” *Inform. Contr.*, March, 1962, Volume 5, pp. 64-71.