論　文

部分スキャンによる同期化可能な有限状態機械の合成について

四浦　洋[†*]　井上　智生[†]　増澤　利光[†]　藤原　秀雄[†]

On the Synthesis of Synchronizable Finite State Machines with Partial Scan

Hiroshi YOURA[†*], Tomoo INOUE[†], Toshimitsu MASUZAWA[†], and Hideo FUJIWARA[†]

あらまし　本論文では，有限状態機械に部分スキャンを用いて，同期化可能な順序回路を合成する問題について考察する．はじめに，部分スキャンを用いた有限状態機械の同期化の原理を示し，通常入力とスキャン入力による拡張同期化系列を提案する．次に，最小個のスキャン可能な状態変数で有限状態機械を同期化可能とする問題，および，スキャン可能な状態変数の個数が与えられたとき最短の拡張同期化系列を求める問題について考察する．

キーワード　テスト容易化設計，有限状態機械，同期化系列，部分スキャン設計，拡張同期化系列

1. まえがき

順序回路のテスト生成は，多くの時間を要する問題である．テスト容易化設計は，テスト生成複雑度を削減するための重要なアプローチの一つである[1]．主として用いられるテスト容易化設計法に，記憶素子（フリップフロップ；FF）を外部入出力より自由に制御，観測できるように設計を変更するスキャン設計がある．順序回路内のすべてのFFをスキャン可能にするフルスキャン設計[2]は，順序回路のテスト生成問題を組合せ回路の問題に変換できるが，大きなハードウェアオーバヘッドを要する．全FFのうち一部のFFをスキャン可能にする設計を部分スキャンと言う[3],[4]．部分スキャン設計は，小さいハードウェアオーバヘッドでテスト生成の複雑度を小さくすることができるので，テスト容易化設計として有効である．しかしながら，このような典型的なテスト容易化設計は，論理合成における最適化が行われた後に適用されるため，回路面積や性能（遅延）の最適性が損なわれる問題がある．そこで，テスト容易性を考慮した論理合成（テスト容易化論理合成）が考えられるようになった[5]～[8]．

回路の初期化は，順序回路のテスト生成において，多くの時間を要する処理の一つである．初期状態にかかわらず，特定の状態に遷移させることができる入力系列を同期化系列と言う[9]．順序回路が短い同期化系列をもつことは，テスト生成における初期化の処理を大きく助ける[10],[11]．従って，短い同期化系列で同期化可能な順序回路を設計することがテスト容易化論理合成における一つの目標と考えられる．

順序回路は，有限状態機械を用いて設計される．Jiangら[8]は，有限状態機械において部分スキャン操作を考慮することにより，1回のスキャン操作のあとに入力系列を加えることで状態を特定させる方法を提案し，同期化可能な順序回路の合成方法を示した．

本論文では，部分スキャン操作を1回に限定せず，通常入力の適用とスキャン操作を繰り返すことによって有限状態機械を特定の状態に遷移させる拡張同期化を提案する．複数個の通常入力と複数個のスキャン入力からなり，初期状態に依存せず，有限状態機械を特定の状態へ遷移させる入力系列を拡張同期化系列と言う．小さいハードウェアオーバヘッドで，短い拡張同期化系列をもつ順序回路を合成するための問題として，(1) 付加する状態数を最小にする問題，(2) スキャン可能なFFを最小にする問題，(3) 拡張同期化系列長を最小にする問題，を定式化する．問題 (1)，(2) に対して，拡張同期化系列中の通常入力を1種類に限定したもとで，最小個の付加状態数と最小個のスキャン可能なFF数を導く定理を示し，問題 (3) に対して，最短の拡張同期化系列とそれを得るための状態割当てを求めるヒューリスティックアルゴリズムを示す．定理より，MCNC'89[12]のすべてのベンチマーク有限

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630-01 Japan
* 現在，旭化成マイクロシステム株式会社

状態機械について，1個のスキャン可能なFFで拡張同期化可能であることを示す．更に，提案するアルゴリズムによって，MCNC'89のほとんどのベンチマーク有限状態機械について，1個のFFをスキャン可能とするときの最短の拡張同期化系列と状態割当てが得られることを示す．

2. 有限状態機械の同期化

2.1 有限状態機械

本論文では有限状態機械（Finite State Machine; 以下FSMと記す）M を以下のように定義する．

$$M = (S,\ I_n,\ \delta)^{(注1)}$$

但し，S は状態集合，I_n は入力記号の集合（入力集合），δ は状態遷移関数を表す．δ は $I_n \times S$ から S への関数であり，入力記号 $i \in I_n$，状態 $s, s' \in S$ に対して，$s' = \delta(i,\ s)$ は，状態 s のときに，入力記号 i を印加すると状態が s' に遷移することを表す．

入力記号の系列（入力系列）$\tilde{i} \in I_n^+$ に対する状態遷移関数 $\hat{\delta}$ を次のように定義する．

$$\begin{aligned} &|\tilde{i}| = 1 \text{のとき：}\quad \hat{\delta}(\tilde{i},\ s_p) = \delta(\tilde{i},\ s_p) \\ &|\tilde{i}| > 1 \text{のとき：}\quad \hat{\delta}(\tilde{i},\ s_p) = \hat{\delta}(\tilde{i}',\ \delta(i,\ s_p)) \\ &\qquad\qquad \text{但し，} \tilde{i} = i\tilde{i}'(i \in I_n) \end{aligned} \tag{1}$$

また，入力系列 $\tilde{i} \in I_n^+$，状態の部分集合 $S_p \subseteqq S$ に対して，状態遷移関数 $\hat{\hat{\delta}}$ を次のように定義する．

$$\hat{\hat{\delta}}(\tilde{i}, S_p) = \{\hat{\delta}(\tilde{i},\ s_p) | s_p \in S_p\} \tag{2}$$

以下では，特に混乱のない限り，$\delta, \hat{\delta}, \hat{\hat{\delta}}$ を単に δ と表す．入力系列 $\tilde{i} \in I_n^+$，状態の部分集合 $S_c \subseteqq S$ に対して，$\tilde{i}$ を印加すると S_c の状態に遷移する状態の集合を $\delta^{-1}(\tilde{i}, S_c)$ と表す．つまり，δ^{-1} を次のように定義する．

$$\delta^{-1}(\tilde{i},\ S_c) = \{s_p \in S | \delta(\tilde{i},\ s_p) \in S_c\} \tag{3}$$

2.2 FSMの同期化

FSMの同期化，および，同期化系列は次のように定義される．

[定義 1] 同期化，同期化系列

FSM M の初期状態に依存せず，ある特定の状態へ遷移させることを M の同期化と言う．FSM $M = (S, I_n, \delta)$ が $\delta(\tilde{i},\ S) = \{s_r\}$ なる状態 $s_r \in S$ と入力系列 $\tilde{i} \in I_n^+$ をもつとき，M は同期化可能と言い，$\tilde{i}$ を M の同期化系列（synchronizing sequence）と言う．また，同期化系列によって遷移する最終状態 s_r を（$\tilde{i}$ による）リセット状態と呼ぶ．

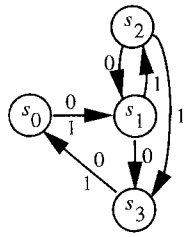

図1 FSM M_1 の状態遷移図
Fig. 1 State diagram of FSM M_1.

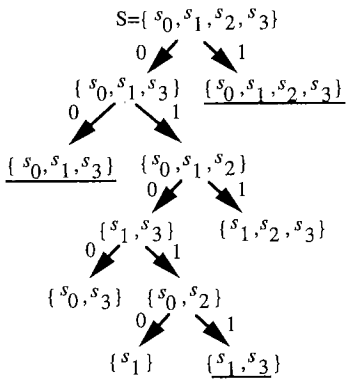

図2 FSM M_1 の同期木
Fig. 2 Synchronizing tree of FSM M_1.

与えられたFSM $M = (S, I_n, \delta)$ の同期化可能性を調べたり，同期化可能な場合に同期化系列を求める問題は同期木を作ることにより解くことができる[9]．同期木の各節は状態集合を表し，根はすべての状態の集合 S を表す．節が表す葉以外の各節は $|I_n|$ 個の子をもつ．子に接続する枝は I_n の入力記号で順にラベル付けされている．状態集合 S_p を表す節に $i \in I_n$ でラベル付けされた枝で隣接する子は集合 $\delta(i, S_p)$ を表す．同じ集合が既に上段に現れている場合や，単一の状態からなる集合に対してはこれを葉とする．同期木において，各節が表す状態集合の要素数をあいまい度と言う．すなわち同期化系列とは $|S| = n$ とすると，S（あいまい度$=n$）から単一の状態からなる集合（あいまい度$=1$）に遷移させるような入力系列である．

[例 1] 図1のFSM M_1 の同期木を図2に示す．

(注1)：本論文ではFSMの出力については考慮しないので，FSMの出力に関する定義は省略する．

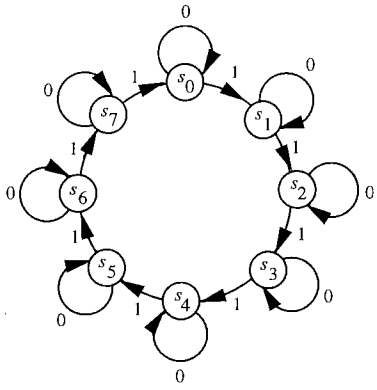

図3 FSM M_2 の状態遷移図
Fig. 3 State diagram of FSM M_2.

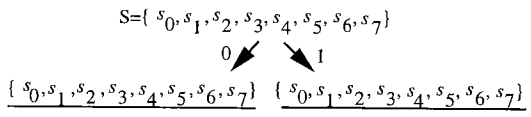

図4 FSM M_2 の同期木
Fig. 4 Synchronizing tree of FSM M_2.

M_1 は S（あいまい度=4）から入力系列 01010 を印加することにより，$\{s_1\}$（あいまい度=1）となり，状態 s_1 に同期化できる（図 2 参照）．すなわち，$\delta(01010,\ S) = \{s_1\}$ であり，入力系列 01010 が M_1 の同期化系列となる．

図 3 の FSM M_2 においては，同期木は図 4 のようになり，M_2 が同期化不可能であることわかる．次節では，同期化不可能な FSM を部分スキャンにより同期化する方法について述べる．

2.3 部分スキャンを用いた拡張同期化

本節ではまず部分スキャンについて説明し，次に部分スキャンを用いて FSM を同期化する方法について説明する．

FSM は状態割当てを行い，順序回路に合成される．状態割当てとは，FSM の状態数を n，順序回路の FF の数を m ($\geq \lceil \log_2 n \rceil$) とすると，FSM の状態を順序回路の状態に対応させるため，FSM の状態を FF に対応する状態変数の値の組で表すことである．部分スキャンとは，いくつかの FF を通常の外部入力（通常入力）とは別に設けられたスキャン入力により，0，1 どちらの値にでも設定できるようにする設計である．この設定をスキャン操作と呼び，スキャン操作可能な FF に対応する状態変数をスキャン可能な状態変数と言う．また，いくつかの FF の値をリセット入力により 0，1 どちらか一方の値にのみ設定できるようにする設計もある．この設定をリセット操作と呼び，リセット操作可能な FF に対応する状態変数をリセット可能な状態変数と言う．

FSM の状態集合を $S = \{s_t | 0 \leq t \leq n-1\}$ とする．FSM は m 個の状態変数 $y_0, y_1, \cdots, y_{m-1}$ をもち，そのうちの k 個がスキャン可能な状態変数とする．一般性を失うことなく，$y_0, y_1, \cdots, y_{k-1}$ をスキャン可能な状態変数とする．スキャン可能な状態変数による S の分割 $\pi_s = \{T_0, T_1, \cdots, T_{2^k-1}\}$ を次のように定義する(注2)．

$$T_i = \{s_t | y_0(s_t) y_1(s_t) \cdots y_{k-1}(s_t) = i\}$$

ここで，$y_h(s_t)$ とは，状態 s_t の状態変数 y_h の値を表し，$y_0(s_t)y_1(s_t)\cdots y_{k-1}(s_t) = i$ とは，$y_0(s_t)y_1(s_t)\cdots y_{k-1}(s_t)$ を 2 進数とみたとき，その値が i と等しいことを表す．このとき π_s をスキャン分割と言い，T_i をスキャン分割ブロックと言う．

同時に，スキャン可能でない状態変数による状態集合 S の分割 $\pi_{ns} = \{P_0, P_1, \cdots, P_{2^{m-k}-1}\}$ を以下のように定義する．

$$P_j = \{s_t | y_k(s_t) y_{k+1}(s_t) \cdots y_{m-1}(s_t) = j\}$$

π_{ns} を非スキャン分割と言い，そのブロック P_j を非スキャン分割ブロックと言う．

π_s と π_{ns} の積について，次式が成り立つ．

$$\pi_s \cdot \pi_{ns} = \{\{s_0\}, \{s_1\}, \cdots, \{s_{n-1}\}\} = \mathbf{0} \qquad (4)$$

以下では，スキャン可能な状態変数 $y_0 y_1 \cdots y_{k-1}$ を i に設定するスキャン操作をスキャン入力記号 i_s を印加すると言う．ここでは，スキャン入力記号は i_s と書き，通常入力記号 i と区別する．

FSM M と状態割当て f により，スキャン入力による遷移が決まる．M をスキャン入力に対して拡張した FSM M^f を定義する．

$$M^f = (S,\ I,\ \sigma_f)$$

但し，状態割当て f によりすべての $s_t \in S$ が状態割当てされているものとし，$I = I_n \cup I_s$（I_n：通常入力集合，I_s：スキャン入力集合），σ_f は状態割当て f に対する状態遷移関数を表し，$I \times S$ から S への関

(注2)：Pomeranz ら [10] は部分リセットを用いる場合について，リセット可能でない状態変数による状態集合 S の分割を提案している．

数である．状態割当て f で定まるスキャン分割ブロック，非スキャン分割ブロックをそれぞれ T_i^f, P_j^f とする．入力記号 $i \in I$ に対して，σ_f を次のように定義する．

$$\sigma_f(i,\ s) \quad = \quad \begin{cases} \delta(i,\ s), & i \in I_n \\ s' \in T_i^f \cap P_j^f, & i \in I_s \end{cases} \qquad (5)$$

但し，$s \in P_j^f$

ここで，状態 $s' \in S$ は状態 $s \in S$ に入力記号 i を印加した後の状態である．

2.4 拡張同期化系列

通常入力とスキャン入力による同期化を特に拡張同期化と呼ぶ．以下に FSM の拡張同期化，および，拡張同期化系列を定義する．

[定義 2] 拡張同期化，拡張同期化系列

状態割当てされた FSM M^f の初期状態に依存せず，スキャン入力と通常入力によりある特定の状態に遷移させることを M^f の拡張同期化と言う．また，M^f の状態集合を S とすると，M^f が $\sigma_f(\tilde{i},\ S) = \{s_t\}$ なる入力系列 $\tilde{i} \in (I_n \cup I_s)^+$ をもつとき，M^f は拡張同期化可能と言い，入力系列 $\tilde{i}$ を M^f の拡張同期化系列（extended synchronizing sequence）と言う．

k 個のスキャン可能な状態変数をもつ FSM M を実現する順序回路に対して，1個のスキャン入力記号 i_s の入力に対応するスキャン操作には k クロック時間を要する(注3)．しかし本論文では，簡単のため，拡張同期化系列の長さをその系列に含まれる入力記号の数とする．

[例 2] 図 3 の FSM M_2 は通常入力だけからなる同期化系列をもたない．そこで，M_2 に対して部分スキャン設計を用いて拡張同期化する．表 1 のように状態割当て f を行い，スキャン可能な状態変数を y_0 とする．M_2 は入力系列 $0_s110_s10_s$ により，状態 s_3 に拡張同期化できる．すなわち，$\{s_3\} = \sigma_f(0_s110_s10_s,\ S)$ であり，入力系列 $0_s110_s10_s$ が拡張同期化系列となる．

長さ $l(\geq 2)$ のスキャン入力記号のみからなる入力系列 $\tilde{i_s} = i_{1s}i_{2s}\cdots i_{ls} \in I_s^+$ による状態集合 $S_p \subseteq S$ の遷移は $\sigma_f(\tilde{i_s},\ S_p) = \sigma_f(i_{ls},\ S_p)$ となる．すなわち，連続するスキャン入力系列 $\tilde{i_s}$ を最後に印加するスキャン入力記号 i_{ls} に置き換えることにより，より短い拡張同期化系列が得られる．従って，以下で扱う拡張同期化系列には，スキャン入力記号が連続して現れることはないものとする．

表1 FSM M_2 の状態割当て f
Table 1 State assignment f of FSM M_2.

	y_2	y_1	y_0
s_0	1	1	0
s_1	0	1	0
s_2	1	0	0
s_3	0	0	0
s_4	0	0	1
s_5	1	0	1
s_6	0	1	1
s_7	1	1	1

Scannable state variable: y_0

2.5 拡張同期化問題

小さいハードウェアオーバヘッドで，短い同期化系列をもつ順序回路を合成するためには，最小数のスキャン可能な状態変数で拡張同期化可能とすること，および，拡張同期化系列を短くすることが重要である．これまでの例では，状態割当てが決まっている FSM を扱ってきたが，FSM に短い拡張同期化系列をもたせるためには，状態の割当て方が重要である．また，状態割当てが決まっている場合，拡張同期化可能にするためには，すべての状態変数をスキャン可能にしなければならない FSM が存在することを示すことができる．従って，ここでは次の三つの拡張同期化問題について考察するが，いずれの問題も与えられる FSM は状態割当てが行われていないものとする．

[問題 1] FSM M が与えられたとき，最小個のスキャン可能な状態変数で M を拡張同期化可能とするための状態割当てを求める．

[問題 2] FSM M とスキャン可能な状態変数の数 k が与えられたとき，最小個の状態付加により M と等価な拡張同期化可能 FSM M' とその状態割当てを求める．

[問題 3] FSM M とスキャン可能な状態変数の数 k が与えられたとき，M が拡張同期化系列をもつかどうか判定し，もつときには最短の拡張同期化系列とその状態割当てを求める．

3. スキャン可能な状態変数の最小化

本章では，前節で定義した問題 1，2 について考察する．

文献 [8] でも，スキャン操作は 1 回のみという仮定のもとで問題 2 を考察しており，FSM M が与えられたとき，等価な状態を追加することにより，1 個のリ

(注3)：k 個のスキャン FF を 1 個のシフトレジスタとして実現したとき．

セット可能な状態変数を用いて長さ2の拡張同期化系列をもつFSM M' を構成する方法を示している．この構成法では，M（状態数を n とする）の一つの通常入力記号に着目し，その入力によって l 個の状態が同一の状態に遷移する場合，$n-2l$ 個の状態を追加することにより M' を構成している．

本章では，以下の定理を示す．この定理は，スキャン操作を複数回行う場合について，同期化可能にするために必要なリセット可能な状態変数の数の上界（問題1），および，k 個のリセット可能な状態変数で同期化可能にするために付加する状態の数の上界（問題2）を与えている．

[定理1] [13] 状態数 n の状態機械の通常入力記号 i_n に関する状態遷移図を $G(i_n)$ とする．また，$G(i_n)$ の弱連結成分の数を $m(i_n)$ と表す．このとき，通常入力記号 i_n とスキャン入力（またはリセット入力）による同期化に関して，以下が成り立つ．

（1） $G(i_n)$ に自己ループが存在しないとき

（a） $n \geq 4$，かつ，$G(i_n)$ が $n/2$ 個の長さ2のサイクルのみからなるとき：

- $n/2$ が偶数の場合，1個のスキャン可能な状態変数では同期化不可能．$n/2$ が奇数の場合，1個のリセット可能な状態変数では同期化不可能であり，1個のスキャン可能な状態変数で同期化可能．また，いずれの場合も，2個のリセット可能な状態変数で同期化可能．
- （等価な）状態を1個追加すれば，1個のリセット可能な状態変数で同期化可能．

（b） その他の場合：1個のリセット可能な状態変数で同期化可能．

（2） $G(i_n)$ に自己ループが存在するとき．

- $(1-\frac{1}{2^{k-1}})n+1 < m(i_n) \leq (1-\frac{1}{2^k})n+1$ なら，$k-1$ 個のスキャン可能な状態変数では同期化不可能であり，k 個のリセット可能な状態変数で同期化可能．
- k 個のスキャン可能な状態変数では同期化不可能なFSMに，状態を $\frac{2^k}{2^k-1}(m(i_n)-1)-n-1$ 個追加しても，k 個のスキャン可能な状態変数では同期化不可能であり，(等価な) 状態を $\frac{2^k}{2^k-1}(m(i_n)-1)-n$ 個追加すれば，k 個のリセット可能な状態変数で同期化可能．

(注1) 定理1において $k=1$ とすれば，(等価な) 状態を $2(m(i_n)-1)-n$ 個追加すれば，1個のリセット可能な状態変数で同期化可能であることわかる．通常入力記号 i_n によって l 個の状態が同一の状態に遷移する場合，$m(i_n) \leq n-l+1$ となり，たかだか $n-2l$ 個の状態を追加することにより1個のリセット可能な状態変数で同期化可能であることがわかる．このことより，同期化可能にするために付加する状態数について，定理1は文献[8]の結果の一般化である．

(注2) 定理1では，2種類以上の入力を利用した同期化を考慮していない．従って定理1では，同期化可能にするために必要なスキャン可能な状態変数の数の上界を与えているが，2種類以上の入力を考慮した場合には，より少ないスキャン可能な状態変数で，同期化できる場合もある（1個のスキャン可能な状態変数で同期化可能にするために付加する状態変数の数についても同様）．但し，たとえ状態遷移図に自己ループを含まないようなFSMであっても，2種類以上の通常入力を利用しても1個のスキャン可能な状態変数では同期化できないものが存在する．図5はその一例である．従って，定理1（1）で示す，同期化可能にするために使用するリセット可能な状態変数の数（2個）は，最悪時を考える限り，最小数である（1個のスキャン可能な状態変数で同期化可能にするために追加する状態変数の数についても同様）．

定理1より，FSMが1個のリセット可能な状態変数で同期化可能なための十分条件は，ある通常入力記号 i_n について以下が成り立つことである．

（1） $G(i_n)$ に自己ループが存在しないとき，$n \leq 3$，または，$G(i_n)$ が長さ2のサイクル以外の弱連結成分を含む．

（2） $G(i_n)$ に自己ループが存在するとき，$m(i_n) \leq n/2+1$．

つまりこの手法において，1個のリセット可能な状態変数では同期化できないのは，どの通常入力記号 i_n に関しても，$G(i_n)$ が $n/2$ 個の長さ2のサイクルからなるか，あるいは，$G(i_n)$ が $n/2+1$ 個より多い弱連結成分からなるときのみである．MCNC'89のベン

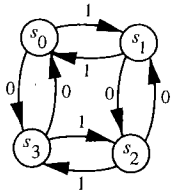

図5 FSM M_3：同期化できないFSMの例
Fig. 5 FSM M_3 : Example of unsynchronizable FSM.

チマーク FSM [12] について確かめたところ，すべての回路がこの十分条件を満たし，1個のリセット可能な状態変数で同期化可能であることがわかった．従って，実際の回路では，この1個のリセット可能な状態変数で同期化できるための十分条件が成り立つと考えられる．

4. 拡張同期化系列の最短化

本章では，問題3について考察する．FSMの状態数を n，状態変数の数を m とすると，状態割当て後のFSMに $2^m - n$ 個の状態変数に対応しない状態が存在する．このような状態を無効状態と言い，その遷移はドントケアである．与えられたFSM M とスキャン可能な状態変数の数 k に対して，最短の拡張同期化系列を求めるためには，(1) 系列を構成する通常入力記号，スキャン入力記号，(2) 状態割当て，(3) 無効状態の遷移，のすべての組合せの中から探索する必要がある．ここでは，効率良く短い拡張同期化系列を求めるために発見的手法を用いた拡張逆順探索法 (Extended Reverse Order Search; EROS) を提案する．EROSは，効率良く同期化系列を求めることのできる逆順探索法 [8] をスキャン入力に対応するように拡張したものである．はじめに逆順探索法について説明する．

[逆順探索法] (Reverse Order Search; ROS)

逆順探索法は発見的手法により，同期木の葉から根にあいまい度が1から n となるまで探索し，同期化系列を求める手法である．各 $s \in S$ に対して，s をリセット状態の候補とする現在の状態集合を $S_c(s) = \{s\}$ とする．各 $S_c(s)$ $(= S_c)$ に対して，$|\delta^{-1}(i_{sct}, S_c)| = max_{i_n \in I_n}(|\delta^{-1}(i_n, S_c)|)$ を満たす通常入力記号 i_{sct} を選択し，S_c から $\delta^{-1}(i_{sct}, S_c)$ に探索を移す $(S_c = \delta^{-1}(i_{sct}, S_c))$．但し，$S_c = \phi$ となったとき，バックトラックする．順次，S_c がすべての状態集合 S となるまで探索を続け，それまでに選択した通常入力記号を選択の逆順にたどると同期化系列となる．

[例3] 図1のFSM M_1 の同期化系列を逆順探索法により求める．リセット状態の候補を s_1 とする．

$$\begin{aligned}
\delta^{-1}(0, \{s_1\}) &= \{s_0, s_2\} \\
\delta^{-1}(1, \{s_0, s_2\}) &= \{s_1, s_3\} \\
\delta^{-1}(0, \{s_1, s_3\}) &= \{s_0, s_1, s_2\} \\
\delta^{-1}(1, \{s_0, s_1, s_2\}) &= \{s_0, s_1, s_3\} \\
\delta^{-1}(0, \{s_0, s_1, s_3\}) &= S
\end{aligned}$$

よって同期化系列は01010となる（図2参照）．

EROSは逆順探索において，選択される入力として通常入力に加えて，スキャン入力も考えることにより，効率良く短い拡張同期化系列を求める方法である．スキャン入力による遷移を決めるためには，状態割当てを行わなければならない．しかし，すべての状態の割当てを一度に行うと，後のスキャン入力による遷移の自由度がなくなってしまう．また，i_s をスキャン入力記号とすると，状態割当て f により，$\sigma_f^{-1}(i_s, S_c)$ が変わってくる．そこで，EROSは状態割当てを必要に応じて順次定めていく．つまり，EROSの実行途中においては，状態割当て f は部分関数である．状態割当て f によるスキャン分割ブロック，非スキャン分割ブロックをそれぞれ T_i^f, P_j^f とする．初期状態割当てを f_0 とすると，$\forall i, j[T_i^{f_0} = \phi, P_j^{f_0} = \phi]$ を満たす（以下これを $f_0 = \phi$ と略す）．i_s が選択されるとき，$\sigma_{f_x}^{-1}(i_s, S_c)$ が定まり，かつ，最大となるように f_c を拡張した f_x を求める．具体的には，現在の状態割当て f_c に対して，スキャン入力記号 i_s を選択したとき，状態割当て f_x は次の条件を満たす．但し，以下では状態割当て f によって状態 s の状態割当てが定まっていないとき，$f(s) = \perp$ と表す．

スキャン入力記号 i_s が選択されたとき，

$S_a = \{s \in S_c | f(s) = \perp\}$

$S_p = \sigma_{f_x}^{-1}(i_s, S_c)$

とすると，

- $\forall i, j[T_i^{f_c} \subseteq T_i^{f_x}, P_j^{f_c} \subseteq P_j^{f_x}]$ (f_x は f_c の拡張.)
- $\forall s \in S_p[f_x(s) \neq \perp]$ (S_p のすべての状態は f_x により状態割当てされている.)
- $\forall s \notin S_a \cup S_p[f_c(s) = \perp \Rightarrow f_x(s) = \perp]$ (f_x は $\sigma_{f_x}^{-1}(i_n, S_p)$ の定義に必要な部分のみ f_c に拡張する.)
- $\forall s \in T_{i_s}^{f_x}[f_c(s) = \perp \Rightarrow s \in S_c]$ (f_x により新たにスキャン分割ブロック $T_i^{f_x}$ に加えられる状態は S_c の状態に限る.)
- $max_{i_n \in I_n}(|\sigma_{f_x}^{-1}(i_n, S_p)|)$ $= max_f(max_{i_n \in I_n}(|\sigma_f^{-1}(i_n, S_p)|))$

(f_x は，$max_{i_n \in I_n}(|\sigma_f^{-1}(i_n, S_p)|)$ が最大になるように定める.)

次に，EROSのアルゴリズム（図6）の流れを説明する．

[拡張逆順探索法] (EROS)

入力：FSM $M^\phi = (S, I_n \cup I_s, \sigma)$,
スキャン可能な状態変数の数 k

出力：拡張同期化系列とその状態割当て f

```
eros(M, k)
/* M: FSM, k: number of scannable state variables */
{
    S = valid_states(M);
    S_inv = invalid_states(M);
    I_n = normal_input_patterns(M);
    I_s = scan_input_patterns(k);
    f_c = φ /* state assignment*/;

    for(∀s ∈ S) {S_c(s) = {s}; ess(s) = ε}
    do{
        for(∀s ∈ S){
            S_c = S_c(s); ess = ess(s);
            i_sct = select_input(S_c, ess);
            ess = i_sct · ess;
            S_c = σ^{-1}(S_c, i_sct);
            if(|S_c| ≦ 1 and S_c is alredy tried) backtrack(s);
        }
    }while(S_c(s_max) ≠ S ∪ S_inv);
    return ess(s_max);
}
select_input(S_c, ess)
{
    i_xn = select_normal_input(S_c);
    if(|S_c| == 1 or previous(i_sct) ∉ I_s){
        i_xs = select_scan_input(S_c);
        if(|S_c| ≧ 2 and S_inv is not included){
            S_c = S_c ∪ S_inv;
            trans_change(S_inv);
        }
        f_x = select_state_assign(S_c);
    }
    if(|σ^{-1}(i_xn, S_c)| > |σ^{-1}(i_xs, S_c)|){
        i_sct = i_xn;
    }else{
        i_sct = i_xs;
        f_c = f_x;
    }
    return i_sct;
}
```

図6 拡張逆順探索法のアルゴリズム
Fig. 6 EROS algorithm.

Step 1 各状態 $s \in S$ について，s をリセット状態の候補とする現在の状態集合を $S_c(s)$，それまでに得られた拡張同期化系列の部分系列を $ess(s)$ とする．初期値として，$S_c(s) = \{s\}$,$ess(s) = \epsilon$ とする（ϵ は，長さ 0 の系列を表す）．

Step 2 各 $S_c(s)$ に対して，以下の更新手続きを行う（以下，$S_c = S_c(s)$，$ess = ess(s)$ として説明する）．

Step 2.1 $|\sigma_{f_c}^{-1}(i_{xn}, S_c)| = max_{i_n \in I_n}(|\sigma_{f_c}^{-1}(i_n, S_c)|)$ を満たす $i_{xn} \in I_n$ を求める (*select_normal_input*()).

Step 2.2 状態集合 $S_c' = \{s \in S_c | f_c(s) \neq \perp\}$ に対して，$|\sigma_{f_c}^{-1}(i_{xs}, S_c')| = max_{i_s \in I_s}(|\sigma_{f_c}^{-1}(i_s, S_c')|)$ を満たす $i_{xs} \in I_s$ を求める (*select_scan_input*()). $|S_c| \geq 2$，かつ，それまでに S_c に無効状態の集合 S_{inv} を加えていなければ $S_c = S_c \cup S_{inv}$ とする．1回前に選択した通常入力記号を i_{pre}，1回更新する前の S_c を S_{pre} とすると，$\sigma(i_{pre}, S_{inv}) = S_{pre}$ とする (*trans_change*()). 状態割当て f_x を選択する．(*select_state_assign*()).

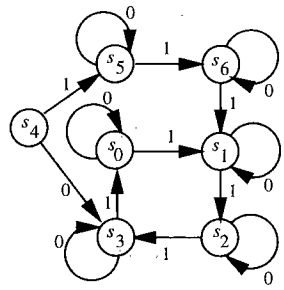

図7 FSM M_4 の状態遷移図
Fig. 7 State diagram of FSM M_4.

Step 2.3 $|\sigma_{f_c}^{-1}(i_{xn}, S_c)|$ と $|\sigma_{f_x}^{-1}(i_{xs}, S_c)|$ を比較し，大きい方の入力記号を i_{sct} として選択する．現在の状態集合 S_c を $\sigma^{-1}(i_{sct}, S_c)$ に更新する．但し，i_{xs} が選択されたときには，状態割当て f_c を f_x に更新する．

Step 2.4 $|S_c| \leq 1$，または，S_c がそれまでに S_c として現れているとき，バックトラックする (*backtrack*()). バックトラックする入力と状態割当てのすべての組合せをつくしたとき，リセット状態の候補 s に関して拡張同期化不可能とする．

Step 3 $|S_c(s_{max})| = max_{s \in S}(|S_c(s)|)$ を満たす s_{max} に関して，$S_c(s_{max})$ が無効状態を含むすべての状態集合 $S \cup S_{inv}$ であるとき，それまでの入力系列を選択の逆順にたどった系列を拡張同期化系列として終了．もし，いずれの状態 $s \in S$ についても拡張同期化不可能ならば，FSM M は同期化不可能として終了．さもなければ，Step 2 へ．

[例 4] 図7の FSM M_4，スキャン可能な状態変数の数1が与えられたとする．無効状態を s_7 とする．また，$I_n = \{0, 1\}, I_s = \{0_s, 1_s\}$ である．ここでは，リセット状態の候補を s_1 のときの拡張逆順探索法について説明する．

（1） $S_c = \{s_1\}$，$ess = NILL$，$f_c = \phi$.

（2） $|\sigma_{f_c}^{-1}(0, S_c)| = 1$，$|\sigma_{f_c}^{-1}(1, S_c)| = 2$，また，$max_f(max_{i_s \in I_s}(|\sigma_f^{-1}(i_s, S_c)|)) = 2$ である．$1 \in I_n$ を選択する．$S_c = \{s_0, s_6\}$.

（3） $|\sigma_{f_c}^{-1}(0, S_c)| = |\sigma_{f_c}^{-1}(1, S_c)| = 2$ である．ここで，S_c に無効状態 s_7 を加え，$1 \in I_n$ に対して，s_7 から s_1 に遷移するものとする．$f = \phi$ で

あるから，$0_s, 1_s$ のうち 0_s を選択する．状態割当てを f_x は $T_0^{f_x} = \{s_0, s_6, s_7\}, T_1^{f_x} = \{s_1, s_2, s_3\}$ $P_0^{f_x} = \{s_0, s_1\}, P_1^{f_x} = \{s_6, s_2\}, P_2^{f_x} = \{s_7, s_3\}$ とする．$\sigma_{f_x}^{-1}(0_s, S_c) = \{s_0, s_1, s_2, s_3, s_6, s_7\}$ となる．$0_s \in I_s$ を選択し，f_c を f_x に更新する．$S_c = \{s_0, s_1, s_2, s_3, s_6, s_7\}$，$ess = 0_s1$.

（4） ここでは通常入力による遷移のみを考える．$|\sigma_{f_c}^{-1}(0, S_c)| = |\sigma_{f_c}^{-1}(1, S_c)| = 6$ である．$0 \in I_n$ を選択する．$S_c = \{s_0, s_1, s_2, s_3, s_4, s_6\}$，$ess = 00_s1$.

（5） $\forall i, j[T_i^{f_c} \subseteqq T_i^{f_x}, P_i^{f_c} \subseteqq P_i^{f_x}]$，$s_5 \in T_0^{f_x}$，$s_4 \in T_1^{f_x}$ に $P_3^{f_x} = \{s_4, s_5\}$ なる状態割当て f_x に対して，$\sigma_{f_x}^{-1}(1_s, S_c) = S \cup S_{inv}$ となる．$1_s \in I_s$ を選択し，f_c を f_x に更新する．$S_c = S \cup S_{inv}$，$ess = 1_s00_s1$.

以上より M_4 の拡張同期化系列は 1_s00_s1 となる．

5. 実験結果

MCNC'89のベンチマークFSMを用いて，EROSの有効性を示すための実験を行った．Jiangらの手法[8]はスキャン入力記号がはじめの1個のみの拡張同期化系列を求める手法と考えることができる．そこで，本実験ではスキャン可能な状態変数の数が1のときの拡張同期化系列の長さを，Jiangらの手法の結果[8](Jiang)と比較している（表2）．表は左からベンチマークFSMの名前，FSMの状態数，入力ビット数，拡張同期化系列の長さを最小値，Jiangらの手法の結果，EROSの結果の順に示している．入力ビット数はその値を l とすると，2^l 個の通常入力記号が存在することを表す．また，最小値とは，EROSの前探索により求められた，スキャン可能な状態変数の数が1個のときの最短の拡張同期化系列の長さであり，計算時間が24時間以上であるものについては探索をうちきった．すべての回路でJiangらの手法と等しいか，より短い系列が得られた．*planet*，*planet*1など，Jiangらの手法で4以上の系列長の回路に対して特に短い系列が得られている．これは，EROSにより求められた拡張同期化系列中のスキャン入力記号は，Jiangらの手法で系列長3以下の回路に対しては，Jiangらの手法と同じく1個であるのに対し，4以上の回路に対してはスキャン入力記号が複数個存在し，スキャン操作を繰り返す効果が現れるためと考えられる．また，Jiangらの手法では同期化できない回路（$modulo12, tav$）が，EROSにより同期化可能である．EROSにより求められた系列長は，すべて求められた最小値と一致しており，EROSに用いた発見的手法は有効であると考えられる．

表2 MCNC'89ベンチマークFSMの拡張同期化系列を求める実験結果（スキャン可能な状態変数の数：1）
Table 2 Experimental Results for MCNC'89 benchmark FSMs by one scannable state varible.

FSM	状態数	入力 (bit)	拡張同期化系列の長さ		
			最小値	Jiang	EROS
bbtas	6	2	2	2	2
dk16	27	2	N/A	3	3
dk17	8	2	2	2	2
dk27	7	1	2	2	2
dk512	15	1	N/A	3	3
ex1	20	9	2	2	2
ex2	19	2	2	2	2
ex3	10	2	2	2	2
ex4	14	6	N/A	5	4
ex5	9	2	2	2	2
ex7	10	2	2	2	2
lion	4	2	2	2	2
lion9	9	2	2	2	2
mc	4	3	2	2	2
modulo12	12	1	N/A	†	6
planet	48	7	N/A	8	4
planet1	48	7	N/A	8	4
s1	20	8	2	2	2
s1a	20	8	2	2	2
sand	32	11	N/A	5	4
shiftreg	8	1	3	3	3
styr	30	9	2	2	2
train11	11	2	2	2	2
tav	4	4	3	†	3

†：同期化不可能，　N/A：計算時間24時間以上

6. むすび

本論文では，有限状態機械に部分スキャンを用いて，同期化可能な順序回路を合成する問題について考察した．通常入力とスキャン入力を組み合わせて有限状態機械を同期化する拡張同期化系列を提案し，拡張同期化問題について考察した．その結果として，有限状態機械を拡張同期化するために必要なスキャン可能な状態変数の個数に関する定理，および，スキャン可能な状態変数の個数が与えられたとき，最短の拡張同期化系列を求める拡張逆順探索法を提案した．

謝辞 本研究に関し，多くの貴重な意見を頂いた井上美智子助手，高畠勝之氏はじめ情報論理学講座の皆様方に感謝致します．

文　献

[1] H. Fujiwara, "Logic Testing and Design for Testability." Cambridge, MA: The MIT Press, 1985.

[2] E.B. Eichelberger and T.W. Williams, "A logic design structure for LSI testing," Proc. 14th Design Automation Conf., pp.462-468, June 1977.

[3] V.D. Agrawal and K.T. Cheng, "A complete solution to the partial scan problem," Proc. Int. Test Conf., pp.44–51, 1987.
[4] V. Chickermane and J.H. Patel, "An optimization based approach to the partial scan design problem," Proc. Int. Test Conf., pp.377–386, 1990.
[5] S. Devadas, H.-K.T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "A synthesis and optimization procedure for fully and easily testable sequential machines," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol. 8, no. 10, pp.1100–1107, Oct. 1989.
[6] B. Eschermann and H.-J. Wunderlich, "Optimized synthesis techniques for testable sequential circuits," IEEE Trans. Comput.-Aided Des. Intergrated Circuits & Syst., vol.11, no.3., pp.301–312, 1992.
[7] V.D. Agrawal and K.-T. Cheng, "Finite state machine synthesis with embedded test function," Jour. Electronic Testing: Theory and Applic., 1, pp.221–228, 1990.
[8] N. Jiang, R.M. Chou, and K.K. Saluja, "Synthesizing finite state machines for minimum length synchronizing sequence using partial scan," Dig. 25th Int. Symp. Fault-Tolerant Comput. vol.25, pp.41–49, June 1995.
[9] F. Hennie, "Finite State Models for Logical Machines," New York: John Wiley, 1968.
[10] I. Pomeranz and S. Reddy, "On the role of hardware reset in synchronous sequential circuit test generation," IEEE Trans. Comput., vol.43, no., pp.1100–1105, Sept. 1994.
[11] K.-T. Cheng and V.D. Agrawal, "Initializability consideration in sequential machine synthesis," IEEE Trans. Comput., vol.41, no.3, pp.374–379, March 1992.
[12] S. Yang, "Logic synthesis and optimization benchmarks user guide version 30," Microelectronics Center of North Carolina, Research Triangle Park, NC, Tech. Rep, Jan. 1991.
[13] T. Masuzawa, T. Inoue, H. Youra, and H. Fujiwara, "One resettable state variable is almost sufficient for synthesizing synchronizable FSMs," 奈良先端科学技術大学院大学情報科学研究科技術報告書, NAIST-IS-TR96014 (ISSN:0919-9527), 1996 (URL: http://isw3.aist-nara.ac.jp/IS/TechReport2/report/ 96014.ps として入手可能).

(平成8年3月4日受付, 7月10日再受付)

四浦　洋

平6阪大・基工・情報卒. 平8奈良先端大博士前期課程了. 現在, 旭化成マイクロシステム(株)設計開発センターにてVLSIの設計開発に従事.

井上　智生　(正員)

昭63明治大・工・電子通信卒. 平2同大大学院博士前期課程了. 同年松下電器産業(株)入社, 明治大大学院博士後期課程を経て, 現在奈良先端大情報科学研究科助手. テスト生成, 並列処理, テスト容易化設計の研究に従事. IEEE, 情報処理学会各会員.

増澤　利光　(正員)

昭57阪大・基礎工・情報卒. 昭62同大大学院博士課程了. 阪大情報処理教育センター助手, 阪大基礎工学部助教授を経て, 現在奈良先端大情報科学研究科助教授. 分散アルゴリズム, 並列アルゴリズム, テスト容易化設計の研究に従事. 工博. ACM, IEEE, EATCS, 情報処理学会各会員.

藤原秀雄　(正員)

昭44阪大・工・電子卒. 昭49同大大学院博士課程了. 阪大工学部助手, 明治大理工学部教授を経て, 現在奈良先端大情報科学研究科教授. 論理設計, テスト容易化設計, テスト生成, 高位合成, フォールトトレランスの研究に従事. 著書「Logic Testing and Design for Testability」(MIT Press)など. 工博. IEEE Fellow.