IEEE Asian Test Symposium, pp. 130-135, November 1996.

An Approach to the Synthesis of Synchronizable Finite State Machines with Partial Scan

Tomoo Inoue, Toshimitsu Masuzawa, Hiroshi Youra, and Hideo Fujiwara
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-01, Japan

Abstract

Initialization of sequential circuits is one of time-consuming processes in test generation for sequential circuits, and hence synthesizing sequential circuits of which synchronizing sequences are short is an important approach to reducing the cost of test generation for the circuits. In this paper, we propose an approach to the synthesis of finite state machines (FSMs) with partial scan. We focus on repeating partial scan for synchronizing FSMs, and present an ***extended synchronizing sequence*** *which consists of scan inputs and normal inputs, and which takes a circuit to a single specific state, regardless of the initial state. To synthesize synchronizable FSMs, we formulate a problem of minimizing extended synchronizing sequence length, and present a heuristic algorithm for the problem. We show the experimental results of the minimization of extended synchronizing sequence length on MCNC'91 benchmark FSMs. The experimental results show that the proposed heuristic algorithm can find a minimum-length extended synchronizing sequence for most of MCNC'91 benchmark FSMs, and the length of the extended synchronizing sequence is three or less for all the benchmark FSMs.*

1 Introduction

Test generation for sequential circuits is a time-consuming problem [1]. *Design for testability* (DFT) is an important approach to reducing the complexity of test generation. A popular DFT method is *scan design*, which refers to making memory elements (flip-flops) controllable and observable via extra primary ports. Although *full scan* design [2] referring to making all filp-flops scannable in a circuit can transform the sequential test generation problem to the combinational one, large hardware overhead is required. The design such that a subset of filp-flops are scannable is said to be *partial scan* [3, 4]. This DFT approach can reduce the complexity of test generation with small overhead. However, conventional DFT methods by using scan design is applied to logic-synthesized circuits, and consequently the optimized area and/or performance (delay) of the circuits are degraded. Hence, testability is also taken into account *during* the synthesis of the circuits (*synthesis for testability*) [5, 7, 6, 10].

Initialization of circuits is one of time-consuming processes in test generation for sequential circuits. An input sequence such that when applied to a circuit, it will drive the circuit to a single specific state, regardless of the initial state is called a *synchronizing sequence* [8]. When a circuit has a short synchronizing sequence, it is easy to initialize the circuit in the process of test generation for the circuit. Therefore, it can be considered that designing sequential circuits having short synchronizing sequences is an approach to synthesis for testability.

A sequential circuit is modeled by a finite state machine (FSM). Cheng and Agrawal [9] presented the conditions for initializability of FSMs and an automatic state assignment algorithm for logic minimality and initializability. Jiang *et al.* [10] focused on the fact that partial scan can reduce the *ambiguity* of the states of FSMs, and presented a method of synchronizing FSMs after partial scan. Pomeranz and Reddy [11] considered the effect of repeating *partial reset* on test generation.

In this paper, we consider repeating partial scan for synchronizing FSMs, and present an *extended synchronizing sequence* which consists of scan inputs and normal inputs, and which drives a circuit to a single specific state, regardless of the initial state. To generate synchronizable FSMs with small hardware overhead, we formulate a problem of minimizing extended synchronizing sequence length and present a heuristic algorithm to solve the problem. We show the experimental results of the minimization of extended synchronizing sequence length on MCNC'91 benchmark FSMs [12]. The experimental results show that the proposed heuristic algorithm can find a minimum-length extended synchronizing sequence for most of MCNC'91 benchmark FSMs, and the length of the extended synchronizing sequence is three or less for all the benchmark FSMs.

2 Preliminaries

2.1 Finite State Machines

The behavior of a sequential circuit is modeled by a finite state machine (FSM). An FSM M can be defined by a triple [1]:

$$M = (S, I, \delta) \tag{1}$$

where S is the set of states, I is the set of inputs, and δ is the state transition function which maps states and inputs into states, i.e., $\delta : S \times I \to S$. When an input $i \in I$ takes FSM M from a *current* state $s \in S$ to the *next* state $q \in S$, the state transition is expressed as

$$q = \delta(s, i). \tag{2}$$

[1]In general, an FSM is defined by a six-tuple which also includes the set of initial states, the output function and the set of outputs. However, those are suppressed from the definition because they play no role in the subsequent discussion.

The state transition function δ is normally extended to a function δ^* of $S \times I^* \rightarrow S$ as follows:

$$\begin{array}{l}\delta^*(s,\epsilon) = s \\ \delta^*(s,xi) = \delta(\delta^*(s,x),i)\end{array} \tag{3}$$

where $s \in S, x \in I^*, i \in I$, and ϵ denotes an empty sequence of inputs. Moreover, we define the following function for an input sequence $x \in I^*$ and for a subset $Q \subseteq S$:

$$\hat{\delta}^*(Q,x) = \{\delta(q,x) | q \in Q\}. \tag{4}$$

In the rest of this paper, the extended function δ^* is simply denoted by δ if no confusion occurs.

For a subset $Q \subseteq S$ and an input sequence $x \in I^*$, we express the set of states $Q' \subseteq S$ such that $Q = \delta(Q', x)$ as $\delta^{-1}(Q, x)$, i.e.,

$$\delta^{-1}(Q,x) = \{s \in S | \delta(s,x) \in Q\}. \tag{5}$$

2.2 Synchronizing Sequences

Definition 1(Synchronizing sequence): An input sequence of an FSM M is a *synchronizing sequence* for M if the input sequence takes FSM M to a certain final state, regardless of the initial state of FSM M. The final state to which a synchronizing sequence takes an FSM M is called the *reset state* of the synchronizing sequence for M. An FSM M is said to be *synchronizable* if FSM M has a synchronizing sequence. $\square$

A synchronizing sequence for an FSM $M = (S, I, \delta)$ can be derived from the state transition diagram by constructing the *synchronizing tree* in which node represents the uncertain states of FSM M. First, we make the root node which represents the set of all states S. For each input $i \in I$, an edge is created to a new child. The new child corresponds to the set of states to which input i takes FSM M from the set represented by its parent, i.e., $\delta(S, i)$. This procedure is repeated for each new node, and a node becomes a leaf when the node corresponds to a single state, or is identical to some other node in the tree. After the synchronizing tree is constructed, a synchronizing sequence can be found as any path in the tree from the root to a leaf containing a single state. In the synchronizing tree, the number of states represented by a node is called the *ambiguity* of the node.

Example 1: Consider FSM M_1 illustrated in Fig. 1(a). The set of states S of FSM M_1 is $\{s_0, s_1, s_2, s_3\}$. Fig. 1(b) shows the synchronizing tree of FSM M_1. As shown in this figure, FSM M_1 is synchronized by the synchronizing sequence $(0, 1, 0, 1, 0)$, and the reset state is s_1. $\square$

As shown in the above equation, a synchronizing sequence is an input sequence that reduces the ambiguity from the total number of states to 1.

Example 2: Fig. 2(b) shows the synchronizing tree of FSM M_2 illustrated in Fig. 2(a). As shown in Fig. 2(b), there is no leaf node representing a single state in the synchronizing tree, i.e., FSM M_2 is not synchronizable. $\square$

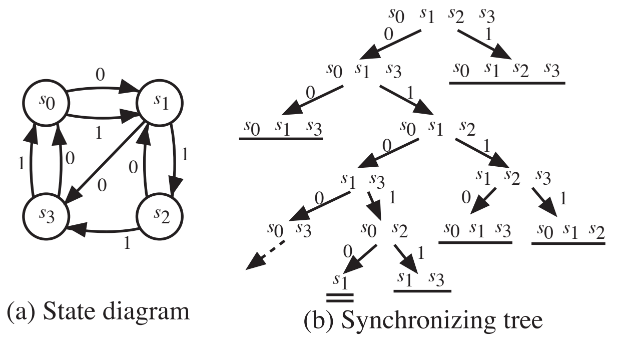


Figure 1: FSM M_1

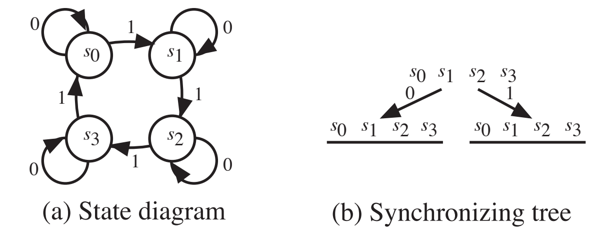


Figure 2: FSM M_2

2.3 Encoded FSMs

When an FSM is implemented by a sequential circuit, each state of the FSM is assigned to a unique binary code word. This assignment is said to be *state encoding* or *state assignment*. The value (or content) of a flip-flop in a sequential circuit is represented by a *state variable*. Note that m state variables represent 2^m states. Hence, when the set of states S of an FSM M is encoded with m state variables, $2^m - |S|$ states represented by m state variables do not correspond to any state of FSM M. Such states are called *invalid states*.

Partial scan design of a sequential circuit refers to chaining a subset of flip-flops into a shift register (scan chain). The flip-flops in the scan chain are said to be *scannable* flip-flops. Each of scannable flip-flops can be set to any value of 0 and 1 by shifting a single-bit sequence into an extra input port (scan-in). We consider a bit sequence applied to scannable flip-flops in a sequential circuit as an input of the FSM corresponding to the sequential circuit. Such an input is called a *scan-input*. The state variables corresponding to scannable flip-flops are called *scannable state variables* (or *scannable variables* for short), and the other state variables are called *non-scannable state variables* (or *non-scannable variables* for short).

The state encoding of the set of states S with k scannable variables and $m - k$ non-scannable variables is defined as follows.

Definition 2(State encoding): Let S be the set of states of FSM M. Let $m(\geq \lceil \log_2 |S| \rceil)$ be the number of state variables. Let k be the number of scannable variables $(0 \leq k \leq m)$. Let S^e be the *encoded state set* such that

$$S^e = S \cup S_{inv} \tag{6}$$

where S_{inv} is the set of invalid states, which satisfies

$$|S^e| = 2^m,\ S \cap S_{inv} = \phi. \tag{7}$$

Let $f_s : S^e \to B^k$ be the *scan encoding function* of S with k scannable variables, where $B = \{0, 1\}$. Let $f_n : S^e \to B^{m-k}$ be the *non-scan encoding function* of S with $m - k$ non-scannable variables.

A state code word a of a state $s \in S^e$ is expressed as

$$a = (f_s(s), f_n(s)). \tag{8}$$

□

Here, we consider the two partitions over an encoded state set S^e by scan and non-scan encoding functions, f_s and f_n as follows.

The *scan partition* over the encoded state set S_e by the scan function f_s is

$$\pi_s = \{T^{f_s}_{a_s} | a_s \in B^k\} \tag{9}$$

where $T^{f_s}_{a_s}$ is called a *scan partition block* which is expressed as

$$T^{f_s}_{a_s} = \{s | f_s(s) = a_s\}. \tag{10}$$

The *non-scan partition* over the encoded state set S_e by the non-scan function f_n is

$$\pi_n = \{P^{f_n}_{a_n} | a_n \in B^{m-k}\} \tag{11}$$

where $P^{f_n}_{a_n}$ is called a *non-scan partition block* which is expressed as

$$P^{f_n}_{a_n} = \{s | f_n(s) = a_n\}. \tag{12}$$

Note that

$$\pi_s \cdot \pi_n = \mathbf{0}, \tag{13}$$

where $\mathbf{0}$ is the *zero partition* in which every block is a singleton.

Thus, we define an *encoded FSM* of an FSM as follows.

Definition 3(Encoded FSMs): Let $M = (S, I, \delta)$ be an FSM. Let $m (\geq \lceil \log_2 |S| \rceil)$ be the number of state variables. Let k be the number of scannable variables $(0 \leq k \leq m)$. Let S^e be the encoded state set of S. Let $f_s : S^e \to B^k$ be the scan encoding function. Let $f_n : S^e \to B^{m-k}$ be the non-scan encoding function.

FSM M with the encoding functions f_s and f_n is said to be a (f_s, f_n)*-encoded FSM* M^e of FSM M, and is defined by

$$M^e = (S^e, I^e, \delta_e). \tag{14}$$

The set of inputs I^e is

$$I^e = I \cup I_s, \tag{15}$$

where the set of inputs of FSM M, I, is referred to as the set of *normal-inputs*, and I_s is the set of scan-inputs, which is given by

$$I_s = B^k. \tag{16}$$

The state transition function δ_e satisfies

$$\forall s \in S,\ \forall i \in I,\ \delta_e(s, i) = \delta(s, i), \tag{17}$$

and

$$\forall s \in S^e,\ \forall i \in I_s,\ \delta_e(s, i) = T^{f_s}_{i} \cap P^{f_n}_{f_n(s)}, \tag{18}$$

where $T^{f_s}_{a_s}$ and $P^{f_n}_{a_n}$ are the scan partition block of $a_s \in B^k$ by f_s and the non-scan partition block of $a_n \in B^{m-k}$ by f_n, respectively. □

For a scan-input sequence $x_s = i_1 i_2 ... i_n \in I^+_s$, the state transition function of an encoded FSM $M^e = (S^e, I^e, \delta_e)$, δ_e satisfies the following property.

$$\forall s \in S^e,\ \delta_e(s, x_s) = \delta_e(s, i_n). \tag{19}$$

As shown in Equation (18), FSMs with partial scan have a state transition by scan-inputs, and the state transition is defined by state encoding. Next, we consider a method for synchronizing FSMs using scan-inputs.

2.4 Extended Synchronizing Sequence

Definition 4(Extended synchronizing sequence): An input sequence consisting of normal-inputs and scan-inputs of an encoded FSM M^e is an *extended synchronizing sequence* for M^e if the input sequence takes a certain final state, regardless of the initial state of encoded M^e. The final state to which an extended synchronizing sequence takes an encoded FSM M^e is called the *reset state* of the extended synchronizing sequence for M^e. An encoded FSM M^e is *extended-synchronizable* if encoded FSM M^e has an extended-synchronizing sequence. □

In the same way as synchronizing trees, the *extended synchronizing trees* can be constructed for an encoded FSM, and the extended synchronizing sequence can be derived from it.

Example 3: Consider FSM M_2 shown in Fig. 2 again. This FSM M_2 has no synchronizing sequence consisting only of normal-inputs. Consider the state encoding shown in Fig. 3(a), where two state variables are assigned and one of the two state variables is scannable. Fig. 3 shows the extended synchronizing tree of encoded FSM M^e_2 of FSM M_2. From Fig. 3, we can see that the encoded FSM M^e_2 is extended-synchronizable with the sequence $(\underline{1}, 1, \underline{0})$ and can be taken to a reset state s_0, where an underlined input denotes a scan-input. □

3 Minimization of ESS length

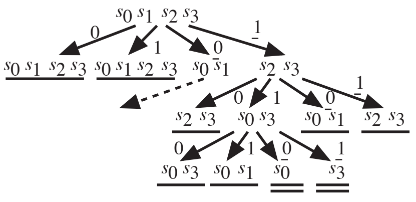
3.1 Problem Formulation

As mentioned in the previous section, an extended synchronizing sequence (ESS) which includes scan-inputs can synchronizing FSMs. In this section, we consider the synthesis of an extended-synchronizable encoded FSM of an FSM.

In synthesizing encoded FSMs, the state assignment and the transitions from invalid states are determined. Determining the state assignment and the transitions from invalid states play important roles in

s	$f_s(s)$	$f_n(s)$
s_0	0	0
s_1	0	1
s_2	1	1
s_3	1	0

(a) State encoding



(b) Extended synchronizing tree

Figure 3: Extended synchronization of FSM M_2

synthesizing an encoded FSM having a short ESS as well as optimizing area and/or performance of the sequential circuit which implements the FSM. In [13], we have considered two problems concerning extended-synchronizable encoded FSMs: (1) the minimization of the number of scannable variables to obtain an extended-synchronizable encoded FSM, and (2) the minimization of the number of extra states to obtain an extended-synchronizable encoded FSM. In this paper, we consider the following problem.

Minimum-length ESS problem:

Given: An FSM $M = (S, I, \delta)$, the number m of state variables and the number k of scannable variables.

Solution: An (f_s, f_n)-encoded FSM $M^e = (S^e, I^e, \delta_e)$ of FSM M having the minimum-length ESS, where $f_s : S^e \to B^k$ and $f_n : S^e \to B^{m-k}$. □

3.2 Algorithm

We present a heuristic algorithm *min_ess* for the minimum-length ESS problem. This algorithm is based on the reverse-order-search (ROS) [10] algorithm. Fig. 4 shows details of the algorithm *min_ess*.

A given FSM $M = (S, I, \delta)$ is considered to be a partially-defined encoded FSM $M^e = (S^e, I^e, \delta_e)$ such that

$$\begin{cases} S^e = S \cup S_{inv} \\ I^e = I \cup I_s \\ \forall s \in S,\, \forall i \in I,\, \delta_e(s,i) = \delta(s,i) \\ \forall s \in S^e, \forall i \in I_s,\, \delta_e(s,i) = \perp \\ \quad (\text{i.e., } \forall s \in S^e,\, f_s(s) = f_n(s) = \perp) \\ \forall s \in S_{inv}, \forall i \in I,\, \delta_e(s,i) = \perp \end{cases}$$

where $\perp$ denotes undefined. The *min_ess* algorithm starts from a state $s_c \in S$ as a candidate of the reset state to which the minimum-length ESS takes M^e. Then, *min_ess* estimates the set of states from which an input $i \in I^e$ takes M^e to the candidate state s_c for each $i \in I^e$. Here, when estimating the previous states of s_c by a scan-input $i_s \in I_s$, *min_ess* defines only the state transition (or state encoding) to s_c by i_s partially according to some strategies (mentioned below). The *min_ess* algorithm selects the input $i_c (\in I^e)$ such that the number of states from which input i_c takes M^e to the candidate reset state s_c (i.e., $|\delta_e^{-1}(s_c, i_c)|$) becomes maximum, and changes the target state set from $\{s_c\}$ to the previous state set $S_p = \delta_e^{-1}(s_c, i_c)$. This computation of the previous state set is executed concurrently for every state $s \in S$, and repeated until the previous state set becomes the set of all states S^e. The minimum-length ESS is obtained by arranging the inputs that are selected during the computation from the single reset state to all the states according to the reverse order of the selection.

Let $s_r \in S$ be a candidate reset state, and let $S(1) = \{s_r\}$. Let $S(n) (\subseteq S^e)$ be the n-th current state set, and let $i(n) (\in I^e)$ be the n-th selected input, i.e., $S(n) = \delta_e^{-1}(S(n-1), i(n-1))$. Suppose $S(l)$ is the current state. The previous state set $S(l+1)$ from which a scan-input $i_s \in I_s (= B^k)$ takes M^e to the current state $S(l)$ is estimated, and the state transitions with respect to $S(l)$ and $i(l)$ are defined if i_s is selected as the l-th input $i(l)$ as follows.

1. State transitions from invalid states: If $\delta_e(S_{inv}, i(l-1)) = \perp$ and $i(l-1) \in I$ $(l \geq 2)$, then define $\delta_e(S_{inv}, i(l-1)) = S(l-1)$ and $S(l)$ is updated to $S(l) \cup S_{inv}$.
2. State assignments: Let $S_u (\subseteq S(l))$ be the set of states of which state assignments are not defined, i.e.,

$$S_u = \{s \in S(l) | f_s(s) = f_n(s) = \perp\}. \qquad (20)$$

In order to maximize the cardinality of the previous state set (i.e., $|S(l+1)|$), for all states in S_u, $f_s(s)$ and $f_n(s)$ are defined so that they maximize

$$\left|\{a_n \in B^{m-k} | f_n(s) = a_n, s \in S(l)\}\right|, \qquad (21)$$

and minimize

$$\left|\{i_s \in I_s | f_s(s) = a_s, s \in S(l)\}\right|. \qquad (22)$$

Then, let $S_p (\subseteq S^e)$ be the set of states whose $f_n(s)$ are the same as some state in $S(l)$, i.e.,

$$S_p = \{s \in S^e | \exists s' \in S(l), f_n(s) = f_n(s')\}. \qquad (23)$$

Let $A_s = \{i_s | f_s(s) = i_s, s \in S_p\}$, and let $A_n = \{a_n | f_n(s) = a_n, s \in S_p\}$. The cardinality of the previous state set $S(l+1)$ will become $|A_s| \times |A_n|$. Let $S'_u = \{s \in S^e | f_s(s) = f_n(s) = \perp\}$. Let S_d be a set of states obtained by extracting $|A_s| \times |A_n| - |S_p|$ states from S'_u. In order to maximize the previous set of $S(l+1)$, S_d is determined so that it maximizes

$$\forall i \in I, |\delta_e^{-1}(S_p \cup S_d, i)|, \qquad (24)$$

and for $\forall s \in S_d$, the state encoding functions $f_s(s)$ and $f_n(s)$ which satisfy

$$f_s(s) \in A_s,\ f_n(s) \in A_n, \qquad (25)$$

$$\forall q \neq s \in S_d,\ f_s(s) \neq f_s(q),\ f_n(s) \neq f_n(q) \qquad (26)$$

are defined.

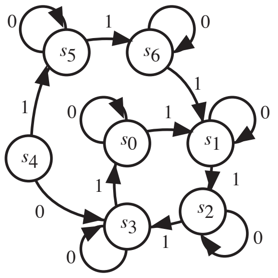
Example 4: Consider FSM M_3 illustrated in Fig. 5(a). Let $M_3^e = (S^e, I^e, \delta_e)$ be an partially-defined encoded FSM of FSM M_3. The number of state variables is three and one of the three state variables is scannable. Hence, $I_s = \{\underline{0}, \underline{1}\}$. Let s_{inv} be an invalid state, i.e., $S_{inv} = \{s_{inv}\}$. Here, we focus only on the search for a candidate reset state s_1.

```
encoded_FSM $min\_ess$(FSM $M$, int $m$, int $k$)
/* $m$: # of state variables, $k$: # of scannable variables */
{
    $S_{inv} = invalid(2^m - |M.S|)$;
    $S_{goal} = M.S \cup S_{inv}$;
    $I_s = scan\_input(k)$;  $I^e = M.I \cup I_s$;
    $S_r = M.S$;
    for($\forall s \in S_r$) {
        $M^e[s].S = \{s\}$;   $M^e[s].f_s = M^e[s].f_n = \phi$;
        $M^e[s].l = 0$;   $M^e[s].ess[0] = \epsilon$;
        $M^e[s].\delta_s = M.\delta$;
    }
    $M^e_{min} = \phi$;
    while($M^e_{min}.S != S_{goal}$ && $S_r != \phi$) {
        for($\forall s \in S_r$) {
            $M_c = M^e[s]$;   $i_p = Mc.ess[Mc.l]$;
            if($i_p \in I_s$) { $I_c = M.I$; } else { $I_c = I^e$; }
            $a_m = 1$;
            for($\forall i_c \in I_c$) {
                if($i_c \in I_s$) {
                    if($M_c.\delta_s(S_{inv}, i_p) == \perp$ && $M_c.l \geq 1$) {
                        $def\_trans(M_c.\delta_s(S_{inv}, i_p) = M_c.\delta_s(Mc.S, i_p))$;
                    }
                    $part\_encode(M_c.f_s, M_c.f_n, M_c.S)$;
                }
                $S_c = \delta_s^{-1}(M_c.S, i_c)$;
                if($|S_c| > a_m$ && $untried(S_c)$) {
                    $i_m = i_c$;   $a_m = |S_c|$;
                }
                if($a_m > 1$) {
                    $M^e[s] = M_c$;   $++(M^e[s].l)$;
                    $M^e[s].ess[M^e[s].l] = i_m$;
                } else { $M^e[s] = backtrack(M^e[s])$; }
            }
            if($all\_tried(s)$) { $S_r = S_r - \{s\}$; }
            if($M^e[s].S == S_{goal}$) {
                $M^e_{min} = M^e[s]$;
                break;
            }
        }
    }
    return $M^e_{min}$;
}
```

Figure 4: Algorithm *min_ess*

1. $S_c = s_1$. $\forall s \in S^e$, $f_s(s) = f_n(s) = \perp$.
2. Select an input that maximizes the previous state set for S_c. $|\delta_e^{-1}(s_1, 0)| = 1$. $|\delta_e^{-1}(s_1, 1)| = 2$. $\forall a_s \in I_s$, $|\delta_e^{-1}(s_1, a_s)| = 2$. Here, select normal-input 1. Update S_c. $S_c = \delta_e^{-1}(s_1, 1) = \{s_0, s_6\}$.
3. Estimate $|\delta_e^{-1}(S_c, i)|$ for all $i \in I^e$. $|\delta_e^{-1}(S_c, \underline{0})| = |\delta_e^{-1}(S_c, \underline{1})| = 2$. Since $\forall i \in I$, $\delta_e(s_{inv}, i) = \perp$, change S_c to $S_c = \{s_0, s_6, s_{inv}\}$, and define $\delta_e(s_{inv}, 1) = s_1$. Select scan-input $\underline{0}$, i.e., $f_s(s_0) = f_s(s_6) = f_s(s_{inv}) = 0$. Define $f_n(s_0) = 00$, $f_n(s_6) = 01$, and $f_n(s_{inv}) = 10$.
   Further, define $f_s(s_1) = f_s(s_2) = f_s(s_3) = 1$. $f_n(s_1) = 00$, $f_n(s_2) = 01$, and $f_n(s_3) = 10$. Thus, $S_c = \{s_0, s_1, s_2, s_3, s_6, s_{inv}\}$.
4. Select normal-input 0. S_c becomes $S_c = \{s_0, s_1, s_2, s_3, s_4, s_5\}$.
5. By defining $f_s(s_4) = 1$, $f_s(s_5) = 0$ and $f_n(s_4) = f_n(s_5) = 11$, S_c becomes $S_c = S^e$ with scan-input $\underline{1}$.

Thus, the state encoding which minimizes the ESS length is obtained as shown in Fig. 5(b), and the minimum-length ESS is $(\underline{1}, 0, \underline{0}, 1)$. □



(a) State diagram

s	$f_s(s)$	$f_n(s)$
s_0	0	00
s_1	1	00
s_2	1	01
s_3	1	10
s_4	1	11
s_5	0	11
s_6	0	01
s_{inv}	0	10

(b) State encoding

Figure 5: FSM M_3

4 Experimental results

The *min_ess* algorithm was implemented in C language and experiments were made using MCNC'91 benchmark FSMs [12] to confirm the effectiveness of the algorithm. We have shown that all the MCNC'91 benchmark FSMs can be extended-synchronized with one scannable variable in [13]. Some FSMs in the benchmark set have reset inputs explicitly, i.e., they are synchronizable by a single normal-input. Hence, we made experiments on the FSMs that are not resettable with a single normal-input, provided that the number of state variables is the minimum, i.e., $\lceil \log_2 n \rceil$, where n is the number of states, and one of the state variables is scannable.

Table 1 shows the experimental results. The first three columns denoted by FSM, ST and PI in this table mean the FSM name, the number of states and the bit size of normal-inputs (i.e., the number of normal-inputs is 2^{PI}), respectively. The fifth column, min_ess, denotes the length of the ESS obtained by the *min_ess* algorithm. The fourth column, ROS, denotes the length of the ESS obtained by the ROS algorithm presented by Jiang *et al.* [10].

From these results, we can see that the ESS obtained by the *min_ess* algorithm is shorter than or as short as that obtained by ROS for all FSMs. Moreover, we can see that the FSMs that cannot be synchronized with the ESSs obtained by ROS, *modulo12* and *tav*, can be extended-synchronizable.

The last column in Table 1, MIN, denotes the minimum length of the ESS obtained by computing all ESSs exhaustively without heuristics. In this column, ‡ denotes that the minimum-length ESS cannot be calculated due to the time/memory limitation. From these results, we can see that the length of the ESS obtained by *min_ess* is minimum for most of the FSMs.

In a master-slave configuration, when a slave is normally allowed to change, a scan-input can be applied to override the value coming from the master, setting the slave to the corresponding value. Hence, if FSMs are implemented on master-slave configurations, a scan-input in an ESS can be applied simultaneously with the previous normal-input followed by the scan-input. The bracketed number of the fifth column, min_ess, in Table 1 represents the length of the ESS provided that a normal-input and the subsequent scan-input are applied

Table 1: Experimental results: ESS length

FSM	ST	PI	ESS Length		
			ROS	min_ess	MIN
bbara	10	4	2	2	2
bbsse	16	7	2	2	2
bbtas	6	2	2	2	2
beecount	7	3	2	2	2
dk14	7	3	2	2	2
dk16	27	2	3	3 (2)	‡
dk17	8	2	2	2	2
dk27	7	1	2	2	2
dk512	15	1	3	3	‡
donfile	24	2	3	3 (2)	‡
ex1	20	9	2	2	2
ex2	19	2	2	2	2
ex3	10	2	2	2	2
ex4	14	6	5	4 (3)	‡
ex5	9	2	2	2	2
ex7	10	2	2	2	2
keyb	19	7	2	2	2
lion	4	2	2	2	2
lion9	9	2	2	2	2
mc	4	3	2	2	2
modulo12	12	1	†	5 (3)	‡
planet	48	7	8	4 (3)	‡
planet1	48	7	8	4 (3)	‡
s1	20	8	2	2	2
s1a	20	8	2	2	2
sand	32	11	5	4 (3)	‡
shiftreg	8	1	3	3	3
sse	16	7	2	2	2
styr	30	9	2	2	2
tav	4	4	†	3 (2)	3
train11	11	2	2	2	2
train4	4	2	2	2	2

†: unsynchronizable ‡: aborted calcuration

simultaneously. From these results we can see that the ESS length is three or less for all the benchmark FSMs, and hence the number of clock cycles required to apply ESSs is short. In particular, for the case where the ESS includes several scan-inputs, the number of clock cycles to extended-synchronize is considerably reduced *modulo12*.

5 Conclusions

In this paper, we presented an approach to the synthesis of synchronizable finite state machines (FSMs) with partial scan. We proposed an *extended synchronizing sequence* which consists of *normal-inputs* and *scan-inputs*, and which takes a circuit to a certain final state, regardless of the initial state of the circuit. We formulated the *minimum-length extended synchronizing sequence problem* and proposed a heuristic algorithm for the problem. We also presented experimental results of the minimization of extended synchronizing sequence length on MCNC'91 benchmark FSMs. The experimental results show that the proposed algorithm can find a minimum-length extended synchronizing sequence for most of benchmark circuits, and the length of the extended synchronizing sequence is three or less for all the benchmark FSMs.

Acknowledgment

It is a pleasure to thank Prof. Kewal K. Saluja (University of Wisconsin-Madison) and Dr. Michiko Inoue (Nara Institute of Science and Technology) for their helpful comments and discussions on this work.

References

[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," in *Proc. 14th Design Automation Conf.*, pp462-468, Jun. 1977.

[3] V.D. Agrawal and K.-T. Cheng, "A complete solution to the partial scan problem," in *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," in *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. Computer-Aided Design*, vol.8, no.10, pp.1100-1107, Oct. 1989.

[6] 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.

[7] B. Eschermann and H.-J. Wunderlich, "Optimized synthesis techniques for testable sequential circuits," *IEEE Trans. Computer-Aided Design,* vol.11, no.3, pp.301-312, Mar. 1992.

[8] F.C. Hennie, *Finite State Models for Logical Machines.* John Wiley, New York, 1968.

[9] K.-T. Cheng and V.D. Agrawal, "Initializability consideration in sequential machine synthesis," *IEEE Trans. Comput.*, vol.41, no.3, pp.374-379, Mar. 1992.

[10] N. Jiang, R.M. Chou, and K.K. Saluja, "Synthesizing finite state machines for minimum length synchronizing sequence using partial scan," in *Dig. 25th Int. Symp. Fault-Tolerant Comput.*, pp.41-49, Jun. 1995.

[11] I. Pomeranz and S.M. Reddy, "On the role of hardware reset in synchronous sequential circuit test generation," *IEEE Trans. Comput.*, vol.43, no.3, pp.1100-1105, Sep. 1994.

[12] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," 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 variable is almost sufficient for synthesizing synchronizable FSMs," Graduate School of Information Science, Nara Institute of Science and Technology, Tech. Rep., NAIST-IS-TR96014, 1996 (available from http://isw3.aist-nara.ac.jp/IS/TechReport2/report/96014.ps).