Practical Synthesis of Speed-Independent Circuits Using Unfoldings

Uisok Kim  Dong-Ik Lee

Dept. of Information and Communication  K-JIST
572 Kwangsan-Ku Kwangju 306-712 KOREA
Tel: +82-62-970-2267
Fax: +82-62-970-2204
e-mail: uskim@geguri.kjist.ac.kr dilee@kjist.ac.kr

Abstract In this paper, we present a practical synthesis method using unfoldings which are based on partial order semantics and hence free of state space explosion inherently. In addition, we suggest several conditions for basic gate implementation in order to enhance practicality of the suggested method.

I. INTRODUCTION

Signal transition graphs[1] in short STGs which are interpreted Petri Nets have been widely accepted as a specification formalism for SI circuits. Further several computer aided logic synthesis tools for SI circuits based on STGs have been developed for research purposes recently. However unfortunately those are not practically acceptable for the following defects: The first defect is expensive time and space complexity. Most proposed design methodologies for SI circuit synthesis are based on enumerating all possible states of given STGs that is State Graphs (SGs) of given STGs have to be generated. This approach may suffer from state space explosion i.e. the number of states of the SG may grow exponentially with the number of signals due to inherent concurrency of the STG. The second problem is related to hazard free implementation. Most of existing methods assume that circuits are implemented in a complex gate due to hazards. However this approach cannot take an advantage of using a standard library and is very disadvantageous in the point of time-to-market.

There are several well-known approaches to solve above two problems. Pastor et al.[2] have developed a methodology for the direct synthesis of SI circuits from STGs without generating SGs. Their approach succeeded in reducing time complexity and memory requirements. However the method has a structural restriction for an STG specification hence not applicable to general STGs. Semenov et al. [3] have suggested an approach using unfoldings [4][5] which is based on partial order semantics and is free of state space explosion inherently instead of SGs. However they only take into account complex gate implementations consequently the method is not practical enough. Kondratyev et al.[6] have suggested Monotonous Cover requirement which is a condition for synthesis of SI circuits using only basic gates such as AND/OR gates with unlimited fanins and a C-element. Although their approach is very systematic practically the method is severely restricted by the following two facts; suggested conditions are characterized in SGs and SGs are analyzed based on BDDs requiring a double exponential time complexity.

As above several existing approaches show state space explosion problem and basic gate implementation are intertwined with each other and require a unified approach to efficiently synthesize SI circuits. In this paper we suggest several sufficient conditions for efficient SI circuit derivation consequently we solve the previously pointed two serious problems based on the unfolding.

II. BASIC DEFINITIONS

A. Petri Net

DEFINITION II.1 (Petri Net) A Petri Net (PN) is a 4-tuple \( \Sigma = (P,T,F,M_0) \) where \( P \) is the set of places, \( T \) is the set of transitions, \( F \subseteq (P \times T) \cup (T \times P) \), such that \( \text{dom}(F) \cup \text{range}(F) = P \cup T \). \( M_0 \) is the initial marking represented by a \( |P| \times 1 \) column vector. By firing of transition \( t \) a marking \( M \) changes into \( M' \) denoted by \( M[t > M'] \). The set of reachable markings from \( M \) is denoted by \( |M > \).

We use dot notations to represent the preset and post-set of a place or a transition as follows:

(i) \( \bullet(t) \bullet \) is the set of input(output) places of \( t \).
(ii) \( \bullet(p) \bullet \) is the set of input(output) transitions of \( p \).

STGs are PNs whose transitions are interpreted as evolution of signals in corresponding circuits and SGs which are used to analyse STGs are finite automata having markings of corresponding STGs as its states. STGs and the corresponding SGs are formally defined as follows:

DEFINITION II.2 (Signal Transition Graph) A Signal Transition Graph (STG) is a 4-tuple \( G = \)
(Σ, X, Z, L) where Σ is a PN or X and Z, X ∩ Z = ∅, are sets of input and non-input signals respectively and L : T → {+, −} × (X ∪ Z) labels each transition of Σ with a signal transition.

**DEFINITION II.3 (State Graph) A State Graph (SG) is a 4-tuple G = (X, Z, E) where X and Z \( \cap Z = \emptyset \), are sets of input and non-input signals respectively.**

\( S \subseteq \{0, 1\}^n \) is the set of binary states and \( E \subseteq S \times S \) is the set of edges labeled by transitions. Each binary state represents binary values of signals in a corresponding marking \( M \) and is expressed as \( S(M) \). \( S(M)_{s_i} \) denotes the entry corresponding to \( s_i \) in \( S(M) \).

Fig. 1 (a) and (b) show an STG and its SG respectively.

**B. Unfolding**

In prior to defining unfoldings\( \Gamma \) we need to introduce OCNs on which unfoldings are based. OCNs which are kind of acyclic PN\( \Gamma \) can be derived from PNs by Algo. II.1.

**ALGORITHM II.1 (Generation of an OCN)\( [4] [5] \)**

(Step1) Copy every place \( p_i \) such that \( M_0(p_i) \geq 1 \) into the OCN.

(Step2) Choose a transition \( t_i \) from the PN.

(Step3) For each place in \( \cdot t_i \) find a copy in the OCN. If a copy cannot be found \( t_i \) go back to Step2. Do not choose the same subset more than once for a given \( t_i \).

(Step4) If any pair of chosen places is not in concurrent relation \( t_i \) go back to Step2.

(Step5) Make a copy of \( t_i \) in the OCN. Call it \( t_i' \). Draw an arc from each place which was found in Step3 to \( t_i' \).

(Step6) For each place in \( t_i \) make a copy in the OCN and draw an arc from \( t_i' \) to it.

(Step7) Repeat Step2-Step6 as many as possible.

In an OCN\( P \Gamma T \) and \( M' \) denote sets of places transitions and markings corresponding to \( P \Gamma T \) and \( M \) in a PN. In an OCN a node in a PN can occur several times and \( j \)-th appearances of node \( p_i \) and \( t_i \) are denoted as \( p_i^j \) and \( t_i^j \) \( (j = 1, 2, 3 \cdots) \) respectively. OCNs\( \Gamma \) which are based on partial order semantics and acyclic\( \Gamma \) are greatly helpful to investigate behavior of concurrent systems. An OCN represents acyclically all possible processes occurring in a given PN. Thus an OCN may expand infinitely\( \Gamma \) even though a given PN is bounded. This means that the state space of an OCN. It was proved that global states also can be defined in an OCN and a complete prefix\( \Gamma \) called unfolding of an OCN preserves the state space of an PN. Hence an unfolding is very useful to analyze a PN without experiencing state space explosion. For the lack of space\( \Gamma \) a formal definition of an unfolding is omitted here. Please refer to \( [4] \) and \( [5] \) for the definition.

**DEFINITION II.4 (Ordering Relations) Let Σ = (P, T, F, M) be an acyclic PN or an OCN and \( x_1, x_2 \in P \cup T \).**

(i) \( x_1 \) precedes \( x_2 \) if denoted by \( x_1 \Rightarrow x_2 \) if \( (x_1, x_2) \) belongs to the reflexive transitive closure of \( FT \), i.e., there is a directed path in Σ from \( x_1 \) to \( x_2 \).

(ii) \( x_1 \) and \( x_2 \) are in conflict\( \Gamma \) if denoted by \( x_1 \neq x_2 \) if there exist distinct transitions \( t_1, t_2 \) such that \( t_1 \in E \) and \( t_1 \Rightarrow x_1 \) and \( t_2 \Rightarrow x_2 \). If \( x \neq x' \) where \( x \in P \cup T \) then \( x \) is in self-conflict.

(iii) \( x_1 \) and \( x_2 \) are concurrent\( \Gamma \) if denoted by \( x_1 || x_2 \) if they are neither in precedence nor in conflict.

**III. CUBE APPROXIMATION**

In this section\( \Gamma \) cube approximation method is introduced. It enables us to extract binary state information.
of unfoldings efficiently.

**Definition III.1** Given an unfolding UT we define two following marking sets
\[ M_{p}' = \{ M \in M_0 \mid |M(p_i')| = 1 \} \]
\[ M_{i}' = \{ M \in M_0 \mid |M(s_i^1)| > 1 \} \]
\( M_{p}' \) and \( M_{i}' \) are the set of binary states corresponding to markings in \( M_{p}' \) and \( M_{i}' \) respectively.

For example \( \Gamma \) for a place \( p_i' \) and a transition \( s^1 \) in Fig. 1 (c) \( \Gamma M_{s_i} = \{ \{ p_3', p_1', p_1' \} \Gamma \{ p_1', p_3', p_5' \} \Gamma \{ p_3', p_1, p_5' \} \Gamma \{ p_1', p_3', p_5' \} \} \Gamma M_{s_i} = \{ \{ p_2', p_4', p_5' \} \Gamma \{ p_3', p_1, p_5' \} \} \Gamma M_{s_i} = \{ \{ p_2', p_4', p_5' \} \} \Gamma \).

In order to obtain marking sets and binary state sets efficiently we define a cube in terms of ordering relations between nodes.

**Definition III.2** For any place and signal \( p_i' \) and \( s_i^j \), cubes \( C_{p_i'} \) and \( C_{s_i^j} \) are defined as follows: For a signal \( s_k \Gamma \)

\[
C_{p_i'}(s_k) = \begin{cases} 
* & \text{if } (s_k^i/m) \text{ is concurrent with } p_i' \\
0 & \text{elseif } (s_k^i/m) \Rightarrow p_i' \text{ and there does not exist any instance of } s_k \text{ between } s_k^i/m \text{ and } p_i' \text{ or } (p_i' \Rightarrow +s_k^i/m) \text{ and there does not exist any instance of } s_k \text{ between } p_i' \text{ and } +s_k^i/m), \\
1 & \text{else.}
\end{cases}
\]

Note that a signal \( s_k \Gamma * \) takes + or − for rising and falling transition of a signal \( \Gamma \) can appear more than once in an STG and \( s_k/m \) denotes \( m \)-th appearance of \( s_k \). Further \( s_k/m \) in an STG can also appear more than once in an unfolding \( \Gamma s_k^i/m \) denotes \( i \)-th appearance of \( s_k^i/m \) in an unfolding. \( C_{s_i^j} \) is defined as an intersection among cubes for places belonging to the preset of \( s_i^j \). That is

\[ C_{s_i^j} = \bigcap_{p_i' \in \Gamma} C_{p_i'} \]

An ordering relation between any pair of nodes in an unfolding can be decided in polynomial time with respect to size of an unfolding. Therefore a cube defined in Def. III.2 can be obtained within a polynomial time. The following lemma shows that a cube defined in Def. III.2 is sufficient for obtaining whole binary states.

**Lemma III.1** For any place \( p_i' \) and signal \( s_i^j \) \( C_{p_i'} \) and \( C_{s_i^j} \) always include \( M_{p_i'} \) and \( M_{s_i} \) respectively.

Table 1 shows the values of cubes of nodes belonging to an unfolding in Fig. 1 (c).

<table>
<thead>
<tr>
<th>CUBE</th>
<th>VALUE</th>
<th>CUBE</th>
<th>VALUE</th>
<th>CUBE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{p_1} )</td>
<td>yes</td>
<td>( C_{p_2} )</td>
<td>yes</td>
<td>( C_{p_3} )</td>
<td>yes</td>
</tr>
<tr>
<td>( C_{p_5} )</td>
<td>yes</td>
<td>( C_{p_6} )</td>
<td>yes</td>
<td>( C_{p_7} )</td>
<td>yes</td>
</tr>
<tr>
<td>( C_{p_9} )</td>
<td>yes</td>
<td>( C_{p_{10}} )</td>
<td>yes</td>
<td>( C_{p_{11}} )</td>
<td>yes</td>
</tr>
</tbody>
</table>

**IV. IMPLEMENTATION OF SI CIRCUITS**

In this section at first several conditions to obtain SI circuit implementation are characterized in terms of both an STG specification and a circuit implementation. After that we explain how to check those conditions on unfoldings.

**A. SI Implementability of STGs**

The following proposition describes sufficient conditions for a given STG to be SI implementable [7].

**Proposition IV.1** [7] An STG is SI implementable if it is bounded consistent and persistent.

Properties in Prop. IV.1 are the key properties for an SI circuit implementation of a given STG. Boundedness guarantees finiteness of the specification and therefore confirms that the specification can be implemented with a finite circuit. Consistency is a property directly related to a state coding and persistency is related to speed-independence of a circuit. These properties can be easily checked in unfoldings using the method in [7]. Hereafter we assume that a given STG is SI implementable.

**B. Implementation of SI circuits**

After checking SI implementability of a given STG a real circuit is derived from the STG. There are several well known SI circuit structures such as an ACgsP (atomic complex gate per signal structure) and an ACgsPER (atomic complex gate per excitation function structure) and an ACgsPER (atomic complex gate per excitation region structure) and we select an ACgsPER structure as shown in Fig. 2. In Fig. 2 SI and RI denote set and reset cover cubes respectively. The set of set(reset) cover cubes and an OR gate is called a set(reset) function S(R). In this paper two different notations a binary notation and a character notation are used to describe a product term. For example 1010 has the same meaning as an \( s_1 s_2 s_3 s_4 \).
DEFINITION IV.1 (Cover Cube) For an instance $s_i^j/l$ of an non-input signal $s_i$ a cover cube $CC_{s_i^j/l}$ is

$$CC_{s_i^j/l}(s_k) = \begin{cases} C_{s_i^j/l}(s_k), & \text{if } s_i \neq s_k, \\ \ast, & \text{if } s_i = s_k. \end{cases}$$

DEFINITION IV.2 (SCC and RCC) For any signal instance $s_i^j/l$ of a non-input signal $s_i$ in an STG a Set Cover Cube $SCC_{s_i^j/l}$ is defined as

$$SCC_{s_i^j/l} = \bigcup_{j=1}^{n} CC_{s_i^j/l}.$$ A Reset Cover Cube $RCC_{s_i^j/l}$ is defined as

$$RCC_{s_i^j/l} = \bigcup_{j=1}^{n} CC_{-s_i^j/l}$$

in the same manner.

For example in Fig. 1(c) for an output signal $R_o$ cover cubes $CC_{R_o}$ and $CC_{-R_o}$ are 1*1*1 and 1*1*0 respectively. A Set Cover Cube $SCC_{R_o}$ and a Reset Cover Cube $RCC_{-R_o}$ are 1*1*1 and 1*1*0 respectively.

An SCC and an RCC defined in Definition IV.2 are directly mapped to AND gates in a signal network. For example if set cover cube $SCC_{s_i^j/l}(s_k)$ is equal to 1 then the AND gate corresponding to $SCC_{s_i^j/l}$ has an input $s_k$ and if equal to 0 it has an inverted $s_k$ and otherwise it does not have $s_k$ as an input.

**LEMMA IV.2** An SCC and an RCC satisfy the following two facts;

1. An $SCC_{s_i^j/l}$ and an $RCC_{s_i^j/l}$ covers all the binary states in $\bigcup_{j=1}^{n} CC_{s_i^j/l}$ and $\bigcup_{j=1}^{n} CC_{-s_i^j/l}$ respectively.

2. The sets of SCs and RCCs cover all the binary states under which $+s_i$ and $-s_i$ are enabled respectively.

DEFINITION IV.3 (SI implementation) A circuit is an SI implementation if and only if

1. Output equivalence: The Boolean expression $f$ of each output signal $s_i$ satisfies the specification. That is, for any marking $M$ in an STG

$$f(S(M)) = \begin{cases} S(M)_i & \text{if } s_i \text{ is not enabled at a mark} \\ \sim M_i & \text{otherwise.} \end{cases}$$

2. Speed-independence: The actual behavior of the circuit when the inputs behave as specified by the specification is independent of gate delays.

Lemma IV.3 presents a sufficient condition for an SI circuit implementation in an ACGpER structure.

**LEMMA IV.3** For a non-input signal $s_i$ if following three conditions are satisfied the signal network implemented according to Def. IV.1 and IV.2 in an implementation framework of Fig. 2 is an SI implementation at a gate level.

(Cond.1) At most one set/reset cover cube can be set at once.

(Cond.2) For any marking $M_l$ and $M_m$ such that $f(S(M_l))=0$ and $f(S(M_m))=1$ $SCC_{s_i} \cap M_l = \emptyset$ and $RCC_{s_i} \cap M_m = \emptyset$ must be satisfied.

(Cond.3) Outputs of set/reset function $S(R)$ always change monotonously.

C. Cover Cube Conditions for SI Circuit Implementation on Unfoldings

In this subsection we characterize cover cube conditions for an SI implementation and explain how to check those conditions with cubes for places and transitions on unfoldings.

**DEFINITION IV.4 (Tree)** The set of nodes in an unfolding satisfying the following three conditions is called a tree $\Gamma$.

(Cond.1) Each tree $\Gamma$ contains exactly one source node $x\Gamma\in\{0\}$ and at least one sink node $y\Gamma\in\{0\}$ of an unfolding.

(Cond.2) A node in a tree $\Gamma$ is not a place $p$ such that $|p|>1$ has exactly one prenode and one postnode if exists in common with $\Gamma$.

(Cond.3) All the postnodes of a place $p$ such that $|p|>1$ are included in a tree $\Gamma$.

$\Gamma_{s_i}$ denotes a tree containing all instances of a specific signal $s_i$. If there exist more than one tree $\Gamma_{s_i}$ for a signal $s_i$ then each tree is represented as $\Gamma_{s_i}(k)\in\{1,2,3\}$. . .

**LEMMA IV.4** All the binary states of an STG can be obtained by cubes of nodes in a tree $\Gamma$.

Lemma IV.4 implies that the concept of tree presents a way to access all the states of an STG efficiently. That is, we only need to consider cubes for nodes in a tree instead of all of the nodes in an unfolding. For example consider the tree $\Gamma_{R_o} = \{p_{01}^{01}\Gamma + R_o^{p_{01}}\Gamma p_{01}^{p_{01}}\Gamma + R_o^{p_{01}}\Gamma p_{01}^{p_{01}}\Gamma - S^{p_{01}}\Gamma p_{01}^{p_{01}}\Gamma + \Sigma^{p_{01}}\Gamma p_{01}^{p_{01}}\Gamma - R_o^{p_{01}}\Gamma p_{01}^{p_{01}}\Gamma + S^{p_{01}}\Gamma p_{01}^{p_{01}}\Gamma\}$ in Fig. 1(c). Cubes in $\Gamma_{R_o}$ are $C_{p_{01}}$, $C_{p_{01}}^{p_{01}}$, $C_{p_{01}}^{p_{01}}$, $C_{p_{01}}^{p_{01}}$, and $C_{p_{01}}^{p_{01}}$ are 0*11, 0*11, 1*0*11, 1*0*11 and *0*0 respectively and they cover whole binary states of the STG in Fig. 1(a).
DEFINITION IV.5 (PPS and NPS) For any non-input signal \( s_\Gamma \) if \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \neq \emptyset \) then the set \( \{ s_\Gamma \} \) is not empty and the condition of Lemma IV.5 is violated. Therefore two AND gates \( R \) and \( B \) corresponding to set cover cubes \( SCC_{+s_\Gamma} \) and \( SCC_{-s_\Gamma} \) satisfy Cond.1 and Cond.2 in Lemma IV.3.

As Fig. 3(a) shows if \( SCC_{+a_1} \equiv SCC_{+a_2} \equiv \{ \{ a_1 \} \} \) then \( SCC_{+a_1} \equiv \{ \{ a_1 \} \} \) which is not empty and the condition of Lemma IV.5 is violated. Therefore two AND gates \( R \) and \( B \) corresponding to set cover cubes \( SCC_{+a_1} \) and \( SCC_{+a_2} \) satisfy Cond.1 and Cond.2 in Lemma IV.3.

LEMMA IV.5 For a non-input signal \( s_\Gamma \) if \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \neq \emptyset \) then the set cover cubes \( SCC_{+s_\Gamma} \) and \( SCC_{-s_\Gamma} \) satisfies Cond.1 and Cond.2 in Lemma IV.3.

LEMMA IV.6 For a non-input signal \( s_\Gamma \) if three conditions are met \( \Gamma \) can be fused by Def. IV.1 and IV.2 satisfies Cond.1 and Cond.2 in Lemma IV.3.

(Cond.1) For any \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \neq \emptyset \) if \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \cap C_p = \emptyset \) then \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \cap C_p \neq \emptyset \) if p belongs to NPS(PPS) for a tree \( \Gamma \) and any instance of \( +s_\Gamma \) does not belong to \( \Gamma \) respectively.

(Cond.2) For any \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \neq \emptyset \) if \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \cap C_p = \emptyset \) then \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \cap C_p \neq \emptyset \) if p belongs to NPS(PPS) for a tree \( \Gamma \) and any instance of \( +s_\Gamma \) does not belong to \( \Gamma \) respectively.

(Cond.3) For any pair \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \neq \emptyset \) if \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \cap C_p = \emptyset \) then \( SCC_{+s_\Gamma} \cap SCC_{-s_\Gamma} \cap C_p \neq \emptyset \) if p belongs to NPS(PPS) for a tree \( \Gamma \) and any instance of \( +s_\Gamma \) does not belong to \( \Gamma \) respectively.

For example Fig. 3(b) shows Lemma IV.6 is violated between a place \( p_1 \) and \( +b \). That is \( SCC_{+b} \cap C_{p_1} = \emptyset \) which is not empty.

DEFINITION IV.6 (\( ; \) operator) A binary operator \( ; \) for cover cubes is defined as follows:

\[
a_1 a_2 \cdots a_n \cap b_1 b_2 \cdots b_n =
\begin{cases}
0 & \text{if } \exists i \ (1 \leq i \leq n), a_i \neq b_i \land (\text{neither } a_i \text{ nor } b_i \text{ is } *), \\
1 & \text{else if } b_1 b_2 \cdots b_n \subseteq a_1 a_2 \cdots a_n, \\
\text{u} & \text{otherwise},
\end{cases}
\]

where \( a_1 a_2 \cdots a_n \) and \( b_1 b_2 \cdots b_n \) are n-dimensional vectors.

LEMMA IV.7 For a non-input signal \( s_\Gamma \) if \( SCC_{+s_\Gamma} \cap C_p \neq \emptyset \) then \( SCC_{-s_\Gamma} \cap C_p \neq \emptyset \) for each place \( p \) on one directed path from \( +s_\Gamma \) to \( -s_\Gamma \) the next instance of \( s_\Gamma \) to \( +s_\Gamma \) to \( -s_\Gamma \) the next instance of \( s_\Gamma \) to \( +s_\Gamma \) to \( -s_\Gamma \) respectively.

If a signal network consists of cover cubes satisfying Lemma IV.5, IV.6, and IV.7, it works speed-independently at a gate level for any input sequence from an environment.

As explained above cover cube conditions to construct SI circuits can be checked in unfoldings efficiently. If Theorem IV.8 is satisfied we can implement SI circuits according to Definition IV.1 and IV.2. However if not satisfied what should we do? Two solutions can be considered to solve it. One is called refinement and signal insertion. In general signal insertion is easier than refinement while refinement cannot always solve problems and may introduce new problems. In this paper we do not present details of methods for refinement and signal insertion because they are beyond range of this paper and we assume that if there are violations of theorems IV.8-IV.1 we can solve them by refinement or signal insertion.

V. EXPERIMENTAL RESULTS

In this paper we suggest an SI circuit synthesis method to enhance practicality in the aspects of time complexity, memory requirements and circuit implementability. The proposed method is implemented as an STG synthesis tool "Unfolder". While synthesizing various STG benchmarks using Unfolder we check the following two facts:

- An SI circuit synthesis method based on unfoldings is superior to methods based on other models (SGs and BDDs) in the points of time and memory requirements.
• SI circuits consisting of just only basic gates can be obtained by the suggested method.

Experimental results in Table 2 are obtained by using SIS[8], Petrify[9] and Unfolder[10] which are based on SGs, BDDs and unfoldings respectively on the SUN sparc 20 and show the first advantage of a suggested method. Table 2 shows that Unfolder based on unfoldings is much more efficient in the points of time complexity and memory requirements. That is, as the number of states increases time and memory required by SIS and Petrify increase exponentially but in case of Unfolder they increase linearly. This means that for small STGs with small concurrency SIS based on SGs are suitable but for big STGs with big concurrency SIS is inefficient. Petrify which is based on BDDs in order to avoid state space explosion is much more efficient than SIS but still has a problem for an STG generating a big state space as shown in Table 2.

Moreover Unfolder succeed in implementing SI circuits from benchmarks using just only basic gates such as AND/OR gates and C-elements. Synthesized circuits are verified for STG specifications by VERSIFY[11] which is a verification tool. Therefore we can conclude that Unfolder based on unfoldings is more practical than any other tools based on SGs or BDDs.

VI. CONCLUSION AND FUTURE WORK

In this paper we suggest a more practical SI circuit synthesis method based on unfoldings in comparison with pre-existing methods. Our tool “Unfolder” succeeds in synthesizing SI circuits very rapidly using just only basic gates. Nowadays we concentrate our all efforts on enhancing the quality of synthesized circuits through additional optimization and a technology mapping.

<table>
<thead>
<tr>
<th>send-done</th>
<th>place in STG</th>
<th>trans. in STG</th>
<th>states</th>
<th>SIS (CPU time sec.)</th>
<th>Petrify (CPU time sec.)</th>
<th>place in unfolding</th>
<th>trans. in unfolding</th>
<th>Unfolder (CPU time sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>half</td>
<td>11</td>
<td>8</td>
<td>9</td>
<td>0.1</td>
<td>0.2</td>
<td>9</td>
<td>8</td>
<td>0.0</td>
</tr>
<tr>
<td>cpu2</td>
<td>16</td>
<td>14</td>
<td>18</td>
<td>0.1</td>
<td>0.4</td>
<td>16</td>
<td>11</td>
<td>0.1</td>
</tr>
<tr>
<td>sbuf-read-ctl</td>
<td>19</td>
<td>16</td>
<td>19</td>
<td>0.3</td>
<td>0.7</td>
<td>21</td>
<td>17</td>
<td>0.1</td>
</tr>
<tr>
<td>rpdlt</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>0.1</td>
<td>0.3</td>
<td>22</td>
<td>22</td>
<td>0.1</td>
</tr>
<tr>
<td>atod</td>
<td>17</td>
<td>14</td>
<td>24</td>
<td>0.2</td>
<td>0.6</td>
<td>14</td>
<td>14</td>
<td>0.1</td>
</tr>
<tr>
<td>vbe0b</td>
<td>26</td>
<td>12</td>
<td>24</td>
<td>0.2</td>
<td>0.6</td>
<td>16</td>
<td>12</td>
<td>0.1</td>
</tr>
<tr>
<td>ram-read-sbuf</td>
<td>28</td>
<td>22</td>
<td>39</td>
<td>0.6</td>
<td>1.8</td>
<td>30</td>
<td>23</td>
<td>0.2</td>
</tr>
<tr>
<td>nak-pa</td>
<td>24</td>
<td>20</td>
<td>58</td>
<td>0.7</td>
<td>1.2</td>
<td>24</td>
<td>20</td>
<td>0.0</td>
</tr>
<tr>
<td>vbe10b</td>
<td>32</td>
<td>22</td>
<td>296</td>
<td>4.7</td>
<td>2.7</td>
<td>62</td>
<td>40</td>
<td>0.4</td>
</tr>
<tr>
<td>master-read</td>
<td>40</td>
<td>28</td>
<td>2108</td>
<td>671.1</td>
<td>39.9</td>
<td>77</td>
<td>51</td>
<td>0.3</td>
</tr>
<tr>
<td>DMF10</td>
<td>41</td>
<td>40</td>
<td>11364</td>
<td>-</td>
<td>36.2</td>
<td>41</td>
<td>40</td>
<td>0.1</td>
</tr>
<tr>
<td>DMF20</td>
<td>81</td>
<td>80</td>
<td>22,020,006</td>
<td>-</td>
<td>665.4</td>
<td>81</td>
<td>80</td>
<td>0.4</td>
</tr>
<tr>
<td>DMF50</td>
<td>121</td>
<td>120</td>
<td>33,285,996,514</td>
<td>-</td>
<td>2861</td>
<td>121</td>
<td>120</td>
<td>1.1</td>
</tr>
<tr>
<td>phal0</td>
<td>71</td>
<td>51</td>
<td>4,684,382</td>
<td>-</td>
<td>690.98</td>
<td>71</td>
<td>51</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 2: Synthesis by SIS, Petrify and Unfolder

ACKNOWLEDGMENTS

This work has been supported in part by Samsung Electronics and by the Korea Science & Engineering Foundation under grant 966-0900-001-2.

References