Time-Frame Folding: Back to the Sequentiality

Po-Chun Chien[†] and Jie-Hong R. Jiang[†‡]
[†]*Graduate Institute of Electronics Engineering,* [‡]*Department of Electrical Engineering*
National Taiwan University
Taipei, Taiwan

***Abstract*—In this paper we formulate *time-frame folding* (TFF) as the reverse operation of time-frame unfolding (TFU), or commonly known as time-frame expansion in automatic test pattern generation (ATPG) and (un)bounded model checking. While the latter converts a sequential circuit into a combinational one with respect to some expansion bound of k time-frames, the former attempts the opposite. TFF arises naturally in the context of testbench generation and bounded strategy generalization, and yet remains unstudied. Unlike TFU, TFF can be highly non-trivial as the subcircuit of each time-frame can be distinct. We propose an algorithm that finds a minimum-state finite state machine consistent with the input-output behavior of the combinational circuit under folding. Empirical evaluation of our method demonstrates its ability in circuit size compaction and suggests potential use in different application domains.**

***Index Terms*—time-frame expansion, time-frame folding, functional decomposition, state minimization**

I. Introduction

Time-frame folding (TFF) is the reverse operation of time-frame unfolding (TFU), or time-frame expansion. While TFU is a well-known technique commonly used in, e.g., automatic test pattern generation (ATPG) [1] and (un)bounded model checking of sequential circuits [2], TFF remains largely unstudied. If fact, TFF finds its natural applications. For example, to test a sequential design, one may look for a testbench that produces some set of desired test patterns of length-bounded input-output sequences. The testbench can be represented directly by a large combinational circuit, corresponding to a time-frame expanded version of a sequential circuit, or represented more compactly by a sequential circuit. For another example, in model-based testing of software systems [3], [4], in state identification [5], and in system initialization [6], one may be asked to compute (non-adaptive or adaptive) homing, distinguishing, and/or synchronizing sequences. These problems can be formulated as quantified Boolean formula (QBF) [7] solving of strategy derivation, e.g., in [8], that computes the intended sequence. Again, the homing, distinguishing, or other strategy under synthesis can be represented directly by a large combinational circuit or more compactly by a sequential circuit.

However, unlike the straightforward derivation of TFU from a given sequential circuit, TFF can be highly non-trivial because the time-frame expanded combinational circuit may not exhibit a common circuit structure shared among different time-frames. Perhaps it is this difficulty that makes TFF largely unaddressed. In this work, we formulate the TFF problem and provides a general solution that makes no structure assumption on the combination circuit under time-frame folding.

To the best of our knowledge, this work is the first to address the time-frame folding issue. Most related prior work on time-frame issues centered around unfolding, e.g. in [9]. While the prior work converts a sequential circuit into a combinational one with respect to some expansion bound k time-frames, our attempt is the opposite. Regarding our method, we rely on multiple-output functional decomposition [10] to identify equivalent states as part of our computation flow. A similar technique has been applied in sequential equivalence checking [11].

The main results of this work include: 1) We motivate and formulate the problem of time-frame folding. 2) We propose an algorithm that finds a minimum-state finite state machine consistent with the input-output behavior of the combinational circuit under folding. 3) We evaluate the proposed algorithm and demonstrate the computational viability and its ability in circuit size compaction for potential use in different application domains.

The rest of this paper is organized as follows. After essential preliminaries are provided in Section II, the problem of time-frame folding is formulated in Section III. Our algorithmic solution is then presented in Section IV, and implementation improvement in Section V. Section VI shows the experiment results, and finally Section VII concludes this paper.

II. Preliminaries

In the sequel, sets are denoted by upper-case letters, e.g. S; the elements in a set are in lower-case letters, e.g. $a \in S$; the cardinality of a set S is denoted as $|S|$. A partition P of a set S into non-empty subsets $S_i \subseteq S$, for $i = 1, \ldots, k$, is denoted by $P = \{S_1|S_2|...|S_k\}$, where $S_i \cap S_j = \emptyset, \forall i \neq j$ and $\bigcup_i S_i = S$. Each S_i is a called a *cell* of P. Let P and P' be two partitions of a set S. Partition P is said to be a *refinement* of P', if $s_i, s_j \in S$ are in different cells of P', then $s_i, s_j \in S$ are in different cells of P. Note that the refinement relation is not symmetric, i.e., that P is a refinement of P' does not imply that P' is a refinement of P. For a set of Boolean variables X, its set of truth assignments is denoted by $[\![X]\!]$, e.g., $[\![X]\!] = \{(0,0),(0,1),(1,0),(1,1)\}$ for $X = \{x_1, x_2\}$. Boolean negation, conjuction, and disjunction are denoted by $\neg$ or overline, $\wedge$ or $\cdot$, and $\vee$ or $+$, respectively.

A. Functional Decomposition

Given a single-output Boolean function $f(X)$, the *functional decomposition* [12], [13] problem asks to re-express $f(X) = f_\mu(X_\mu, f_{\lambda_1}(X_\lambda), \ldots, f_{\lambda_k}(X_\lambda))$, where X_λ and X_μ are called the *bound set* and *free set* variables, respectively, which form a partition on $X = \{X_\lambda | X_\mu\}$.[1] Let $F_\lambda(X_\lambda) = \{f_{\lambda_1}(X_\lambda), \ldots, f_{\lambda_k}(X_\lambda)\}$. To avoid trivial decomposition, it is required that $|F_\lambda| < |X_\lambda|$. Figure 1 illustrates the structural effect of functional decomposition.

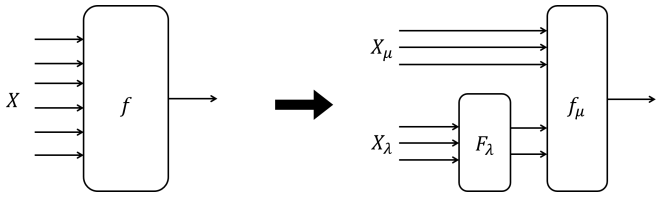


Fig. 1: Effect of functional decomposition.

Functional decomposition can be defined for multiple single-output functions $f_1(X), \ldots, f_m(X)$, and considered as decomposing a multiple-output function $F(X) = (f_1(X), \ldots, f_m(X))$. In [10], a technique called *hyperfunction* encoding is introduced to encode a multiple-output function into a single-output function with $\lceil \log_2 |F| \rceil$ auxiliary pseudo input variables. E.g., for $m = 4$, two auxiliary variables $A = \{\alpha_1, \alpha_2\}$ can be used to build the hyperfunction $h(X, A) = \neg\alpha_1\neg\alpha_2 f_1(X) + \neg\alpha_1\alpha_2 f_2(X) + \alpha_1\neg\alpha_2 f_3(X) + \alpha_1\alpha_2 f_4(X)$. Thereby, a single-output functional decomposition algorithm can be applied to decompose a multiple-output function.

Functional decomposition can be achieved based on the reduced ordered binary decision diagram (ROBDD) [14], [15]. In BDD-based decomposition, the ROBDD of the function $f(X)$ under decomposition is built with the variable ordering constraint that the bound set variables X_λ are ordered above the free set variables X_μ. The *cut set* of the ROBDD is the set of BDD nodes controlled by free set variables that are pointed to by some edge from a node controlled by a bound set variable. Essentially, for c being the cut set size, then $|F_\lambda| \geq \lceil \log_2 c \rceil$.

Example 1: Figure 2 shows the BDD-based decomposition for function y^2 of the time-frame expanded circuit `s27`. The cut set $\{n_1, n_2, n_3\}$ is induced by setting $X_\lambda = \{x_1^1, x_2^1, x_3^1, x_4^1\}$ and $X_\mu = \{x_1^2, x_2^2, x_4^2\}$. Necessarily two bits are needed to re-encode the bound set variables to distinguish the three cut set nodes. Hence, $|F_\lambda| \geq 2$.

B. Finite State Machine

A finite state machine (FSM) can be described by a six-tuple $(I, O, Q, q_1, \Delta, \Omega)$, where I is the input alphabet, O is the output alphabet, $Q \neq \emptyset$ is a finite set of states, $q_1 \in Q$ is the initial state, $\Delta : Q \times I \rightarrow Q$ is the state transition function, $\Omega : Q \times I \rightarrow O$ is the output function. A machine is completely specified if for every state in Q under every input, its output and next state are defined; otherwise, it is incompletely specified. An FSM can be alternatively represented as a state transition graph (STG).

[1] In time-frame folding application, only disjoint decomposition, i.e., $X_\lambda \cap X_\mu = \emptyset$, needs to be considered.

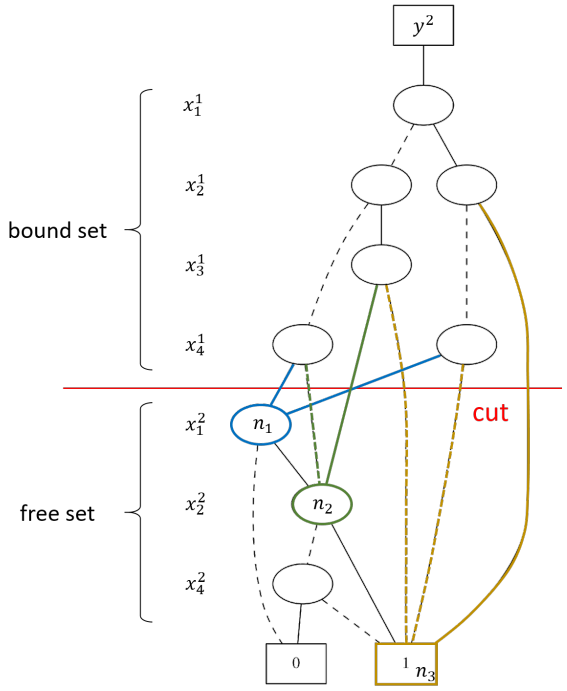


Fig. 2: BDD based functional decomposition.

C. Sequential Circuit and Time-Frame Expansion

An FSM can be implemented by a sequential circuit, which consists of combinational logic netlists realizing the transition and output functions of the FSM and flip-flops holding current state values.

The operation of a sequential circuit can be seen as an iterative combinational circuit that repeats the same computation but taking timestamped inputs. In time-frame expansion/unfolding, a sequential circuit is unrolled to construct an iterative combinational circuit. This is done by cascading duplicated sequential circuits, where the input and output of the flip-flops in the adjacent time-frames are connected together. In this paper, the initial values of the flip-flops (initial state) is constant-propagated throughout the time-frames. Therefore, after expansion, the primary output functions of each time-frame in the expanded circuit can be viewed as a purely combinational logic which depends on the primary inputs of all the previous time-frames.

Example 2: Figure 3 shows the circuit structure of `s27`, where x_i denotes the i^{th} primary input variable, y denotes the primary output variable, and z_i and z_i' denote the current- and next-state variables, respectively, of the i^{th} flip-flop. Let v^t denote the variable v instantiated at the t^{th} time-frame. Figure 4 shows the circuit of `s27` after three time-frames of expansion and simplification with constant propagation of the

initial state values $(z_1^0, z_2^0, z_3^0) = (0, 0, 0)$. Note that after the time-frame expansion all primary output functions are purely combinational, and after further circuit simplification the state transition functions cannot be clearly identified.

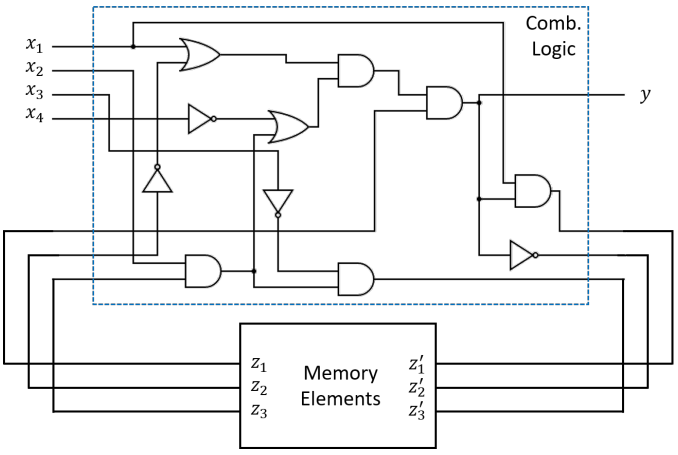


Fig. 3: Sequential circuit s27.

III. Problem Formulation

The problem of *time-frame folding* can be stated as follows.

Problem Statement 1 (Time-Frame Folding): Given a k-iterative combinational circuit $\mathcal{C}_C$ with inputs $X^1, \ldots, X^k$ for $X^t = \{x_1^t, \ldots, x_n^t\}$ and outputs $Y^1, \ldots, Y^k$ for $Y^t = \{y_1^t, \ldots, y_m^t\}$, find a sequential circuit $\mathcal{C}_S$ with inputs $X = \{x_1, \ldots, x_n\}$ and outputs $Y = \{y_1, \ldots, y_m\}$ such that the input-output behavior of $\mathcal{C}_S$ within the first k time-frames is the same as that of $\mathcal{C}_C$. Moreover, the number of states of $\mathcal{C}_S$ is minimized.

Note that the statement makes no assumption on the circuit structure of $\mathcal{C}_C$ but only its inputs and outputs in an iterative form, crucial for time-frame folding.

IV. Algorithm

A. Overview of Algorithmic Flow

The computation flow is shown in Figure 5. Given as input an iterative combinational circuit $\mathcal{C}_C$ with inputs $X^1, \ldots, X^T$ for $X^t = \{x_1^t, \ldots, x_n^t\}$ and outputs $Y^1, \ldots, Y^T$ for $Y^t = \{y_1^t, \ldots, y_m^t\}$, the algorithm returns a sequential circuit with inputs $X = \{x_1, \ldots, x_n\}$ and outputs $Y = \{y_1, \ldots, y_m\}$ consistent with $\mathcal{C}_C$ in T time-frames. It consists of the following steps: 1) state identification by functional decomposition, 2) state transition reconstruction, 3) state minimization, and 4) state encoding. The steps are detailed in the following subsections.

B. State Identification via Functional Decomposition

Given an iterative combinational circuit $\mathcal{C}_C$ with inputs $X^1, \ldots, X^T$ for $X^t = \{x_1^t, \ldots, x_n^t\}$ and outputs $Y^1, \ldots, Y^T$ for $Y^t = \{y_1^t, \ldots, y_m^t\}$, we show that the notion of states at time-frame t is induced by the output functions of $Y^{t+1}, \ldots, Y^T$. Note that the outputs Y^t observed at time t induce an equivalence relation on the set of input assignments $[\![X^1 \cup \ldots \cup X^t]\!]$. Effectively, the equivalence relation forms a partition on $[\![X^1 \cup \ldots \cup X^t]\!]$. Assume that the partition on $[\![X^1 \cup \ldots \cup X^t]\!]$ induced by the equivalence relation imposed by the outputs $Y^{t+1}, \ldots, Y^T$ has k cells (equivalence classes). Then we know the signals communicating from iteration t to iteration $t+1$ in circuit $\mathcal{C}_C$ (i.e., the information of inputs $X^1, \ldots, X^t$ needed to compute outputs $Y^{t+1}, \ldots, Y^T$) must have at least $\lceil \log_2 k \rceil$ bits. In the functional decomposition viewpoint of Figure 1, by decomposing the hyperfunction f of the output functions of $Y^{t+1} \cup \ldots \cup Y^T$ with bound set variables $X_\lambda = X^1 \cup \ldots \cup X^t$ and free set variables $X_\mu = X^{t+1} \cup \ldots \cup X^T \cup A$, where A is the set of pseudo input variables introduced to encode functions $Y^{t+1} \cup \ldots \cup Y^T$, the number of bits needed to communicate from F_λ to f_μ in the picture of Figure 1 is at least $\lceil \log_2 k \rceil$. Essentially the k equivalence classes correspond to the minimum states needed to distinguish the input assignments $[\![X^1 \cup \ldots \cup X^t]\!]$ for the outputs $Y^{t+1}, \ldots, Y^T$ to produce correct valuation. Let $Q^t = \{q_1^t, ..., q_k^t\}$ be the *states* representing the k equivalence classes, and let $\tau^t = \{\tau_{q_1^t}, \ldots, \tau_{q_k^t}\}$ be the set of *transition conditions*, that is, characteristic functions, each characterizing a set of equivalent input assignments in an equivalence class of $[\![X^1 \cup \ldots \cup X^t]\!]$. Then Q^t and τ^t can be obtained from ROBDD-based functional decomposition by noting that Q^t corresponds to the cut set and τ^t corresponds to the path conditions from the root node leading to the cut set nodes. In the sequel, we let $S^t = \{(q_1^t, \tau_{q_1^t}), \ldots, (q_k^t, \tau_{q_k^t})\}$ be the set of state and transition condition pairs at time t.

Example 3: To demonstrate how Q^t and τ^t are obtained from ROBDD-based functional decomposition, we take y^2 in Figure 2 as an example. To compute S^1, we build the hyperfunction $h = \alpha y^2 + \neg\alpha y^3$ of the output functions y^2 and y^3 as illustrated in Figure 6a. By performing functional decomposition on h, we obtain $S^1 = \{(q_1^1, \tau_{q_1^1}), (q_2^1, \tau_{q_2^1}), (q_3^1, \tau_{q_3^1}), (q_4^1, \tau_{q_4^1})\}$, where $\tau^1 = \{\neg x_2^1 x_4^1,\ \neg x_1^1(x_2^1 x_3^1 + \neg x_2^1 \neg x_4^1),\ x_1^1(x_2^1 x_3^1 + \neg x_2^1 \neg x_4^1),\ x_2^1 \neg x_3^1\}$.

It should be noted that to compute S^1 both functions y^2 and y^3 are needed. Considering only y^2 for the derivation of S^1 would be flawed due to the fact that two states in Q^1 that seem to be equivalent at output y^2 may possibly be distinguished at output y^3. Essentially the partition induced by both y_2 and y_3 is a refinement of the partition induced by y_2 only.

For S^2 derivation, functional decomposition on y^3 should be performed as is illustrated in Figure 6b.

Given an iterative combinational circuit $\mathcal{C}_C$, the state identification procedure for computing $S^0, \ldots, S^T$ is outlined in Algorithm 1. In line 1, S^0 and S^T are singleton sets as Q^0 has a single initial state q_1^0 and Q^T has a single don't-care destination state q_*^T. Moreover, the transition conditions to q_1^0 and q_*^T are tautologies. In lines 2-8, S^t for $t = 1, \ldots, T-1$ is computed through functional decomposition in line 7 on the hyperfunction encoded in line 4. Procedure *HyperEncode* encodes the output functions $Y^{t+1}, ..., Y^T$ into a single-output function h using the set A of fresh new variables $\alpha_1, \ldots, \alpha_k$ for $k = \lceil \log_2(|Y^{t+1}| + \cdots + |Y^T|) \rceil$. Procedure *Decompose* performs functional decomposition on the hyperfunction h and

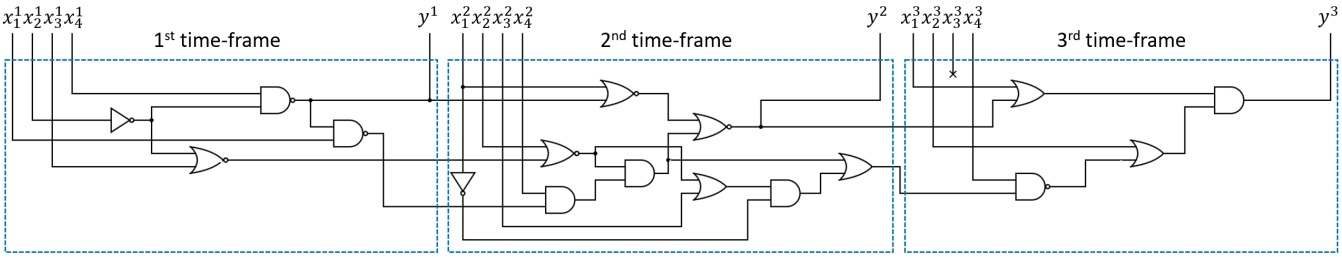


Fig. 4: Time-frame expanded circuit of s27, with initial state propagation and simplification.

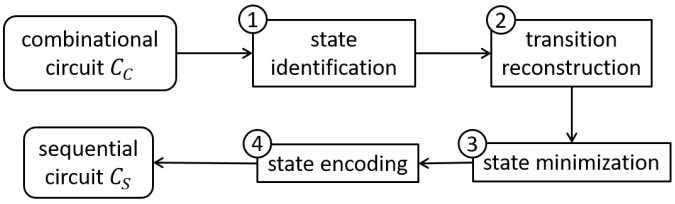


Fig. 5: computation flow

extract the cut set and corresponding transition conditions.

Algorithm 1 StateIdentify

Input: $\mathcal{C}_C$ with inputs $X^1, ..., X^T$ and outputs $Y^1, ..., Y^T$
Output: $\{S^0, S^1, ..., S^T\}$
1: $S^0 := \{(q_1^0, 1)\}$; $S^T := \{(q_*^T, 1)\}$;
2: **for** $t = 1, ..., T-1$ **do**
3: $k := \lceil \log_2(|Y^{t+1}| + \cdots + |Y^T|) \rceil$;
4: $h := HyperEncode(Y^{t+1} \cup ... \cup Y^T, A = \{\alpha_1, \ldots, \alpha_k\})$;
5: $X_\lambda := X^1 \cup ... \cup X^t$;
6: $X_\mu := X^{t+1} \cup ... \cup X^T \cup A$;
7: $S^t := Decompose(h, X_\lambda, X_\mu)$;
8: **end for**
9: **return** $\{S^0, S^1, ..., S^T\}$;

C. Transition Reconstruction

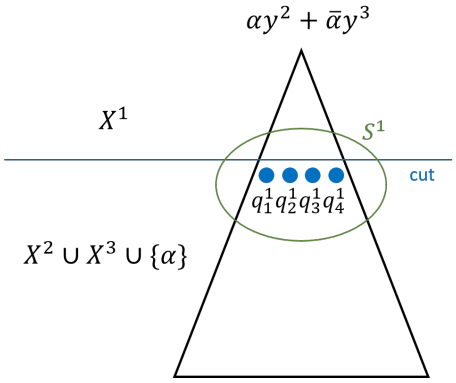
With the sets $S^0, \ldots, S^T$ of state and transition condition pairs being obtained, the next step is to determine the transitions among the states and construct the state transition graph.

Given an iterative combinational circuit $\mathcal{C}_C$, and the sets S^0, ..., S^T as input, Algorithm 2 computes, for every pair (q_i^{t-1}, q_j^t) of states in adjacent two time-frames, the input condition and output response under the transition from q_i^{t-1} to q_j^t. Essentially, the input transition condition can be characterized by the QBF

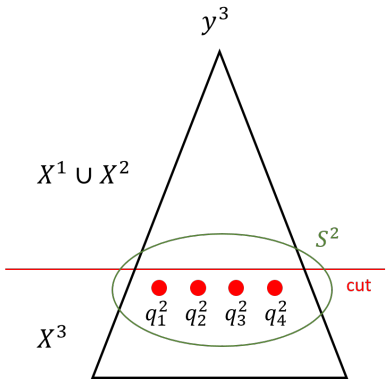
$$\varphi_{i,j}^t = \exists X^1, \ldots, X^{t-1}.\tau_{q_i^{t-1}} \wedge \tau_{q_j^t} \quad (1)$$

and the output transition response can be characterized by the set of QBFs

$$\psi_{i,k}^t = \exists X^1, \ldots, X^{t-1}.\tau_{q_i^{t-1}} \wedge y_k^t \quad (2)$$



(a) Functional decomposition for S^1 derivation.



(b) Functional decomposition for S^2 derivation.

Fig. 6: State identification.

for $y_k^t \in Y^t$. In line 5, the procedure *TransitionTuple* returns the four-tuple $(q_i^{t-1}, q_j^t, \varphi_{i,j}^t, \{\psi_{i,k}^t \mid y_k^t \in Y^t\})$. The algorithm returns the collected four-tuples R for all state transitions. According to R, one can construct a state transition graph (STG).

Example 4: To illustrate, we derive the input condition for the transition from q_1^1 to q_1^2 shown on Figure 7a, where $\tau_{q_1^1} = \neg x_2^1 x_4^1$, $\tau_{q_1^2} = \neg x_2^1 x_4^1 \cdot (\neg x_1^2(\neg x_2^2 + x_3^2) + x_1^2 \neg x_2^2 x_4^2) + \neg x_1^1(x_2^1 x_3^1 + \neg x_2^1 \neg x_4^1) \cdot \neg x_2^2 x_4^2$. $y^2 = (\neg x_2^1 x_4^1 x_1^2 + \neg x_1^1(\neg x_2^1 \neg x_4^1 + x_2^1 x_3^1)) \cdot (x_2^2 + \neg x_4^2) + x_1^1(x_2^1 + \neg x_4^1) + \neg x_1^1 x_2^1 \neg x_3^1$. The input transition condition and the output transition response can be derived by: $\varphi_{1,1}^2 = \exists X^1.\tau_{q_1^1} \wedge \tau_{q_1^2} = \neg x_1^2(\neg x_2^2 +$

Algorithm 2 TransitionReconstruct

Input: $\mathcal{C}_C$, $\{S^0, \ldots, S^T\}$
Output: transition four-tuples R

1: $R := \emptyset$;
2: **for** $t = 1, ..., T$ **do**
3: **foreach** $(q_i^{t-1}, \tau_{q_i^{t-1}}) \in S^{t-1}$ **do**
4: **foreach** $(q_j^t, \tau_{q_j^t}) \in S^t$ **do**
5: $R := R \cup TransitionTuple(\tau_{q_i^{t-1}}, \tau_{q_j^t}, Y^t)$;
6: **end for**
7: **end for**
8: **end for**
9: **return** R;

$x_3^2)+x_1^2\neg x_2^2x_4^2$ and $\psi_1^2 = \exists X^1.\tau_{q_1^1} \wedge y^2 = x_1^2(x_2^2+\neg x_4^2)$, which corresponds to the edge labeled with "00--/0, 011-/0, 10-1/0" between q_1^1 and q_1^2 in Figure 7a.

D. State Minimization

Notice that although by functional decomposition we guarantee that $|S^t|$ is minimum, the STG constructed from R may not be state minimum. It is because equivalent states in different time-frames are not yet considered. In the STG derived from time-frame folding, there is a unique initial state q_1^0 and final don't-care state q_*^T. As the STG is incompletely specified at state q_*^T, the flexibility provides room for state minimization. In our implementation, we apply the SAT-based exact minimization algorithm `MeMin` [16] for STG simplification.

Example 5: The STG in Figure 7a can be minimized to that in Figure 7b. The number of states reduces from 10 (including the unspecified state q_*) to 5. In Figure 7b, each state is annotated with its compatible states in Figure 7a. In each time-frame except for the last, the states being identified are minimized such that none of them can be merged into the same state. For instance, the states reached at the first time-frame q_1^1, q_2^1, q_3^1 and q_4^1 in Figure 7a correspond to different states q_3', q_2', q_5' and q_1', respectively, in Figure 7b.

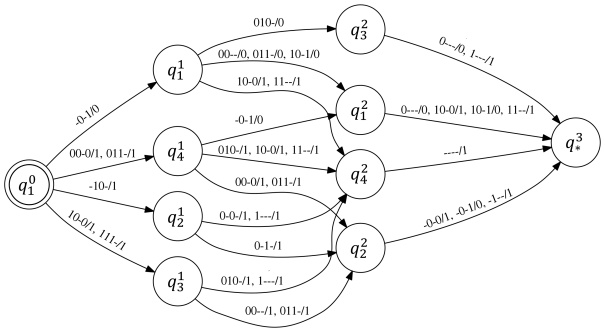
E. State Encoding

To transform an STG into a sequential circuit, a final state-encoding step has to be performed. Let Q be the state set of the STG. In our implementation, we try two different encoding schemes: 1) natual encoding, which uses $\lceil \log_2 |Q| \rceil$ bits, and 2) one-hot encoding, which uses $|Q|$ bits, each of which represents a state in Q.

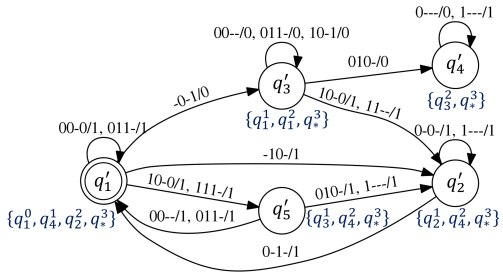
V. Implementation Issues

To improve state identification, we make two modifications to the *StateIdentify* algorithm:

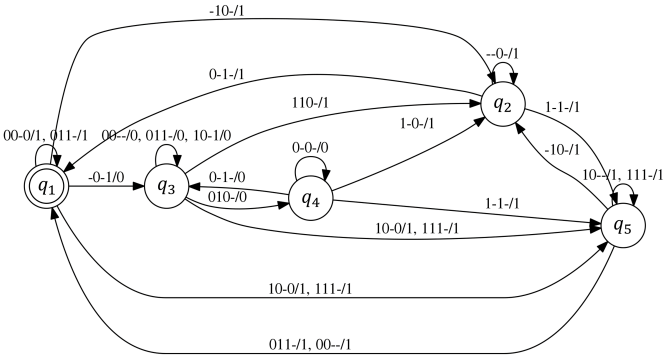
- Reverse-chronological order enumeration: The index t in the for-loop in line 2 enumerates from 1 to $T-1$. As t increases, the number $|Y^t \cup ... \cup Y^T|$ of functions that have to be encoded decreases. Also there is a huge overlap of functions to be encoded at two consecutive time-frames t and $t+1$, which is $\{Y^{t+1}, ..., Y^T\}$. As a result, by reversing the enumeration order for t from $T-1$ to 1, the hyperfunction h can be built incrementally by adding Y^t to h one at a time in each iteration.
- Re-encoding hyperfunction: Now that the state and transition condition pairs identified at each time-frame are constructed in a reverse-chronological order, after we obtain S^t by decomposing the hyperfunction h built at time frame $t+1$, the variable in X^{t+1} is no longer relevant in deciding partition of $[\![X^1 \cup ... \cup X^t]\!]$. Hence, we can re-encode h into a more compact representation with less variables to reduce the circuit size. Essentially the variables X^{t+1} in h can be replaced with a new set of variables of size $\lceil \log_2 |S^t| \rceil$ in a way preserving the cut set nodes of h. Therefore h can be represented more compactly. The re-encoded hyperfunction is then be passed down to the next iteration.



(a) STG from folding 3 time-frames (before state minimization).



(b) STG from folding 3 time-frames (after state minimization).



(c) STG of original `s27` circuit.

Fig. 7: State transition graphs.

VI. Experimental Results

The proposed computation of state identification and transition reconstruction (Algorithms 1 and 2) was implemented in the C++ language as a command, named "`timefold`", within the `ABC` system [17] and used `CUDD` [18] as the underlying BDD package. For state minimization, package `MeMin` [16] was used.

Our method was evaluated with respect to three sets of benchmark circuits. Two were obtained from unfolded and simplified ISCAS and ITC circuits, and one was obtained from QBF solving of adaptive homing sequences [8]. The experiments were conducted on a server with Intel(R) Xeon(R) CPU E5-2620 v4 of 2.10 GHz and 126 GB RAM. A timeout limit of 300 seconds is imposed on command `timefold`, and the same limit is imposed on `MeMin` for state minimization. Also an expansion limit of 5000 time-frames was imposed.

The results on ISCAS and ITC benchmarks are shown in Table I, where Columns 2-5 list the numbers of primary inputs, primary outputs, latches, and AIG nodes, respectively, after optimization of the original sequential circuits, Column 6 lists the maximum time-frames that can be expanded and folded back within the timeout limit, Columns 7 and 8 list the numbers of states of the folded circuit before and after state minimization, respectively, and Column 9 list the number of flip-flops in the resultant sequential circuit under natural encoding. For an entry in the table containing two values separated by "/", it indicates that `MeMin` reached its timeout limit before `timefold` reached its maximum number of time-frames. The value on the left of "/" shows the data that both `timefold` and `MeMin` are executed successfully, while the value on the right shows the data that only `timefold` can be done within the timeout limit. Circuits `b01` and `b02` reached the 5000 time-frame limit and are marked with the "*" sign.

From the table, the numbers of foldable time-frames within 300 seconds vary to some extent, roughly proportional to the growth rate of the number of states. On the other hand, the performance of `MeMin` exhibited somewhat non-robustness. For example, for `s382` expanded with 50 time-frames, the 10617 states can be successfully minized to 1282 states within 300 seconds; in contrast, for `s713` expanded with 3 time-frames, the 11 states cannot be minimized within 300 seconds. For the homing sequence benchmarks, the results are shown in Table II. As the depths of the adaptive homing strategies are not large, our method successfully generates all sequential circuits.

To better understand the relation among the number of states, the number of time-frames, and the runtime, circuits `b07`, `b18`, `s386`, `s1494`, and `s15850` were selected for study. Figure 8 shows the relation between the number of states and the number of expanded time-frames. It can be observed that the number of states before minimization (right y-axis) constantly increased with the number of time-frames, whereas the number of states after minimization (left y-axis) tended to saturate after a certain number of time-frames. (Note that the left and right y-axes are of different scales.) This phenomenon is expectable as all inequivalent states should be distinguished eventually. On the other hand, Figure 9 shows the relation between the total runtime of `timefold` and `MeMin` and the number of time-frames. Their positive correlation is expected.

We verified the consistency between the constructed sequential circuits and their corresponding expanded iterative combinational circuits. In the cases of our experiments, we observed that the constructed sequential circuit tends to become sequentially equivalent to its original sequential circuit when the number of expanded time-frames is sufficiently large. We call this phenomenon as a *fixed point*. However, the sequential equivalence may not happen immediately at the time-frame when the number of states starts to saturate. Let q_1 be the

TABLE I: Results of time-frame folding on ISCAS and ITC benchmarks.

circuit	#PI	#PO	#FF	#gate	#frame	#state	#state-m	#FF'
b01	2	2	5	38	5000*	22493	18	5
b02	1	1	4	16	5000*	9997	8	3
b03	4	4	21	55	76	21182	631	10
b04	11	8	66	351	4	132	130	8
b05	1	26	34	422	569	34268	69	7
b06	2	6	8	29	348	4139	13	4
b07	1	8	39	308	491	35221	83	7
b08	9	4	21	125	66	24215	798	10
b09	1	1	28	120	24/28	10241/42981	3795/-	12/16
b10	11	6	17	162	16/22	3248/8116	602/-	10/13
b11	7	6	30	449	15/20	2542/24122	676/-	10/15
b12	5	6	119	919	97	7621	1004	10
b13	10	10	45	171	117/141	10276/127884	139/-	8/17
b14	32	54	215	3833	2	3	2	1
b15	36	70	415	6897	6	11	8	3
b17	37	97	605	8069	7/11	103/13826	93/-	7/14
b18	37	23	129	2351	76	12756	382	9
b20	32	22	429	8417	2	3	1	0
b21	32	22	429	8339	2	3	1	0
b22	32	22	611	12693	2	3	1	0
s27	4	1	3	7	193	960	5	3
s208.1	10	1	8	48	182	12556	129	8
s298	3	6	14	74	50	5166	135	8
s344	9	11	15	96	5/31	1262/27426	863/-	10/15
s349	9	11	15	96	5/35	1262/31490	863/-	10/15
s382	3	6	21	87	50	10617	1282	11
s386	7	7	6	81	105	1328	13	4
s400	3	6	21	89	50	10617	1282	11
s420.1	18	1	16	101	184	12814	129	8
s444	3	6	21	95	50	10617	1282	11
s510	19	7	6	207	45/148	967/5804	44/-	6/13
s526	3	6	21	91	50	10651	1285	11
s641	35	24	14	97	2/4	3/75	2/-	1/7
s713	35	23	14	98	2/3	3/11	2/-	1/4
s820	18	19	5	216	56	1192	24	5
s832	18	19	5	211	51	1072	24	5
s838.1	34	1	32	213	184	12814	129	8
s953	16	23	29	277	9/18	270/4634	111/-	7/13
s1196	14	14	18	397	1/2	2/460	1/-	0/9
s1238	14	14	18	384	1/2	2/460	1/-	0/9
s1423	17	5	73	433	6/7	498/2731	396/-	9/12
s1488	8	19	6	500	67	2670	48	6
s1494	8	19	6	498	65	2574	48	6
s5378	35	49	127	740	1	2	1	0
s9234.1	36	39	129	750	0/2	-/10	-/-	-/4
s13207	31	121	193	550	10/12	4666/16042	4665/-	13/14
s13207.1	62	152	253	693	0	-	-	-
s15850	14	87	128	374	643/2501	650/2508	11/-	4/12
s15850.1	77	150	436	2324	0	-	-	-
s35932	35	320	1472	7297	5	90	53	6
s38417	28	106	1345	7197	2/3	6/114	5/-	3/7
s38584	12	278	784	4479	5	162	141	8
s38584.1	38	304	1141	8250	0	-	-	-

TABLE II: Results of time-frame folding on homing sequence benchmarks.

name	#PI	#PO	#gate	#frame	#state	#state-m
5s2i0_c	3	3	1	3	6	4
5s2i0_r	3	3	0	3	4	1
5s2i2_c	4	4	1	4	9	4
5s2i2_r	4	4	2	4	8	5
10s5i1_c	12	12	175	4	35	29
10s5i4_c	12	12	105	4	30	24

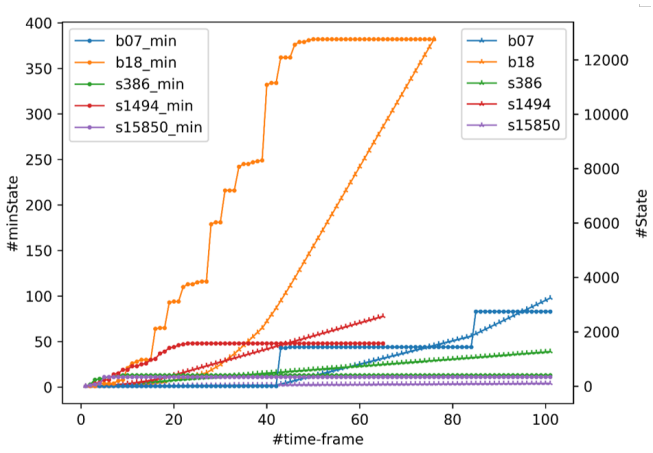


Fig. 8: #state vs. #time-frame.

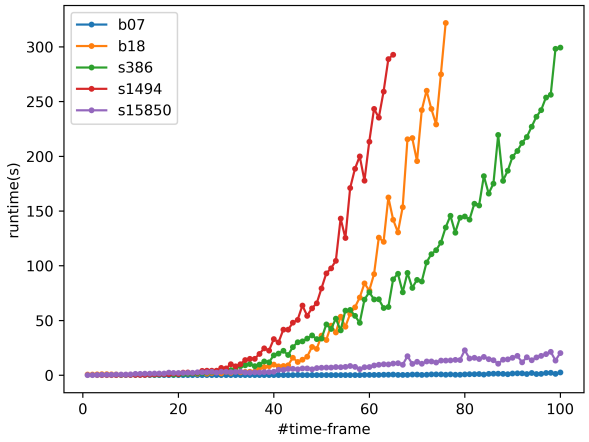


Fig. 9: Total runtime vs. #time-frame.

initial state of the state transition graph, $m_{i,j}$ be the length of the shortest path from state q_i to state q_j, and $n_{i,j}$ be the length of the shortest sequence distinguishing states q_i and q_j. Also let $\mathbf{m}$ be the maximum length among the shortest paths from the initial state to any other states, i.e., $\mathbf{m} = \max\{m_{1,j}\}$, for any $q_j \in Q, j \neq 1$; let $\mathbf{n}$ be the maximum length among the shortest sequences distinguishing any state pairs, i.e., $\mathbf{n} = \max\{n_{i,j}\}$, for any $q_i, q_j \in Q, i \neq j$. In fact, if the reachable state sets grow monotonically during time-frame expansion, then the fixed point is guaranteed by expanding the circuit no greater than $\mathbf{m} + \mathbf{n}$ time-frames.

Table III shows the time-frame numbers when the number of states starts to saturate and when the obtained circuit starts to become sequentially equivalent to the original circuit. Note that not every considered circuit is listed in Table III, because some of them are not able to reach these two conditions within their maximally expanded time frames. Also the table shows the numbers of flip-flops and gates of the folded sequential circuit under two different encoding schemes, and the corresponding reduction ratio on the numbers of flip-flops and gates compared to those of its corresponding original sequential circuit. As generally observed, natural encoding can result in fewer flip-flops, but require more gates, while one-hot encoding can achieve better gate count reduction, but require more flip-flops.

To verify that our proposed method indeed has the ability in circuit size compaction, we compared the sizes of the expanded combinational circuits to their folded sequential circuits in terms of AIG nodes. The ISCAS and ITC benchmark circuits selected for comparison are the ones that have reached the number of time-frames to observe sequential equivalence, and are expanded by that number of time-frames. Note that for time-frame folding, there is no need to expand more than that number of time-frames, since the folded sequential circuit will remain the same, while the expanded combinational circuit will continue to grow in size. Additionally, homing sequence benchmarks are also included for comparison. The results are plotted in Figure 10, where black data points correspond to ISCAS and ITC benchmark circuits, and the blue ones correspond to homing sequence benchmarks. Both natural and one-hot encoding schemes were applied, and the one resulted in a smaller circuit size was taken for comparison. The data points on the right of the gray dotted line correspond to the cases where the obtained sequential circuits are of size smaller than their combinational counterparts. We observed that larger circuits tend to benefit more from our method, as the combinational circuits with over 1500 AIG nodes, when folded into sequential circuits, are all reduced significantly in size. Note that the upper-most (worst-case) point in Figure 10 is the circuit `b03` expanded with 14 time-frames. Although time-frame folding does not achieve compaction in this case, it is expected that, when more time-frames are to be expanded, the iterative combinational circuit size will keep growing while the folded sequential circuit size will remain the same.

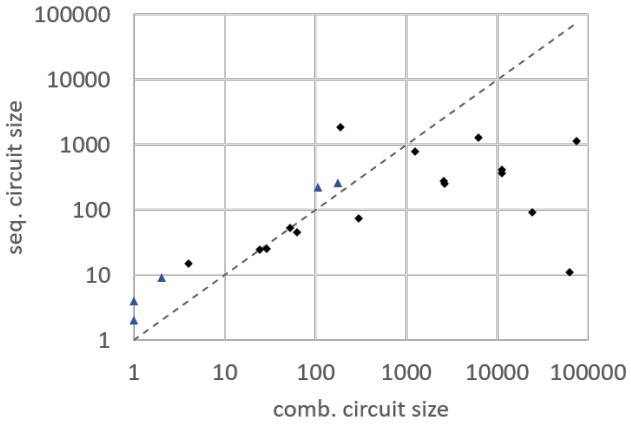


Fig. 10: Circuit size comparison.

TABLE III: Results on folding with fixed points reached.

circuit	#time-frame		expanded circuit	natural encoding				one-hot encoding			
	state saturate	fixed point	#gate	#FF	#gate	reduction (%)		#FF	#gate	reduction (%)	
b01	9	9	52	5	109	0.0	-186.8	18	53	-260.0	-39.5
b02	6	10	4	3	16	25.0	0.0	8	15	-100.0	6.3
b03	14	14	189	10	8947	52.4	-16167.3	631	1848	-2904.8	-3260.0
b05	69	133	62635	7	52	79.4	87.7	69	11	-102.9	97.4
b06	6	7	62	4	82	50.0	-182.8	13	45	-62.5	-55.2
b07	85	85	24438	7	91	82.1	70.5	83	94	-112.8	69.5
b08	55	55	6173	10	3395	52.4	-2616.0	798	1265	-3700.0	-912.0
b18	50	50	74461	9	2516	93.0	-7.0	382	1134	-196.1	51.8
s27	3	5	29	3	25	0.0	-257.1	5	42	-66.7	-500.0
s298	20	23	1243	8	1489	42.9	-1912.2	135	785	-864.3	-960.8
s386	8	9	297	4	124	33.3	-53.1	13	74	-116.7	8.6
s820	12	13	2558	5	276	0.0	-27.8	24	8639	-380.0	-3899.5
s832	12	13	2612	5	248	0.0	-17.5	24	10075	-380.0	-4674.9
s1488	23	23	11298	6	578	0.0	-15.6	48	406	-700.0	18.8
s1494	23	23	11367	6	526	0.0	-5.6	48	364	-700.0	26.9
s15850	5	5	24	4	28	96.9	92.5	11	24	91.4	93.6

VII. Conclusions

We have formulate the time-frame folding problem, and provided a computational solution based on functional decomposition for state identification and transition reconstruction. Our method guarantees the sequential circuit folded from an iterative combinational circuit is state minimized. Experimental results demonstrate the benefit of our method in circuit compaction from an iterative combinational circuit to its sequential counterpart. Our method can be useful in testbench generation, sequential synthesis of bounded strategies, and other applications.

Acknowledgments

The authors thank Kuan-Hua Tu for providing the homing sequence benchmarks, Natalia Kushik and Nina Yevtushenko for valuable discussions motivating this work, and Alan Mishchenko for helpful synthesis suggestions. This work was supported in part by the Ministry of Science and Technology of Taiwan under grants 105-2221-E-002-196-MY3, 105-2923-E-002-016-MY3, and 108-2221-E-002-144-MY3.

References

[1] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., 2006.

[2] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 193–207, 1999.

[3] S. Sandberg, "Homing and synchronizing sequences," in *Model-Based Testing of Reactive Systems: Advanced Lectures*, pp. 5–33, 2005.

[4] N. Kushik, J. López, A. Cavalli, and N. Yevtushenko, "Improving protocol passive testing through "gedanken" experiments with finite state machines," in *Proceedings of International Conference on Software Quality, Reliability and Security (QRS)*, pp. 315–322, 2016.

[5] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[6] J.-K. Rho, F. Somenzi, and C. Pixley, "Minimum length synchronizing sequences of finite state machine," in *Proceedings of Design Automation Conference (DAC)*, pp. 463–468, 1993.

[7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*. IOS Press, 2009.

[8] H.-E. Wang, K.-H. Tu, J.-H. R. Jiang, and N. Kushik, "Homing sequence derivation with quantified Boolean satisfiability," in *Proceedings of International Conference on Testing Software and Systems (ICTSS)*, pp. 230–242, 2017.

[9] A. Mishchenko, N. Een, R. Brayton, J. Baumgartner, H. Mony, and P. Nalla, "GLA: Gate-level abstraction revisited," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 1399–1404, 2013.

[10] J.-H. R. Jiang, J.-Y. Jou, and J.-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," in *Proceedings of Design Automation Conference (DAC)*, pp. 712–717, 1998.

[11] J.-H. R. Jiang and R. K. Brayton, "On the verification of sequential equivalence," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 686–697, 2003.

[12] R. Ashenhurst, *The Decomposition of Switching Functions*, vol. 29, pp. 74–116. Computation Lab, Harvard University, 1959.

[13] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 227–238, 1962.

[14] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, "Bdd based decomposition of logic functions with application to FPGA synthesis," in *Proceedings of Design Automation Conference (DAC)*, pp. 642–647, 1993.

[15] S.-C. Chang, M. Marek-Sadowdka, and T. Hwang, "Technology mapping for TLU FPGAs based on decomposition of binary decision diagrams," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 10, pp. 1226–1236, 1996.

[16] A. Abel and J. Reineke, "MEMIN: SAT-based exact minimization of incompletely specified Mealy machines," in *Proceedings of International Conference of Computer-Aided Design (ICCAD)*, pp. 94–101, 2015.

[17] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proceedings of International Conference on Computer Aided Verification (CAV)*, pp. 24–40, 2010.

[18] F. Somenzi, "CUDD: CU decision diagram package (release 2.4.1)," *University of Colorado at Boulder*, 2005.