On Synthesis of Speed-Independent Circuits at STG Level

Kuan-Jen Lin and Chi-Wen Kuo

Department of Electrical Engineering
Chinese Junior College of Technology and Commerce
Taipei, Taiwan, R.O.C.

Abstract: Synthesizing hazard-free asynchronous circuits directly at Signal Transition Graph (STG) level has been shown to need significantly less CPU time than approaches at the state-graph[10, 16, 4]. However, all previous methods at STG level were based on sufficient conditions only. Hence, the synthesized circuit results generally are inferior, due to the incomplete transformation. In this paper, we present a new Characteristic Graph (CG) to encapsulate all feasible solutions of the original STG in reduced size, which compares favorably with the state graph approach. The requirements of speed independent circuits can then be completely transformed into the CG. Furthermore, we derive a necessary and sufficient condition for speed independent implementation based on a predefined general circuit model, which has not yet been reported. With CGs and this condition, we develop a heuristic synthesis algorithm which derives solutions similar to the state-graph approach while requiring significantly less CPU time.

1 Introduction

Asynchronous circuit design has received much attention in recent years. There are two reasons for the revival of interest. First, asynchronous circuits feature the advantages of clock-skew-freeness as well as low peak power as compared to synchronous circuits. These advantages are expected to become more significant as progress in semiconductor technology leads to even denser and more complex circuits. Recently, attractive designs have been reported in [11, 14, 15]. These designs show the potential of asynchronous circuits. Secondly, new formal methods have been developed to automatically synthesize asynchronous circuits which are able to handle circuits of larger size with far less restriction than can classical methods. Synthesis of asynchronous circuits is the main concern of this paper.

Among the newly developed methodologies, the STG approach is one of the most attractive approaches. Many synthesis methods have been proposed to derive asynchronous circuits for STGs. These methods, due to the different intermediate representations during synthesis, can be classified into two approaches: the state graph approach and the STG-level approach. The state graph approach[3, 9, 1, 7], derives asynchronous circuits from the corresponding state graph of STG. The appealing feature of such an approach is that well-known logic synthesis tools can be modified for such a purpose. However, the complexity of the synthesis process then depends on the state graph, the size of which is exponential with respect to the number of signals in an STG. Alternatively, [10, 16, 4] proposed deriving circuits directly from STGs without explicitly or implicitly using the state graph. As a result, they have reported significant improvement in synthesis time. However, they generally need more restrictions on the STG specification than the state-graph approach. Furthermore, all problem transformations from the state graph to the models at the STG level were based on sufficient conditions only. Hence, the synthesized circuit results generally are inferior.

In this paper, we will present a synthesis method which accepts specification in STGs and results in speed independent asynchronous circuits. The realized circuits can operate correctly under the assumption that the unbounded delays are concentrated on the gates and the delay difference between fork lines is less than one gate delay, i.e., does not violate the isochronic fork assumption[13]. The automatic synthesis is based on the new Characteristic Graph (CG) which can encapsulate all feasible solutions of the original STG in reduced size, which compares favorably with the state graph approach. The requirements of hazard-freeness can then be completely transformed into the CG. Furthermore, we derive a necessary and sufficient condition for speed independent implementation based on a predefined general circuit model. With CGs and this condition, we develop a synthesis algorithm which results in solutions comparable to the state-graph approach while requiring significantly less CPU time.

The remainder of this paper is organized as follows. Section 2 briefly reviews relevant results for the STG approach. Section 3 defines our realization circuit model. Section 4 defines the new CG and shows how to completely transform requirements of speed independent circuits into the CG. A necessary and sufficient condition of hazard-free realization is described in Section 5, which decides the least area upper bound. In Section 6, a heuristic method for circuit realization is proposed and evaluated using examples from *sis*[3] and a set of STGs with thousands of states. Finally, a conclusion is given.

2 Signal Transition Graph

A Signal Transition Graph, STG, can be viewed as an interpreted Petri net in which each transition is interpreted as a physical signal transition of asynchronous behavior[2]. In an STG, the transitions of a signal x, denoted by $x+$ and $x-$, are the rising ($0 \rightarrow 1$) and falling ($1 \rightarrow 0$) transitions, respectively. If the complement signal $\overline{x}$ is used, $x+$ and $x-$ are its falling- and rising-transitions, respectively. If a signal contains only two transitions, then it is *single-cycle*; otherwise, it is *multi-cycle*. For a multi-cycle signal, /number is used to distinguish its individual transitions having the same direction (i.e., *x+/j* represents the *j-th* rising transition of signal x). Henceforth, the notation $x*$ will denote a certain transition of signal x (i.e., either an $x+$ or an $x-$) and $\overline{x*}$ its inverse transition(i.e., either an $x-$ or an $x+$).

According to the complexity of the underlying Petri net, STGs under our consideration are classified into two classes: STG/MG (Marked Graph) and STG/FC (Free-Choice Petri net). In an STG/FC, the relations among transitions can be classified into three types: *concurrent, ordered* and *in conflict*. If two transitions

ASP-DAC '97
0-89791-851-7/$5.00 ©1997 IEEE

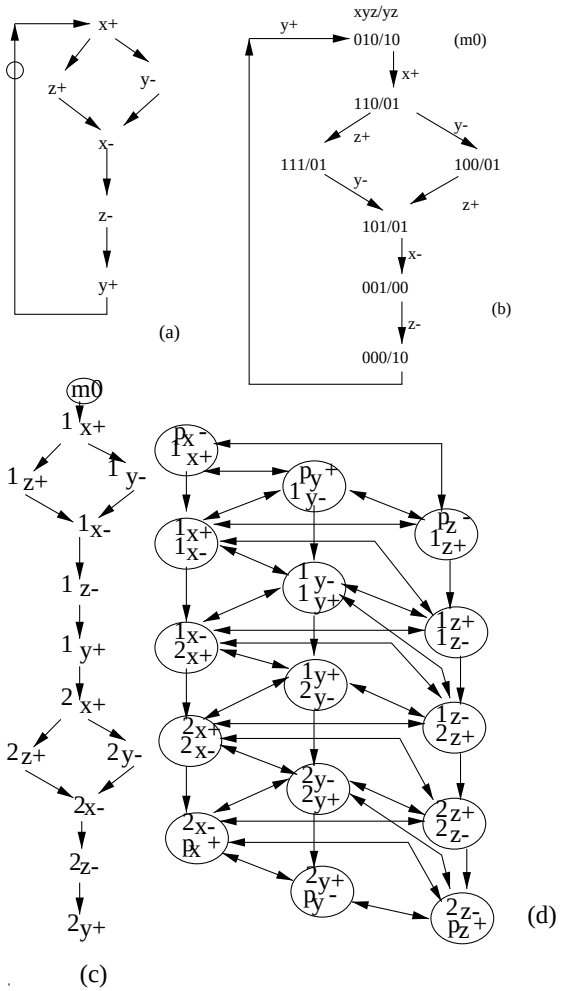

Figure 1: (a) A simple STG example, (b) its corresponding state graph, (c) two periods of its unfolding and (d) its corresponding CG (A bidirectional arc $x* \leftrightarrow y*$ means both $x* \rightarrow y*$ and $y* \rightarrow x*$ exist).

both can be covered by some MG-component and enabled at the same time, they are concurrent. For two nonconcurrent transitions, if they both can be covered by some MG-component, then they are ordered. Otherwise, they are in conflict. For the STG class under consideration (having a live, safe and strongly connected underlying net), we use $x_1* \Rightarrow x_2* \Rightarrow ... \Rightarrow x_k*$ to denote that $x_1*, x_2*, ...x_k*$ are ordered with each other and occur in a cycle as $x_1*, x_2*, ...x_k*, x_1*, x_2*, ...$ within the MG-component covering them. Clearly, $x_1* \Rightarrow x_2* \Rightarrow ... \Rightarrow x_k*$ is exactly the same as $x_k* \Rightarrow x_1* \Rightarrow ... \Rightarrow x_{k-1}*$. In an STG/FC, an arc between two ordered transitions may be redundant. Specifically, an arc $x_i* \rightarrow x_j*$ is not redundant only if there exists a marking in which all input arcs to x_j* already carry tokens except for the arc. Under such a marking, if x_i* is fired, x_j* can be enabled immediately. We call x_i* an enabling transition of x_j* and x_i an enabling signal. Obviously, an enabling signal of x_j* is an input signal used to generate x_j in the circuit. In Fig. 2(a), *a-/1* and $d+$ both are the enabling transitions of $b-$, and *a-/2* is the unique enabling transition of $e-$.

The STG has an equivalent finite-state-machine representation called State Graph (SG). The SG represents the STG concurrency as an interleaving of transitions. As with the total-state model in the classical asynchronous design, all the signals are directly considered as state variables, and their boolean values are used to encode states. To ensure that the state-assignment is consistent, a transition $x+$ ($x-$) can be enabled only in a state whose code for x is 0 (1). The following requirement was proposed by [2] for an STG to have consistent state-assignment and extended by [3].

Definition 1 (Liveness): An STG is live iff
(1) the underlying Petri net is live and safe;
(2) for each signal x, there is at least one SM-component initially marked with one token such that (a) it contains all the transitions of x, and (b) $x+$ and $x-$ occur alternately in the SM-component.

The SG of the live STG in Fig. 1(a) is shown in Fig. 1(b). One can see that it has consistent state-assignment. From the consistently encoded SG of a live STG, we can derive the *output function* (or next-state function in [3]) for each non-input signal. The part below the slash in each vertex of encoded state graph in Fig. 1(b) is the output function of the STG in Fig. 1(a). To ensure that each state-code (input vector) predicts a deterministic output-value, the following property is required.

Definition 2 (Complete State Coding (CSC))[2, 9]: A live STG has the CSC property iff any two states which enable different sets of non-input signal transitions have distinct state-codings.

For a live STG with the CSC property, we can derive the logic implementation of the output function for each non-input signal. We want to find a set of cubes to cover the on-set of the considered signal and not to intersect any state-code in its off-set. A cube c is said to be covered by another cube c' if literal $x_i \in c'$ then $x_i \in c$ and vice-versa. A state is said to be covered by some cube or in some cube if its state-code is covered by the cube. When a state covered by some cube is reached, we say that the cube is *set on*. After a cube is set on, when a state not in this cube is reached, it is *set off*. To enable a transition, some cube must be on. The following definition describes such a cube.

Definition 3 (Enabling Cube): Let $x_1*, x_2*, ...,$ and x_n* be all the enabling transitions of a transition $f*$ in a live STG. Let cube $C = c_1c_2...c_n$, where if x_i* is an x_i+, $c_i = x_i$; otherwise, $c_i = \overline{x_i}$, $i = 1, ..., n$. A cube is an *enabling cube* of $f*$ if (1) it covers all the states enabling $f*$, and (2) it is covered by cube C.

For transition *a+/1* in Fig. 2(a), the smallest enabling cube is $\overline{a}\overline{b}\overline{e}\overline{d}$, and the largest one $\overline{e}$.

3 Realization Circuit Model

Our hazard-free implementation is based on a practical architecture, consisting of two-level combinational logic implementations and an asynchronous memory-element (C-element) for each signal. Under this architecture, we introduce a realization circuit model which ensures that the circuit implementation is functional correct and hazard-free.

The realized circuit model introduced here is an extended version of our previous work, called the *single-cube realization circuit model (SCRCM)*[4], in which we primarily focused on polynomial-time heuristic realization without tackling complete transformation problem and gate sharing. The following definition describes the construction principles of the extended realization circuit model.

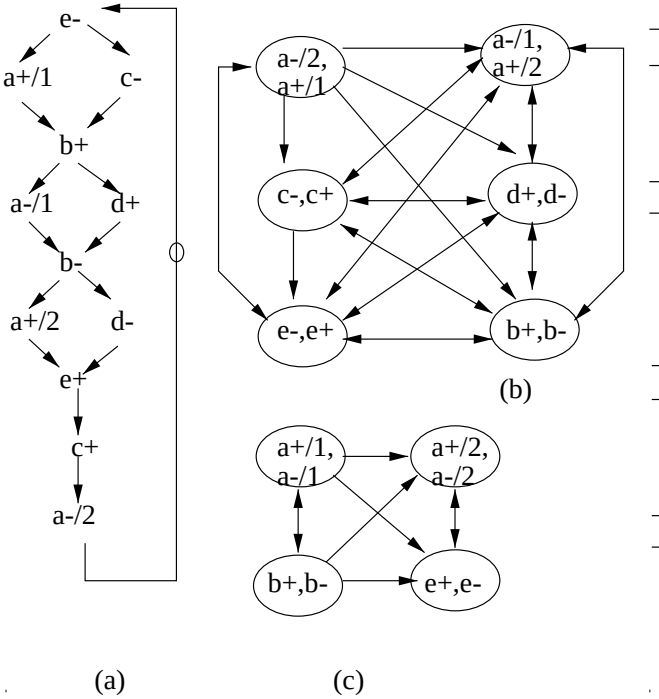

Figure 2: (a) An STG/MG G with multi-cycle signals. (b) The *Cg*(G, $a\bar{b}c\bar{d}e$,*a-/2)* of G. (c) The *Cg*(G, $\bar{a}\bar{b}\bar{e}$, {*a+/1, a+/2*}*)*.

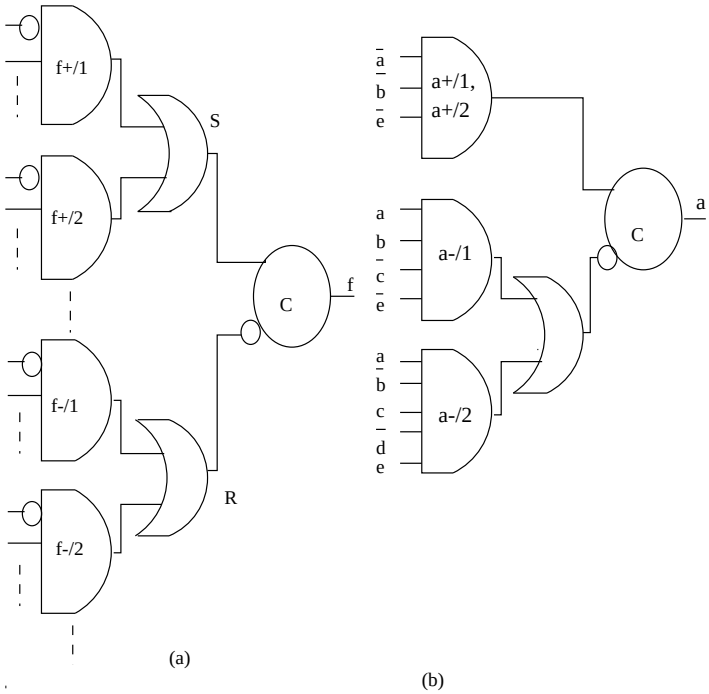

Figure 3: (a) The circuit template of ESCRCM. (b) The canonical implementation of signal a in Fig 2(a).

Definition 4 (Extended SCRCM): The ESCRCM for a non-input signal f in a live STG has the circuit configuration shown in Fig. 3(a). Each $f+$ ($f-$) is activated by exactly one AND gate in the set (reset) subcircuit. Two OR gates collect all AND outputs for rising transitions and falling transitions to set and reset the memory element, respectively. Each AND gate, possibly having inverters attached to its input terminals, implements a cube, AND_T, responsible for a transition set T of f. AND_T must satisfy the following requirements:
(1) AND_T is an enabling cube for each $f*$ in T.
(2) Each time, after an $f*$ in T is enabled, AND_T must be set off before any next $\overline{f*}$ is enabled.
(3) Once AND_T is set off, it remains off until any $f*$ in T is enabled again.

The requirements are intended to qualify the required waveform for pulses produced by the AND gate. Since AND_T is an enabling cube for any $f* \in T$, it turns on when any $f* \in T$ is enabled. Then, by (2), it turns off before a subsequent $\overline{f*}$ is enabled and then turns on again only when another $f* \in T$ or itself is enabled. Recently, Kondratyev *et al.*[7] also proposed a synthesis system based on the same circuit architecture as ESCRCM and also assumed that each transition was triggered by exactly one AND gate. They proposed the *Monotonous Covering* requirement for cubes, which requires the cube: (1) covers all enabling states; (2) changes at most once inside the associated excitation and quiescent region and; (3) does not cover any state outside the two regions. This is a sufficient condition for our ESCRCM because it does not allow gate sharing between different transitions for a signal as in ESCRCM. Except for gate sharing, the Monotonous Covering requirement and our three requirements for cubes are essentially the same. However, our requirements are defined with the specific intention of capturing the required pulse waveform for synthesis at STG level, rather than on the state graph as in [7]. We will further propose a necessary and sufficient condition for an STG to have ESCRCM implementation. Another work [1] allows multi-level logic implementation synthesized at the state graph. However, the sufficient and necessary realization condition is lacking in their approach.

The desirable property of ESCRCM is shown below.

Lemma 1[5]: The ESCRCM implementation of a signal is speed independent (i.e., hazard-free under the unbounded gate-delay model)[5].

4 Characteristic Graph

Our circuit synthesis of an STG is transformed onto the new *Characteristic Graph (CG)*, without restoring to the exponential-size state graph. The CG encapsulates the whole solution space for ESCRCM, whose size is linearly proportional to the number of transition pairs in the original STG. To implement a signal with ESCRCM, it is rather straightforward to derive an enabling cube for a transition, but it is not trivial to find such a cube which can turn off at the correct time and remains off for a proper period. On the CG, we can efficiently verify and identify such a cube for ESCRCM. It will be shown that a strongly connected subgraph in CG identifies a permissible cube for ESCRCM. In this section, after introducing the CG, we will describe how the ESCRCM requirements are transformed into graph properties in the CG, which can then be easily verified.

The following definition shows the construction principles of a CG. Note that although a CG is defined only for an STG/MG, based on the MG-decomposition in the previous section it can be applied for STG/FC in synthesis.

Definition 5 (Characteristic Graph (CG)): The CG of a live STG/MG with initial marking m_0 is constructed by the following steps:
I1. From m_0, unfold the STG/MG into an acyclic graph ASTG by traversing STG/MG twice.
2. Derive precedence relations for any two transitions in the ASTG.
3. For each signal f with k transitions in the ASTG, there are $k+1$ vertices in the CG: $k-1$ vertices are labeled with $k-1$ pairs of two successive transitions of f, another vertex is labeled with a pseudo transition preceding all transitions in the ASTG and the first transition of f reached from m_0, and one last vertex is labeled with the last transition of f and a pseudo transition succeeding all the transitions in the ASTG. The pseudo transition P_{f*} represents an inverse transition of the first (last) transition.
4. A directed arc exists between two vertices in the CG, $v_1 \rightarrow v_2$, if the first transition in v_1 is always fired before the second transition in v_2 from m_0.

Basically, our first two steps in creating a CG are as follows [6]. After all the precedence relations are derived, through a table look-up mechanism, we can construct a CG in $O(N^2)$ time. Let x_i+/*p* be a transition in the original STG/MG. We denote its k-th occurrence in the unfolding as ${}^k x_i$+/*p*. An example of unfolding the STG in Fig. 1(a) is shown in Fig. 1(c). Its resultant CG is shown in Fig. 1(d). For example, the arc $({}^2x+, {}^2x-) \rightarrow ({}^1y+, {}^2y-)$ means that ${}^2x+$ is always fired before ${}^2y-$ from m_0. For conciseness, some trivial arcs are not shown since the relations to vertices representing the transitions of the same signal have the transitive property, e.g. $({}^1x+, {}^1x-) \rightarrow ({}^1y-, {}^1y+)$ implies $({}^1x+, {}^1x-) \rightarrow ({}^1y+, {}^2y-)$, $({}^1x+, {}^1x-) \rightarrow ({}^2y-, {}^2y+)$ and $({}^1x+, {}^1x-) \rightarrow ({}^2y+, P_y-)$.

To derive and verify the ESCRCM circuit for some signal in an STG/MG, we will need two induced subgraphs of the CG, where a subgraph $G'(V', E')$ of a graph $G(V, E)$ is said to be *induced* by vertex set V', $V' \subset V$, if E' contains only those arcs in E whose terminals all are in V'. In the following, we will define the construction rules of such subgraphs.

Definition 6-1:
Given a nonempty transition set T of signal f in a live STG/MG G and a cube $x_1...x_n$,
Cg($G, x_1x_2...x_n, T$) is a subgraph of CG of G induced by the vertex set V' derived as follows:
(Let ^{1}*f+/p be the first enabled transition in T when unfolding G from initial marking.)*

(1) If $x_1x_2...x_n$ is not an enabling cube for T, $V' = \emptyset$ (i.e. Cg is empty).
(2) Otherwise, V' is the vertex set in CG which contains those and only those vertices labeled with transition pairs (x_i-, x_i+) occurring between ^{1}*f+/p* and ^{2}*f+/p*, i = 1,...,n.
(If x_i *is* $\overline{f}$, x_i- *is considered as falling between* ^{1}*f+/p and* ^{2}*f+/p since its effect on f is a result of the firing of* ^{1}*f+/p.)*

Definition 6-2: Given a cube $x_1x_2...x_n$ and a live STG/MG G, *Cg*($G, x_1x_2...x_n, \emptyset$) is a subgraph of CG of G induced by the vertex set V', which contains those and only those vertices labeled with transition pairs (x_i-, x_i+), $i = 1, ..., n$, where $x_i - (x_i+)$ can be a pseudo transition.

Let us examine the *Cg*($G, \overline{x}z\overline{y}, z-$) for STG G in Fig. 1(a). The transition pairs of $\{\overline{x},\ z,\ \overline{y}\}$ occurring between ${}^1z-$ and ${}^2z-$ are $\{({}^2x+, {}^2x-), ({}^1y+, {}^2y-), ({}^1z-, {}^2z+)\}$. The subgraph induced by the vertex set labeled by these transition pairs is strongly connected. This also implies that cube $\overline{x}z\overline{y}$ satisfies the ESCRCM requirements for $z-$, as shown in the next subsection. An example for Definition 6-2, *Cg*($G, \overline{x}yz, \emptyset$), has the following vertex set: $\{({}^1x+, {}^1x-), ({}^2x+, {}^2x-), ({}^1y-, {}^1y+), ({}^2y-, {}^2y+), (P_z-, {}^1z+), ({}^1z-, {}^2z+), ({}^2z-, P_z+)\}$. By definition, since the first transition of z after m_0 is an $z+$ (i.e, z is 0 in m_0), the vertices with pseudo transitions will be included. The subgraph induced by this set is also strongly connected. In the next subsection, we will show that the connectness implies that $\overline{x}yz$ is always off in the STG. Note that in Definition 6-1, if signal x_i is m-cycles in the STG, Cg contains m vertices labeled with *(*x_i*−/j,*x_i*+/j), j= 1,..., m*. Two Cg examples derived from the STG with multi-cycle signals in Fig. 2(a) are shown in Fig. 2(b) and (c), where transition ${}^k x*$ is simplified as $x*$ since ${}^1x*$ and ${}^2x*$ do not occur simultaneously in a Cg. The two examples will be used to illustrate circuit realization.

The relationship between the above subgraphs and the requirements of ESCRCM will be established in the following two theorems.

Theorem 1[5]: Let f be a signal in a live STG/MG G and T be a set of transitions of f. A cube $x_1x_2...x_n$(AND_T) satisfies all the requirements of ESCRCM for T if and only if: (a) for each transition $t \in T$, AND_T contains at least one literal x_i with $t \Rightarrow x_i- \Rightarrow \overline{t}$, where $\overline{t}$ is the nearest transition of signal f succeeding t; (b) for each transition $t \in T$, the vertices of nonempty *Cg*(G, AND_T, T) labeled with the transition set of signals $x_1, ..., x_n$ occurring between t and the nearest transition $\in T$ succeeding t are strongly connected.

In this lemma, if x_i is $f(\overline{f})$ itself, its firing $x_i - (x_i+)$ is considered as coming after $f - (f+)$ since its effect on f is a result of the firing of $f - (f+)$.

Let us check whether $a\overline{b}c\overline{d}e$ satisfies the ESCRCM requirements for *a-/2* in Fig. 2(a). This cube contains *e* and *a-/2* $\Rightarrow e- \Rightarrow$ *a+/1*, so condition (a) is satisfied. The check of condition (b) is *Cg*($G, a\overline{b}c\overline{d}e$, *a-/2)* as shown in Fig. 2(b). It is strongly connected, so condition (b) is also satisfied. Both conditions are satisfied, so $a\overline{b}c\overline{d}e$ can be used to activate *a-/2* in the ESCRCM implementation of *a*. Another example is to check $\overline{a}\overline{b}\overline{e}$ for {*a+/1,a+/2*}. Similarly, the first condition is satisfied. Condition (b) is *Cg*($G, \overline{a}\overline{b}\overline{e}$, {*a+/1,a+/2*}*)* as shown in Fig. 2(c). We have two vertex subsets: {*(a+/1,a-/1), (b+,b-)*} and {*(a+/2,a-/2), (e+,e-)*}. Both are strongly connected, so condition (b) is also satisfied. Hence, $\overline{a}\overline{b}\overline{e}$ can be used to activate *a+/1* and *a+/2* in the ESCRCM implementation of *a*.

While exploring for the optimal circuit realization, AND_T can be expanded as large as possible while continuing to satisfy these two conditions satisfied. Here, we will show another application of CG to check whether a cube is always off or always on in an STG/MG. Such checks are required when applying MG-decomposition for STG/FCs.

Theorem 2[5]: Given a cube $x_1x_2...x_n$ for a live STG/MG G, (1) it is always on in G iff all signals $x_1, x_2, ..., x_n$, stay 1-stable in G; (2) it is always off in G iff at least one signal of $x_1, x_2, ..., x_n$, stays 0-stable in G or Cg($G, x_1x_2...x_n, \emptyset$) is not empty and is strongly connected.

In the STG G of Fig 1(a), $\overline{x}yz$ is always off. The $Cg(G, \overline{x}yz, \emptyset)$ is strongly connected. However, for $\overline{x}\ \overline{y}\ \overline{z}$, the $Cg(STG, \overline{x}\ \overline{y}\ \overline{z}, \emptyset)$ consisting of $\{({}^1x+, {}^1x-), ({}^2x+, {}^2x-), (P_y+, {}^1y-), ({}^1y+, {}^2y-), ({}^2y+, P_y-), ({}^1z+, {}^1z-), ({}^2z+, {}^2z-)\}$ is not strongly connected, and the cube will be on in state 000.

5 A Necessary and Sufficient Realization Condition

The necessary and sufficient condition is based on the canonical implementation, which is unique for a signal in the given STG.

Definition 7 (Canonical implementation): The canonical implementation for a noninput signal f in a live STG has the circuit configuration of ESCRCM while AND gates are constructed according to the following steps with set and reset subcircuits derived separately:
(1) Use the smallest enabling cube for each $f*$ as AND_{f*} gate.
(2) If the intersection of any two cubes covers a valid state, then the smallest cube covering them is used to activate their associated transitions, and the original cubes are removed.
(3) Repeat step 2 until no such intersection exists.

Let us examine the canonical implementation of signal a in Fig. 2(a). The two smallest enabling cubes for *a+/1* and *a+/2* are $\overline{a}\overline{b}\ \overline{d}\overline{e}$ and $\overline{a}\overline{b}\overline{c}\ \overline{e}$. These two cubes intersect in a valid state whose code is $\overline{a}\overline{b}\overline{c}\overline{d}\overline{e}$, so the smallest cube covering them, $\overline{a}\overline{b}\overline{e}$, is used in the set subcircuit. For the reset subcircuits, we get $a\overline{b}\overline{c}\ \overline{e}$ for *a-/1* and $a\overline{b}c\overline{d}e$ for *a-/2*. The complete canonical implementation is shown in Fig. 3(b). The canonical implementation in general is not an optimal implementation. Furthermore, it may not satisfy all the ESCRCM requirements. However, it can serve as a necessary and sifficient condition as established in the following theorem.

Theorem 3[5]: Signal f in a live STG/FC G has hazard-free implementation based on ESCRCM if and only if its canonical implementation is an ESCRCM implementation.

The theorem has two major implications. First, if the canonical implementation meets the requirements of ESCRCM, it provides the least area upper-bound for all feasible ESCRCM implementations. Another possible solution which uses the largest enabling cubes as AND_T gives the area *lower-bound*; however, it may not be hazard-free. Both can serve as a good starting point for exploring for the exactly optimal solution[5]. Secondly, if a signal in the original STG cannot be implemented with ESCRCM, then we only need to rectify the STG to have its canonical implementation to meet the requirements of ESCRCM. We have a preliminary result on the rectification strategy. The key idea is that the canonical implementation of a noninput signal in a live STG/FC with CSC property always meets the requirements of ESCRCM if each of its transitions has no concurrent transition. Given a live STG with CSC property, insertion of arcs only (the reduction of states) is enough to rectify the STG. Such rectification is different from [13], which only uses the signal insertion. The rectification result will be included in our future work.

6 Circuit Realization and Results

Our preliminary result has proved that the exact gate-minimization and literal-minimization of speed independent circuits based on the ESCRCM both are NP-hard[5]. In this section, a heuristic method for circuit realization will be presented which was evaluated on a SUN SPARC2 station with STG examples satisfying the CSC property collected in *sis* [3].

The optimization starts with the canonical implementation, which can be derived with the CG. If it is not an ESCRCM, then the STGs need rectification. All the examples in Table 1 except for trimos-send.g have canonical ESCRCM implementations without any rectification on the original STGs. STG trimos-send.g is not persistent[2], so it is rectified with three additional arcs, following the strategy in [8]. Then, we try to remove the C-element. The removal based on that, if and only if all ANDs in the set (reset) part are able to completely cover on-set (off-set), can the C-element and the reset (set) combinational part be directly removed simultaneously, and the remaining set (reset) combinational part still retain functional correctness and hazard-freeness. After the possible removal, we try to merge gates of different transitions of the same signal. Since the number of rising(falling) transitions for a signal in the STG generally is small, exhaustive search is adopted. Literal minimization is then carried out. The main concern in this step is that literal deletion does not destroy the connectness of certain vertices in Cgs. Those certain vertices include those enabling signals and one of signals which can ensure that the selected cube always turns off before next inverse transition of the enabled one. We use a greedy strategy to select those literals to be deleted. For connectness, we choose to delete literals whose corresponding vertices have minimal degrees, i.e. less contribution for connectness. Here we give an example. We try to optimize the cube for *a-/2* in the STG G in Fig. 2(a). The canonical implementation for *a-/2* is $a\overline{b}c\overline{d}e$. The corresponding sub-CG is shown in Fig. 2(b), where the vertices *(c-,c+)* and *(e-,e+)* must be included. We subsequently delete vertices *(a-/1,a+/2)*, *(b-,b+)*, $(c-, c+)$ and *(a-/2,a+/1)*, and find that the remaining subgraph is still strongly connected. Hence, the optimal solution for *a-/2* is ce. The literal minimization step is the most critical setp of the overall procedure in terms of time complexity. Since the check of connectness in a Cg needs $O(N^2)$, this particular step needs $O(L \times M \times N^2)$, where L is the literal number of canonical implementation, M is the number of MG-components, and N is the transition number of the largest MG. M is crucial for an STG with more than one MG-component. From a practical point of view, for the STG/FC class in which no transition is concurrent with any free-choice transition, M is always less than N[5]. Most of the examples in the literature belong to this class.

The heuristic algorithm has been written in C and successfully evaluated on a SPARC2 station with the set of STG benchmarks (without CSC violation) from *sis*. Table 1 shows the evaluated results, where $T(S)$ means the number of transitions (states). We compare our synthesis result with that of version 3.0 SYN[1], which derives speed-independent circuits also based on a predefined circuit template in the domain of state graph. The *literal count* in the combinational part is used as an area criterion because those evaluated examples are all two-level. The final synthesis result from our heuristic algorithm is shown in column *Ours*. Compared with SYN, we have slightly better result. As the results obtained by SYN already have approached the exact optimization, the improvement on area naturally is not very significant.

The significant improvement over SYN is in CPU time. Our realization is 1 to 3 orders of magnitude faster for most cases with more than 50 states. The individual speedup ratio depends on the

Circuits	T	S	Literals (Memories)		CPU time (s)	
			SYN	Ours	SYN	Ours
chu133.g	14	24	12(2)	13(2)	0.55	0.100
full.g	8	16	8(2)	8(2)	0.22	0.050
hybridf.g	16	80	14(3)	14(3)	1.78	0.100
vbe10b.g	22	256	32(7)	32(7)	14.28	0.283
vbe5b.g	12	24	11(2))	11(2)	0.46	0.067
vbe5c.g	12	24	8(3)	10(3)	0.42	0.067
chu172.g	13	12	7(1)	7(1)	0.21	0.117
chu150.g	14	26	11(1)	11(1)	0.51	0.067
converta.g	14	18	20(3)	20(3)	0.71	0.100
ebergen.g	14	18	14(2)	14(2)	0.5	0.117
hazard.g	10	12	10(2)	10(2)	0.21	0.050
qr42.g	14	18	14(2)	14(2)	0.45	0.117
nowick.g	16	20	16(3)	18(2)	0.82	0.117
wrdatab.g	24	216	34(5)	34(5)	15.55	0.283
pe-send-ifc.g	53	108	73(5)	68(4)	26.87	4.983
pe-rcv-ifc.g	50	53	67(6)	59(6)	6.48	4.117
nak-pa.g	20	58	20(4)	20(4)	1.95	0.200
chu-ffo.g	16	64	12(2)	12(2)	2.86	0.083
sbuf-ram-write.g	24	64	20(3)	19(2)	2.42	0.317
trimos_send2.g	18	84	27(6)	27(6)	3.44	0.283
master-read.g	28	2108	33(7)	34(7)	1324.1	0.317
Total	-	-	463(71)	455(68)	-	-

Table 1: Experimental Results(I)

characteristics of the STG. Generally, the CPU time of SYN is quite sensitive to the state number while ours depends mainly on the transition number and conditional structure, which determines the number of MG-components. For further comparison, a set of larger-size STG/MGs derived from the multiple-input blocks (pipelined and nonpipelined) in [8] is evaluated. The original block in [8] has 2 inputs. We extend the input (output) number to 4, 5 and 6. As shown in Table 2, our work has significantly reduced the CPU time from hours to seconds with similar results. Note also that our algorithm has time polynomially proportional to the number of transitions, but SYN takes exponentially increasing time.

7 Conclusion

We have presented a new synthesis approach for realizing hazard-free circuits under the speed independence model for Signal Transition Graphs (STGs) with a practical configuration. In our approach, the synthesis problem can be completely transformed into the new STG-level model, CG, which encapsulates the solution space of hazard-free realization based on a predefined but general circuit model. The CG has size complexity comparable to that of the original STG and significantly less than that of the corresponding state graph. It enables us to explore the realizability as well as the optimization of hazard-free realization. Furthermore, we have derived a necessary and sufficient condition for hazard-free realization. Previously, only sufficient conditions have been reported. With CGs and this condition, we have developed a heuristic synthesis algorithm which results in solutions comparable to the state-graph approach while requiring very little CPU time.

Circuits	T	S	Literals (Memories)		CPU time (s)	
			SYN	Ours	SYN	Ours
mi2.g	12	64	14(3)	14(3)	1.49	0.08
mi4.g	20	1024	26(5)	26(5)	184.6	0.18
mi5.g	24	4096	32(6)	32(6)	4022	0.28
mi6.g	28	16385	+	38(7)	>25000	0.43
mi2np.g	12	42	13(3)	15(3)	1.38	0.07
mi4np.g	20	594	25(5)	29(5)	93.1	0.16
mi5np.g	24	2292	30(6)	36(6)	1534.3	0.33
mi6np.g	28	8922	+	43(7)	>13000	0.44

+: Unable to synthesize with the given CPU time limit with 48MB main memory.

Table 2: Experimental Results(II)

References

[1] P. A. Beerel and T. H.-Y Meng, "Automatic Gate-Level Synthesis of Speed-Independent Circuits," In *Proc. ICCAD*, pp. 581-586, 1992.

[2] T. A. Chu, "Synthesis of Self-Timed Control Circuits from Graphical Specifications", PhD thesis, MIT, June, 1987.

[3] L. Lavagno, K. Keutzer, A. Sangiovanni Vincentelli, "Algorithms for Synthesis of Hazard-free Asynchronous Circuits," In *Proc. 28th DAC*, pp. 302-308, 1991.

[4] K. J. Lin, J. W. Kuo and C. S. Lin, "Direct Synthesis of Asynchronous Hazard-Free Circuits Based on Lock Relation and MG-Decomposition from STGs," In *Proc. European Conference on Design Automation*, pp. 178-183, 1994.

[5] K, J. Lin, "Synthesis of Speed-Independent Circuits from Signal Transition Graphs," PhD thesis, National Taiwan University, R.O.C., 1996.

[6] M. A. Kishinevsky, A. Y. Kondratyev and A. R. Taubin, "Specification and Analysis of Self-Timed Circuits," *Journal of VLSI Signal processing*, vol. 7, pp. 117-135, 1994.

[7] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen and A. Yakovlev. "Basic Gate Implementation of Speed-Independent Circuits," In *Proc. 31th DAC*, pp. 56-62, 1994.

[8] T. H. Meng, *Synchronization Design for Digital Systems*, Kluwer Academic, 1990.

[9] C. W. Moon, P. R. Stephan and R. K. Brayton, "Synthesis of Hazard-free Asynchronous Circuits from Graphical Specifications," In *Proc. ICCAD*, pp. 322-325, 1991.

[10] E. Paster and J. Cortadella, "Polynomial Algorithm for the Synthesis of Hazard-free Circuits from STGs," In *Proc. ICCAD*, pp. 250-254, 1993.

[11] I. E. Sutherland, "Micropipelines," *Commun. ACM*, vol 32, no 6, pp. 720-738, 1989.

[12] P. Vanbekbergen, F. Catthoor,

[13] K. Van Berkel, "Beware the Isochronic Fork," *Intergration VLSI Journal*, vol. 13(2), pp. 103-128, 1992.

[14] K. Van Berkel, R. Burgess and J. Kessels, A. Peeters, M. Roncken and F. Schalij, "A Fully-Asynchronous Low-Power Error Corrector for the DCC Player," *IEEE Journal of Solid-State Circuits*, vol. 29, pp. 1429-1439, 1994.

[15] T. E. Williams, and M. A. Horowitz, "A Zero-Overhead Self-Timed 160ns 54b CMOS Divider," *IEEE Journal of Solid-State Circuits*, pp. 1651-1661, Nov. 1991.

[16] C. Ykman-Couvreur, B. Lin, G. Goossens and H. D. Man, "Synthesis and Optimization of Asynchronous Controllers Based on Extended Lock Graph Theory," In *Proc. European Conference on Design Automation*, pp. 512-517, 1993.