An Optimal Approach to Implementing Self-timed Logic Circuits from Signal Transition Graphs[†]

Edwin C.Y. CHUNG[‡] and Lindsay KLEEMAN

Department of Electrical and Computer Systems Engineering
Monash University
Clayton, Victoria 3168, Australia
Fax: +61 3 565 3454

Abstract—Scaling of integrated circuits in recent years has resulted in improvements in speed and density of VLSI circuits. However scaling has also aggravated clock skew problems due to increasing wire delays. Consequently, to exploit the speed improvements of scaling and to avoid synchronisation failure in synchronous systems, designers are now turning towards self-timed system design for solutions. Amongst the techniques for the synthesis of self-timed circuits, an approach using *Signal Transition Graphs* was introduced by Chu [1, 2, 3]. In an attempt to simplify his method and to achieve efficient results, he uses a procedure called *net contraction* to decompose a Signal Transition Graph into simpler subgraphs known as contracted STGs. With the possibility of state assignment problems in the contracted STGs, net contraction introduces complications and inefficiencies. The cause of these deficiencies can be shown to be the criterion upon which a signal is retained during net contraction. To avoid these deficiencies, a new approach is presented that derives circuit implementations from uncontracted state graphs using Quine-McCluskey tabular Karnaugh mapping and Prime Implicant tables. This approach is shown to produce hazard free implementations with a minimum number of gates in contrast to Chu's sub-optimal method.

Key words—Self-timed VLSI circuit, net contraction, Signal Transition Graphs, state graph, Quine-McCluskey Karnaugh mapping, prime implicant tables.

1. Introduction.

Advancements in VLSI technology over the past decade have resulted in a significant scaling down of feature sizes of integrated circuits. An effect of this is the lowering of the *transit time* of transistors, giving us faster devices. However, since scaling entails

[†] The work described in this paper is supported in part by the Australian Telecommunication and Electronics Research Board.
[‡] Holds a scholarship from the Australian Postgraduate Research Award

an increase in wire delays relative to transit time [4], scaling also aggravates clock skew as designs require faster and larger circuits. Synchronous systems with asynchronous inputs are also inherently susceptible to *synchronisation failure* due to the *metastable failure* of storage devices [5, 6, 7]. This further limits the maximum speed at which a synchronous circuit can be operated reliably. To circumvent these timing difficulties and to exploit the superior speed of scaled devices, designers are now turning towards self-timed system design methodology.

In contrast to synchronous systems, temporal control over the activities of a self-timed system is not centralised by a global clock, but is instead distributed throughout the individual self-timed elements making up the system. Without a global clock, the worsening of clock skew with scaling will not be a limiting factor to a self-timed system, nor will there be a need to synchronise asynchronous inputs to a global clock, avoiding synchronisation failure in these systems. However, additional circuitry is necessary to support localised communication amongst the self-timed circuits to achieve any required global synchronisation.

Currently, a number of techniques for the synthesis of self-timed circuits are available. These range from the use of predefined circuit configurations [8, 9, 10, 11] to the use of *Trace Theory* [12], *Petri Nets* [13, 14], *Signal Transition Graphs* [3], compilation of commands [15] and combinations of the above [16]. In techniques which use predefined circuit configurations, remnants of synchronous circuits frequently remain, but with a distributed internal clocking arrangement of some sort. In the other techniques however, an asynchronous state machine is frequently derived from a description of circuit behaviour specified in terms of the sequences of signal transitions of the circuit. For many logic designers, text based circuit description, such as commands [15] and traces [12], are not as illustrative as graphical notations, such as Signal Transition Graphs [3] and a graphical form of traces [17].

Based on different ideas and emphases, the above techniques approach the implementation of self-timed circuits very differently. In this paper, the technique for the synthesis of self-timed circuits from Signal Transition Graphs as introduced by CHU [3] is examined. Attempting to simplify the circuit synthesis procedure and to achieve efficient results, a graph-theoretic operation called *net contraction* is used to decompose a Signal Transition Graph (STG) into simpler descriptions known as *contracted STGs*. An implementation of the circuit is then derived from these simplified descriptions using *state graphs* and Karnaugh maps.

In this paper, the technique of net contraction is shown to be inefficient and unnecessarily complex. A new approach using Quine-McCluskey tabular Karnaugh mapping and Prime Implicant tables is presented. This method can be completely computer automated in contrast to Chu's technique which may require intervention. The new approach is also shown to produce minimum hazard free implementations at least as efficient and sometimes more efficient that Chu's method in terms of number of gates.

The paper is organised as follows: In section 2 preliminary topics related to STGs are introduced, and in section 3 the technique of net contraction is described and its limitations discussed. An elegant solution to these limitations is presented in section 4 and an example is presented. The example not only clarifies the new approach but also shows that the Chu's method is sub-optimal in terms of number of gates. The new approach is justified more formally in section 5, and conclusions presented in section 6.

2. Preliminaries

A brief, informal introduction to the basic concepts of STGs and Chu's technique of implementing STGs is presented in this section.

2.1. Signal Transition Graphs

The specification of the behaviour of a self-timed circuit can be performed with a *Signal Transition Graph*. An example of an STG is the description of a Müller C-element in Fig. 1b. Note that in the STG description, the name of a signal is suffixed with a '+' or a '-' to denote a rising or a falling transition of the signal. To facilitate the implementation, transitions of input signals are underlined to differentiate them from non-input (internal and output) signals.

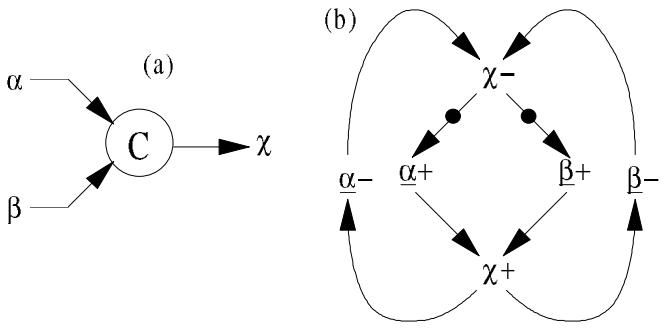


Figure 1: (a) A block diagram of a Müller C-element and (b) its STG description.

In addition to the interconnection of signal transitions, an STG description is also marked with a set of dots known as *tokens* (as in Petri nets). The tokens on the STG are interpreted by continually firing the enabled transitions as in the following rules:

> A signal transition is enabled (i.e., can occur) when all its input arcs are marked with tokens. The firing (or occurrence) of an enabled transition will then remove a token from each of its input arcs while adding a token to each of its output arcs.

As an example, consider the STG description in Fig. 1b. Note that, with the tokens at the output arcs of χ-, both transitions α+ and β+ are enabled. The firing of these transitions will each independently shift the token from their input arc to their output arc as illustrated in Fig. 2. After the firing of both α+ and β+, the tokens are now at the input arcs of χ+ as shown in Fig. 2d. This indicates that transition χ+ is enabled and the firing of χ+ will shift the tokens to its output arcs enabling both transitions α- and β-. Similar to the firing of α+ and β+, the firing of α- and β- will eventually shift the tokens to the input arcs of χ-. Firing χ- will then shift the tokens to its output arcs completing a circle of execution.

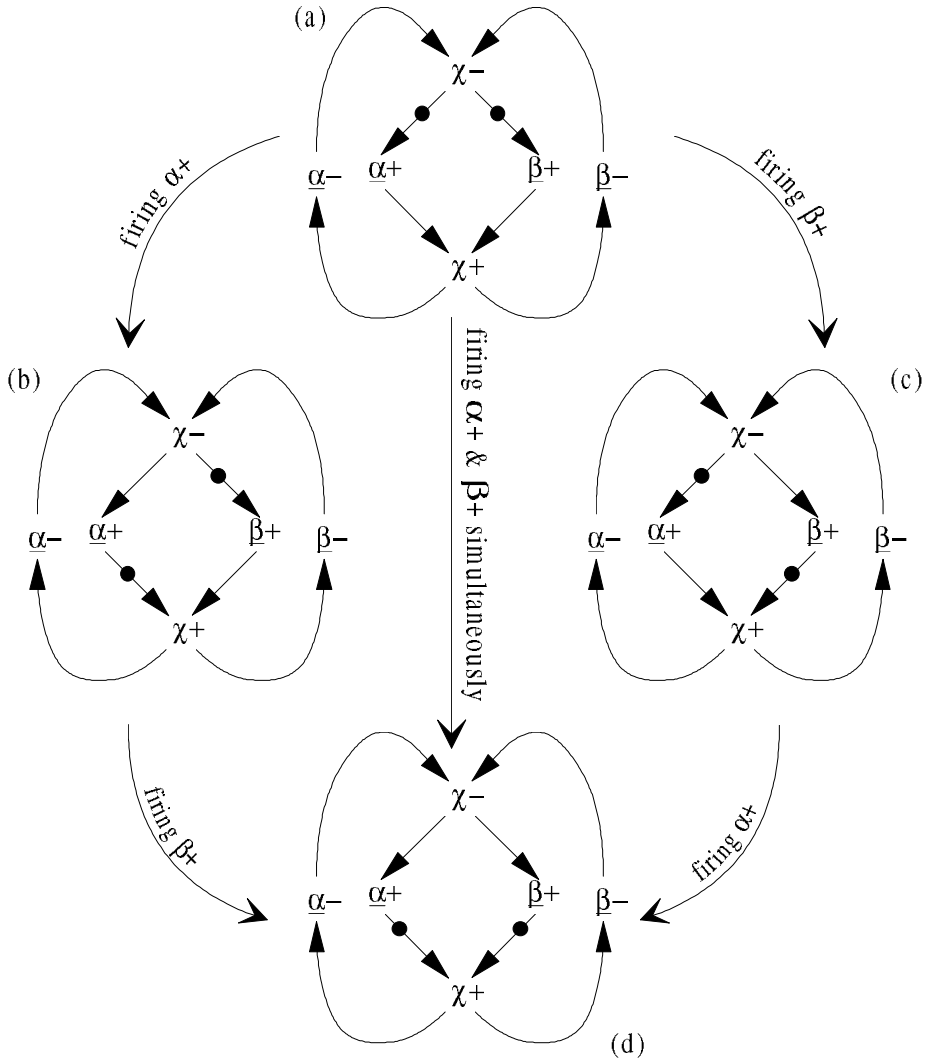


Figure 2. The occurrence of α+ followed by β+ will result in the sequence (a) (b) (d), while the occurrence of β+ followed by α+ will result in the sequence (a) (c) (d).

In addition, Signal Transition Graphs can also specify signal transition sequences dependent upon the state of input or non-input signals. However, since this is not relevant here, interested readers should consult [3] for further details.

2.2. State Graphs

Though flexible and expressive in describing circuit behaviour, STGs are not in a form where circuit implementation is directly extractable. To derive logic expressions for output and internal signals from an STG, a *state graph* description is required.

To illustrate the concept of a state graph and its relation to STGs, consider the token markings associated with the repetitive firing of enabled transitions described above. Besides indicating the enabled transitions at the different stages of execution, these markings also define the state of the circuit. For example, consider Fig. 1b. Note that besides indicating transitions $\alpha+$ and $\beta+$ are enabled, the token marking also indicates that transition $\chi-$ has occurred, representing a state where $\alpha = \beta = \chi = 0$. The firing of $\alpha+$ while in this state will shift the token to a marking as shown in Fig. 2b, representing a state where $\alpha = 1$ while β and χ remains at 0.

The sequences of token markings associated with the repetitive firing of enabled transitions can be represented graphically in a state graph as shown in Fig. 3. Note that the state of the circuit is denoted by a binary vector $\langle \alpha\ \beta\ \chi \rangle$ and the change in token marking caused by the firing of a transition is denoted by a directed arc labelled with the transition.

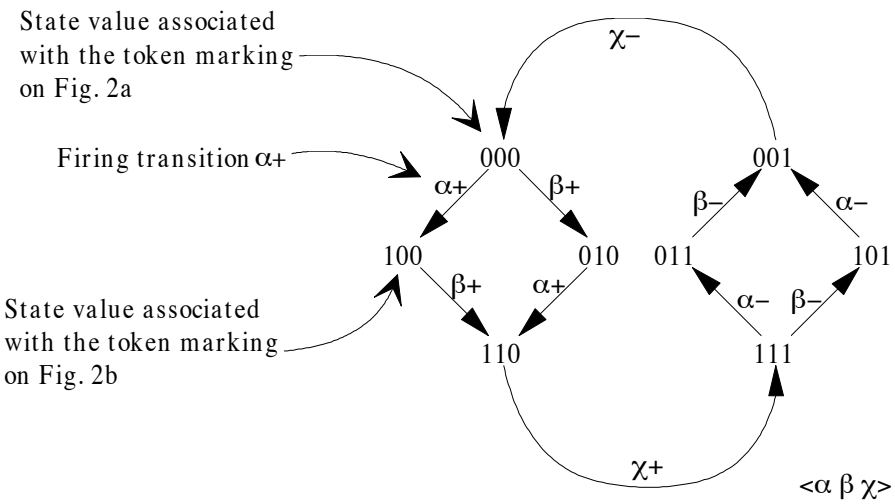


Figure 3. The state graph description of the Müller C-element.

2.3. Mapping a State Graph to a Karnaugh Map

Given a state graph description, each of the states can be taken to correspond to a cell in a Karnaugh map (K-map). The K-map for a non-input (internal or output) signal is then obtained by filling each of the cells of the K-map with the *implied value* of the non-input signal in the state graph. The implied value of a signal j in state s, denoted by $f(s, j)$, is defined by Chu as:

$$f(s,j) = \begin{cases} s'(j) & \text{if there exists a state } s' \text{ such that } s[j^*\rangle s' \\ s(j) & \text{otherwise} \end{cases}$$

Where $s(j)$ denotes the logical state of signal j in state s and

$s[j^*\rangle s'$ denotes the existence of the path $s \xrightarrow{j^*} s'$ in state graph description.

As an example, the K-map for output χ in the state graph of Fig. 3 is shown in Fig. 4.

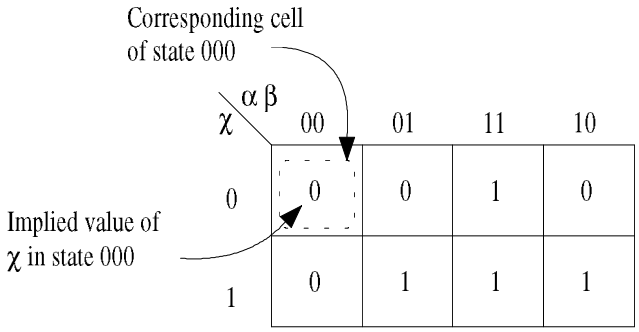
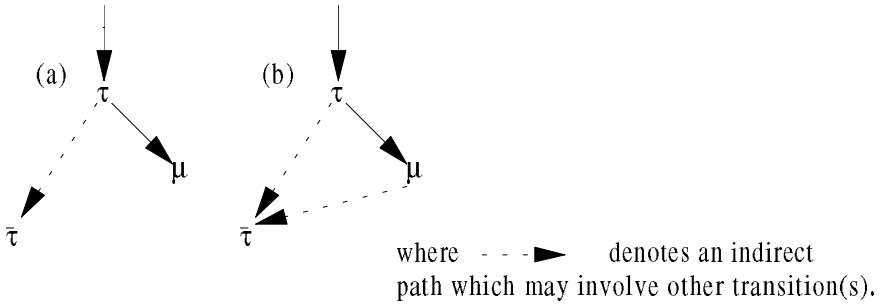


Figure 4. Karnaugh map for output χ of the Müller C-element.

2.4. Persistency and the State Assignment Problem

To ensure that a STG can be implemented without hazards, the STG needs to be checked for *persistency* and for *state assignment problems*.

Persistency[1]**:** [1 p568, 2 p221]
An STG is *non-persistent* if there exist transitions τ, $\bar{\tau}$ (the reverse transition of τ) and μ, such that τ enables μ while $\bar{\tau}$ is concurrent with μ, as shown in Fig. 5a. To eliminate the non-persistency between τ, $\bar{\tau}$ and μ, it is only required to eliminate the concurrency between $\bar{\tau}$ and μ by inserting an additional constraint to force $\bar{\tau}$ and μ to be sequentially ordered as shown in Fig. 5b.



[1] In Meng's work [16], the term *semi-modular* (*semi-modularity*) is used instead of *persistent* (*persistency)*.

Figure 5: (a) Condition for non-persistent in STGs and (b) the addition of a constraint to ensure persistency.

As an example, consider the STG in Fig 6a. As bolded in the figure, the STG is non-persistent with *a*+ enabling *c*+ while *c*+ is also concurrent with *a*- (top portion Fig. 6a) and also *a*+ enabling *c*- while *c*- is concurrent with *a*- (lower portion Fig. 6a). To correct the non-persistency, additional constraints are required to force the sequential ordering between *c*+ and *a*- as well as *c*- and *a*- as bolded in Fig. 6b.

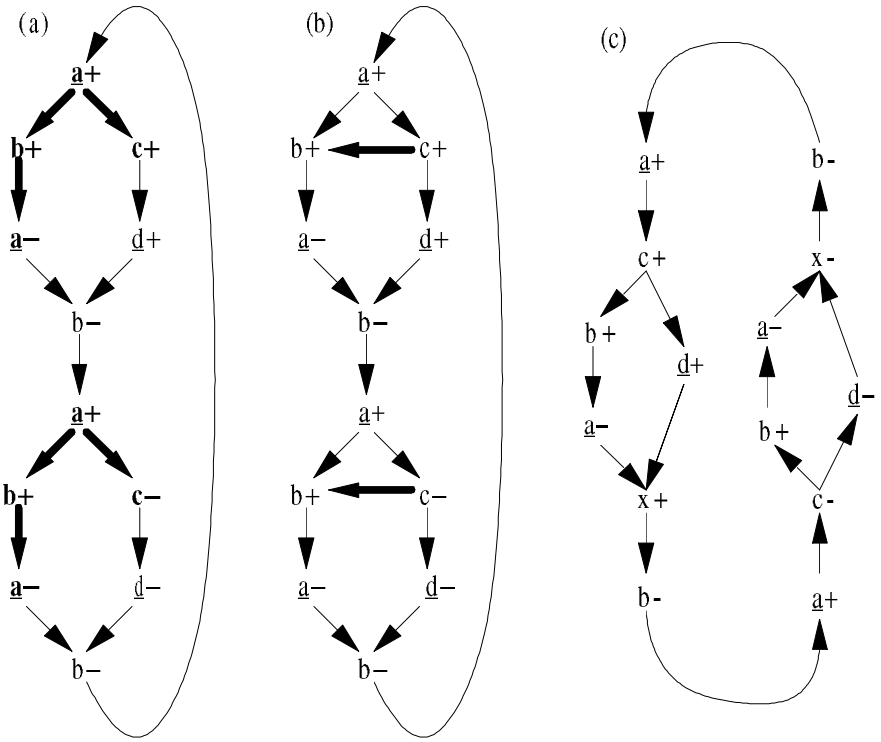


Figure 6: (a) Original STG description of QR42 (taken from [18]), (b) after the insertion of constraints to eliminate non-persistency and (c) after the elimination redundant constraints.

State assignment problem: [1 p567, 2 p222]

An STG must be also persistent in terms of its state assignment in order to achieve an implementation. An STG is *non-persistent due to state assignment* if its state graph description does not have unique state values. In other words, the state values associated with the token markings on the STG do not uniquely code the state of the STG. To differentiate between the two types of non-persistency, the term *state assignment problem* is used when referring to STGs or state graphs which are non-persistent due to their state assignment.

The cause of a state assignment problem is the presence of a *complementary set* in the STG. A complementary set is defined as a sequence of signal transitions with no net effect on the state of the circuit. As an example, consider the STG in Fig. 7a. Note that the complementary set {*Zr+*, *Zr-*} on the left of the STG translates into the sequences of state transitions [**110** → 111 → **110**] and [**100** → 101 → **100**] as in Fig. 7b, causing the state assignment problem.

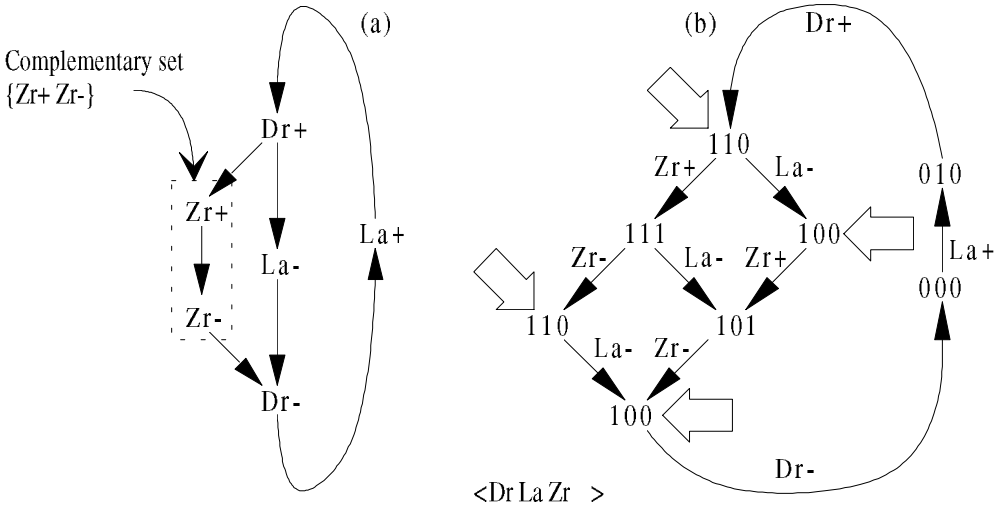


Figure 7: (a) An STG with state assignment problem and (b) its state graph.

To resolve a state assignment problem, all the complementary sets in the STG must be eliminated. This can be done by inserting additional internal signal transitions or in some cases, by rearranging the order of signal transitions in the STG. For example, to resolve the state assignment problem in the STG in Fig. 7a, a transition of internal signal *x* is inserted between transitions *Zr+* and *Zr-* to annul the complementary set and another between transitions *Dr+* and *La+* to preserve *liveness* as in Fig. 8a. An STG is said to be *live* if all its signal transitions appear in sequential pairs $\{\tau, \overline{\tau}\}$. STGs must be live to be realisable.

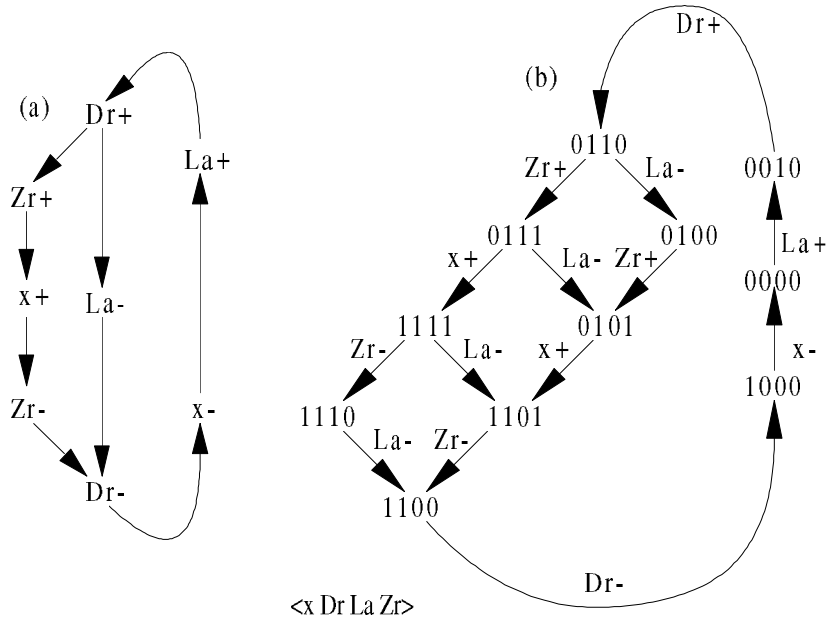


Figure 8: (a) The STG in Fig. 7a after the insertion of transition *x*+ and *x*-, and (b) its state assignment problem free state graph description.

3. Net Contraction

After having obtained a persistent and state assignment problem free STG, a circuit implementation is derived by Chu's method using a series of steps starting with *net contraction* [1 p567, 2 p222]. In net contraction, the STG is decomposed into simpler *contracted STGs*, from which an implementation of the circuit is derived. In this section, net contraction and its limitations are discussed.

3.1. Theory of Net Contraction

Instead of deriving the logic expression of a non-input signal, γ, from the full STG, Chu suggested it would be more efficient to work from the simplified STG_γ description containing only signals essential to the implementation of γ. Net contraction performs the simplification from STG to STG_γ.

Firstly, the set of signals essential for the implementation of γ is determined. Chu claims (incorrectly as we show below) that these are the set of signals whose transition can ***directly*** cause a change in γ [3 p92] and calls it the *input set* of γ and defines it as follows:

Definition of the Input Set

$$I(\gamma) = \{\alpha \in J \mid \alpha^* \Re \gamma^*\}$$

where α^* denotes either a rising of a falling transition of signal α,

J is the set of all the input, output and internal signals of the circuit.

$\alpha^* \Re \gamma^*$ denotes an arc $\alpha^* \rightarrow \gamma^*$ in the STG

Having selected all the essential signals required for the implementation of γ, signal transitions involving signals which are not an element of $I(\gamma) \cup \{\gamma\}$ are then eliminated from the STG description. As illustrated in Fig. 9, during the elimination of each of these signal transitions, input and output arcs are reorganised to preserved the temporal relations of the remaining signal transitions. Redundant constraints resulting from the elimination of these signal transitions, as illustrated in Fig. 10, are also removed before the final STG_γ is obtained.

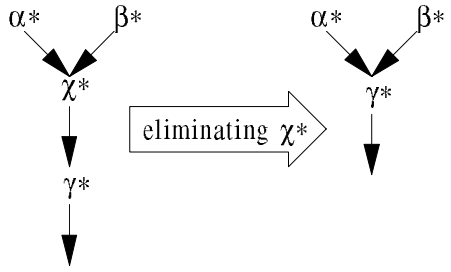


Figure 9. Elimination of signal transition χ^* from a portion of an STG description.

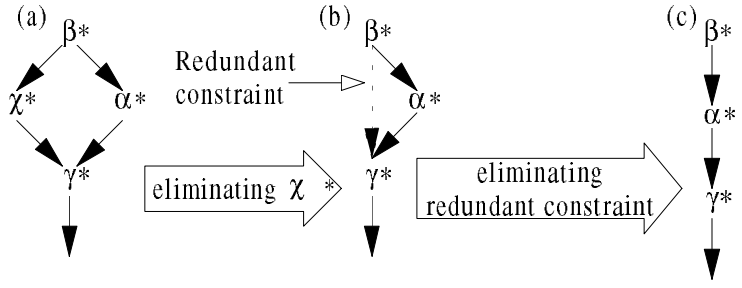


Figure 10: The elimination of χ^* from (a) results in a redundant constraint from β^* to γ^* in (b) and (c) the structure after the removal of the redundant constraint.

As an example, consider the STG description of a trigger module in Fig. 11a. To obtain the contracted STG for *Ia*, the input set of *Ia*, I(*Ia*), is first determined. As

bolded in Fig. 11b, changes in *Ia* are caused by changes in *Ir* and *Or*, as such I(*Ia*) = {*Ir*, *Or*}. Since *Oa* is the only signal that is not an element of I(*Ia*) ∪ {*Ia*}, only transitions *Oa*+ and *Oa*- will be eliminated giving a contracted STG as shown in Fig.11c. Redundant constraints in the STG are then removed to obtain STG_{Ia} as shown in Fig. 11d.

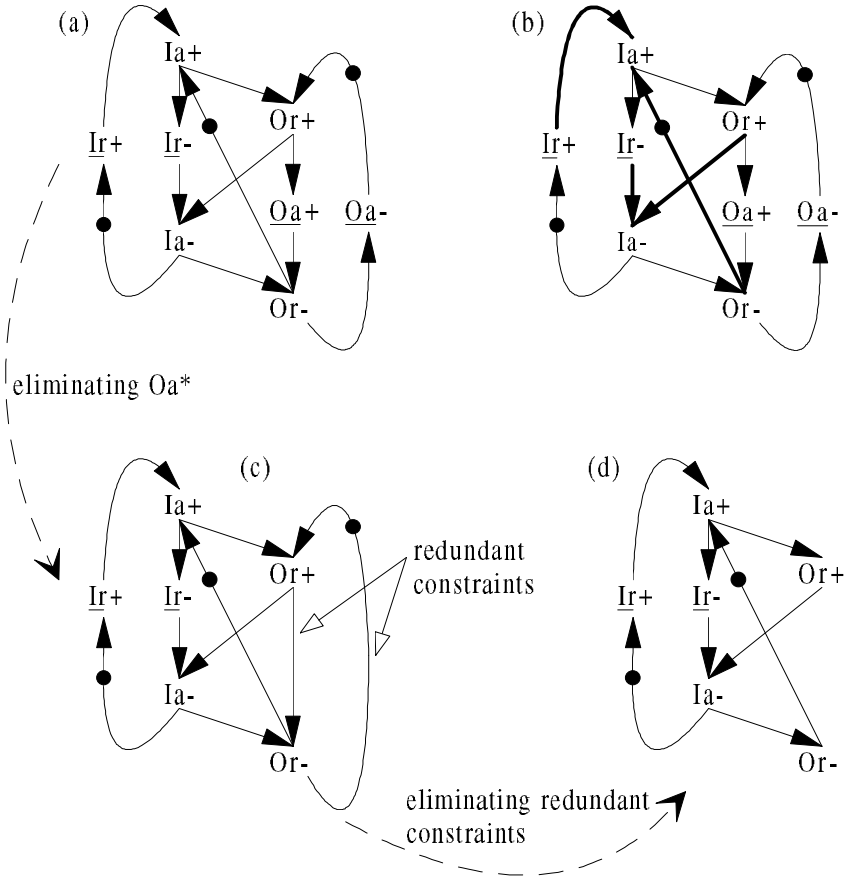


Figure 11: (a) An STG description of a trigger module [2, Fig. 3], (b) arcs which cause *Ia** highlighted, (c) after the elimination of *Oa*+ and *Oa*- and (d) the contracted STG for *Ia* after the removal of redundant constraints.

3.2. Limitations of Net Contraction.

In net contraction, since the input set is used to determine which signals remain in the contracted STG and which signals are eliminated, the contracted STG for a non-input signal γ will only contain signals that are causally related to γ and γ itself. At the state graph level, this means that there exist paths

$$\longrightarrow S_1 \xrightarrow{\alpha*} S_2 \xrightarrow{\gamma*} S_3 \longrightarrow \text{ for all } \alpha \in I(\gamma).$$

That is, there exist at least two states S_1 and S_2, where a change in α at state S_1 will bring the circuit to state S_2 subsequently enabling signal transition γ^*. Translated to the Karnaugh map, there exists at least two cells corresponding to states S_1 and S_2 across the α-boundary[2] where the implied values of γ differ. Therefore all signals $\alpha \in I(\gamma)$ are considered essential for the implementation of γ. However, because there is the possibility of having different implied values across a τ-boundary for $\tau \notin I(\gamma) \cup \{\gamma\}$, net contraction can also eliminate signals that are essential for the implementation of γ. As an example, consider the QR42 module discussed earlier in Fig. 6. From the STG in Fig. 12a, $I(c) = \{a\}$, but note however, the implied value of c at the neighbouring states, with vector $<a\ b\ c\ d\ x>$, 10110 and 10111 across the x-boundary are different even though $x \notin I(c)$. Note also, the elimination of x during net contraction will lead to a state assignment problem as shown in Fig. 12f.

[2] An α-boundary is a boundary in the Karnaugh map where α is 1 on one side and 0 on the other side.

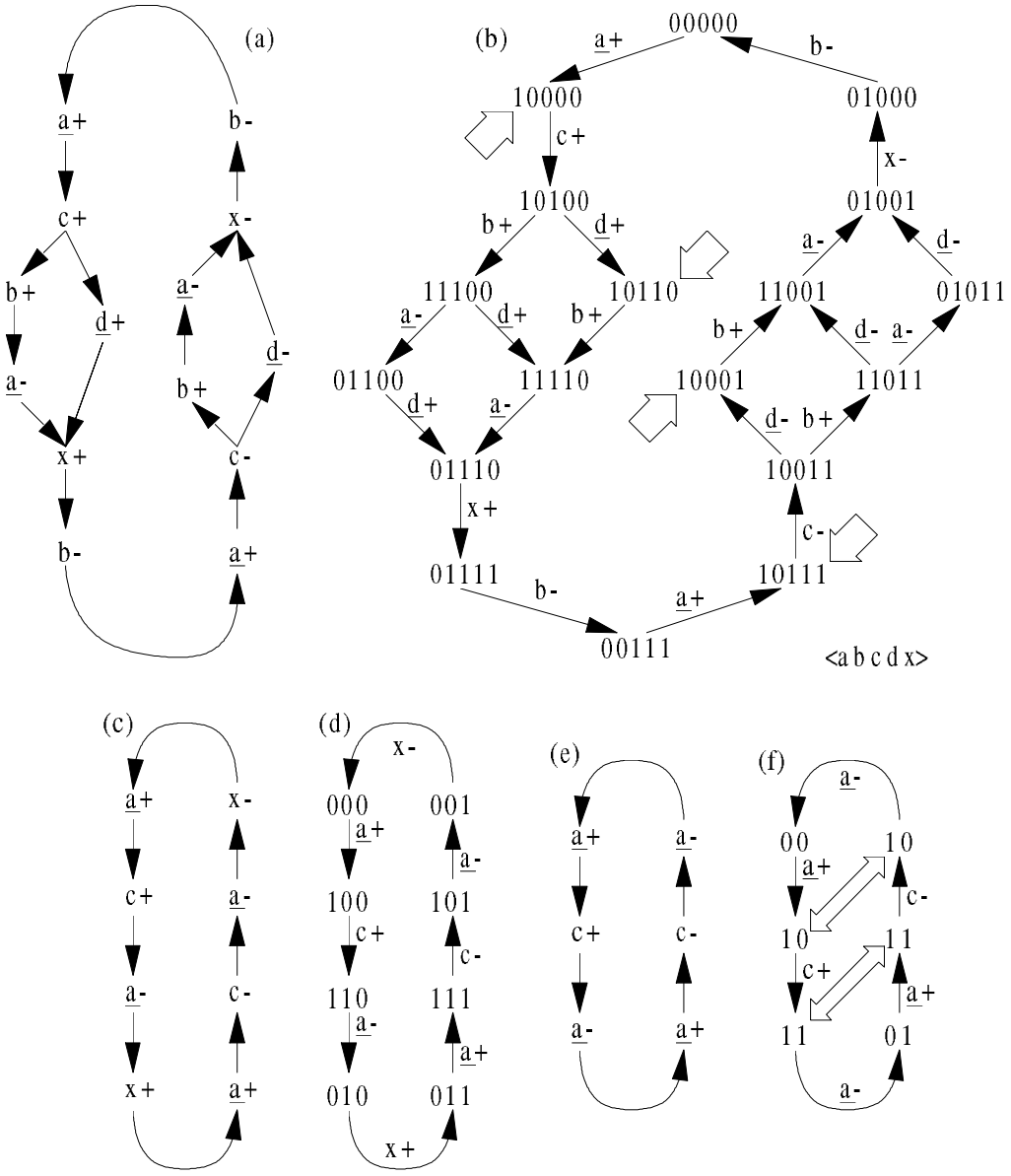


Figure 12: (a) STG of QR42 and (b) state graph. (c) STG_c without the elimination of *x* and (d) state graph. (e) STG_c with the elimination of *x* and (f) state graph with a state assignment problem.

The input set of a signal γ is only capable of selecting a *subset* of essential signals for implementation of γ where the implied value of γ changes in a direct path across the α-boundary as shown in Fig. 13a. This property overlooks the case where the implied values of γ in the two neighbouring states in the K-map across the α-boundary (S_1 and S_2 in Fig. 13) differ due to the indirect path between S_1 and S_2 as shown in Fig. 13b.

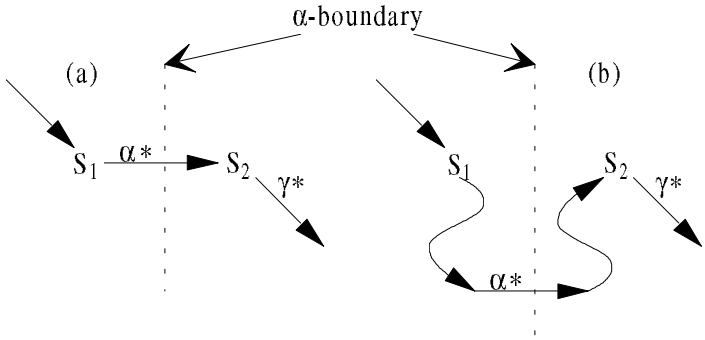


Figure 13. Two possible situations that will result in two neighbouring states across the α-boundary with different implied values for γ: (a) where there is a direct path between S_1 and S_2 and (b) where there is an indirect path.

Owing to the inadequate definition of the input set, I(γ), it may not contain all the necessary signals required for γ's implementation. Signals essential for the implementation of γ can be incorrectly eliminated during net contraction causing state assignment problems in the STG_γ. Hence there is the need to resolve state assignment problems in contracted nets with Chu's technique. The following steps are suggested by Chu to this end:

- The insertion of additional internal signal transitions into the contracted STGs with state assignment problems and the insertion of these signal transitions at their corresponding positions in the original STG description. This procedure requires human judgement and experience, and a well defined algorithm suitable for computer automation has not been established.

- Repetition of the synthesis procedure with the new STG description containing the additional internal signal transitions. This procedure must not remove previously inserted signals for solving state assignment problems otherwise an infinite loop would result!

4. A New Approach to Implementing STGs

To overcome state assignment problems in net contraction and the complications caused, a new approach for the synthesis of self-timed circuits from *uncontracted state graphs* is proposed in this paper. In our new approach, a minimal hazard free implementation of a self-timed circuit is derived from a state assignment problem free state graph description in three stages summarised as follows:

(1) Generation of tabular Karnaugh maps [20 p215-220] for each of the non-input signals.
(2) Derivation of logic expressions for all the non-input signals using Quine-McCluskey tabular Karnaugh mapping.
(3) Elimination of redundant terms in the logic expressions obtained using Prime Implicant tables [20 p220-230].

Although this approach implements circuits from a state graph description only, circuit behaviour specified by Signal Transition Graphs and other descriptions may also be implemented with this technique by first transforming them into a state assignment problem free state graph description as shown in Fig. 14. Note that, without net contraction, this technique is an improved version of the later stages of Chu's technique, approaching the synthesis of STGs along similar lines to the synthesis of Petri nets described in [14].

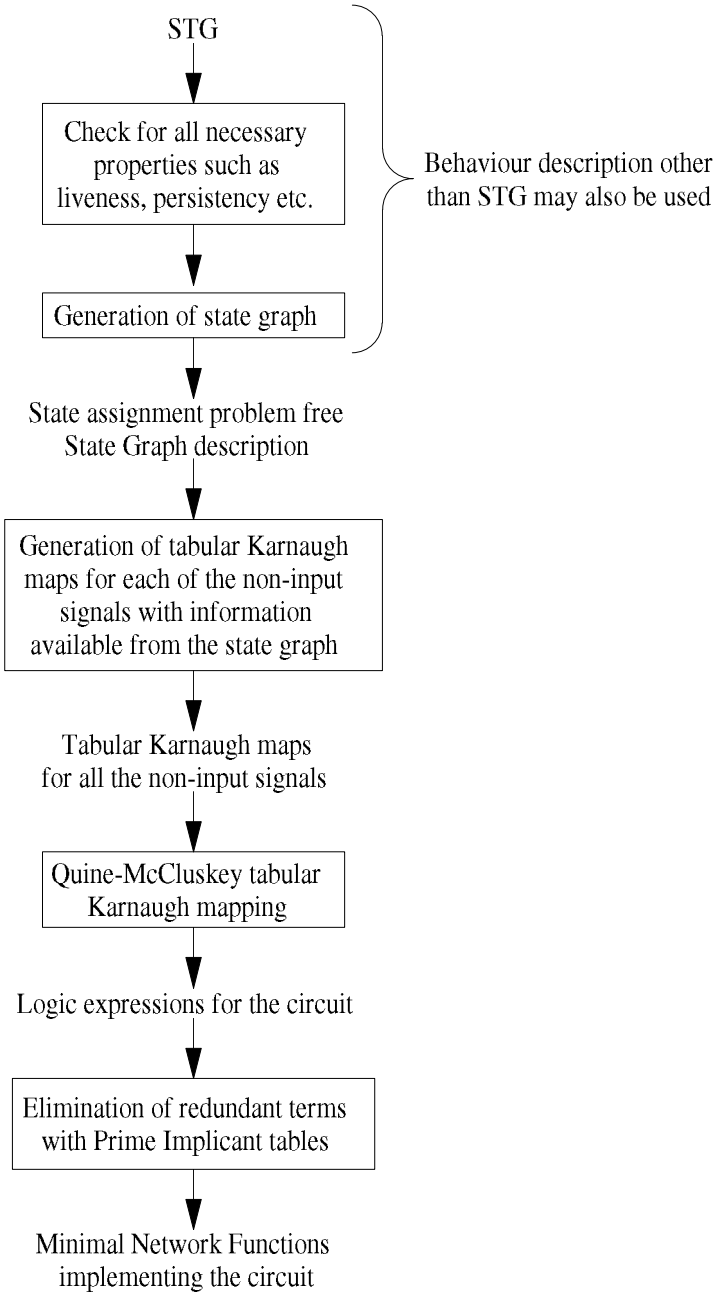


Figure 14. Flow chart summary of the new method.

4.1. Definitions and Algorithms.

A new definition is given below for the implied value:

New Definition of Implied Value

The implied value of a signal j in state s, denoted as $f(s, j)$, is defined as

$$f(s,j) = \begin{cases} X \text{ (don' t care)} & \text{if state } s \text{ is not visited by the circuit} \\ s'(j) & \text{if there exists a state } s' \text{ such that } s[j^*\rangle s' \\ s(j) & \text{otherwise} \end{cases}$$

Where $s(j)$ denotes the logical state of signal j in state s and

$s[j^*\rangle s'$ denotes the presence of an arc from state s to s' caused by signal transition j^* (i.e., $s \xrightarrow{j^*} s'$ in state graph notations).

□

Note, however, that unlike Chu's method, where $f(s, j)$ was defined as

$$f(s,j) = \begin{cases} s'(j) & \text{if there exists a state } s' \text{ such that } s[j^*\rangle s' \\ s(j) & \text{otherwise} \end{cases}$$

$f(s, j)$ has been redefined here to introduce X (don't care) implied value in all states not visited by the circuit[3]. A similar modification has also been utilised in Meng's work [16].

Based on the new definition of implied value, the logic function of a signal j is then defined as follows:

[3] Note that should a noise disturbance perturb the circuit into an unvisited don't care state, the construction of the X state transitions may prevent the circuit ever returning to a valid state. Although not addressed in this paper, the logic designer could take steps to prevent the problem.

Definition of Logic Function

The logic function of a non-input signal j, denoted by f_j, is defined as a Boolean function with domain S, the set of all states traversed by the circuit:

$$f_j(s) = f(s, j) \qquad \forall\, s \in S$$

Using these new definitions of implied value and logic function, tabular Karnaugh maps for all the non-input signals are generated in the first step of the new method with the following algorithm:

Algorithm for Generation of Karnaugh maps

Given the state graph description of a circuit, the K-map of a non-input signal j can be obtained by filling the cells in the K-map with the implied value of signal j at their corresponding states in the state graph. In a similar way, a tabular K-map of a non-input signal j, can be obtained from its implied value as follows.

To determine the logic function of signal j in either the Sum Of Product or the Product Of Sum form, all state values s satisfying the following condition

$$s \in \begin{cases} \{x \mid f(x, j) = 1 \text{ or } f(x, j) = X\} & \text{for logic expression in Sum Of Product form} \\ \{x \mid f(x, j) = 0 \text{ or } f(x, j) = X\} & \text{for logic expression in Product Of Sum form} \end{cases}$$

are arranged into groups according to the number of 1s (or 0s) in the state value.

The logic functions of all the non-input signals (network functions) are then derived using Quine-McCluskey tabular Karnaugh mapping [20 p215-220]. The synthesis procedure concludes with the elimination of redundant terms in these logic functions using prime implicant tables [20 p220-230]. Algorithms for Quine-McCluskey tabular Karnaugh mapping and prime implicant tables are well established and will not be described here. However, it should be mentioned that prime implicant tables used to produce hazard free solutions in this method are slightly modified as compared with the normal prime implicant table [20 p247-249]. Here, each column denotes a transition in the state graph and is specified by the two neighbouring states instead of all neighbouring states with the same implied value. The reason for this

modification is that it is only necessary for the circuit implementation to be hazard free for all signal changes related to the possible state transitions only.

4.2. Example.

An example of the implementation of the now familiar QR42 is presented both for clarification of the new approach and as a comparison with Chu's method. The uncontracted state graph of QR42 is shown in Fig. 15.

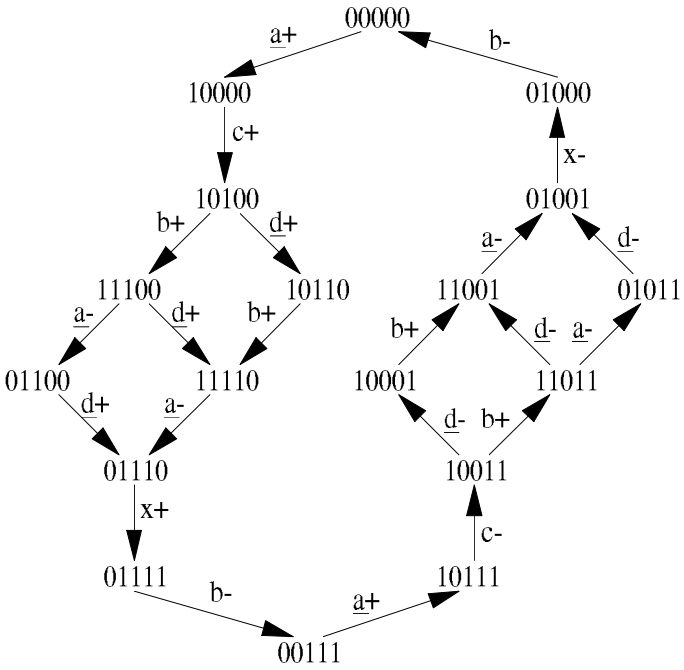


Figure 15. Uncontracted state graph description of QR42.

All the non-input signals, *b*, *c* and *x*, can be implemented in 3 steps as follows:

Step 1: Generation of K-maps.
The implied values of *b*, *c* and *x* are summarised in Table 1. For a sum of products form, tabular K-maps for the signals *b*, *c* and *x* can be obtained by arranging all the state values with 1 and X implied values according to the number of 1s in the state values as shown in Table 2. Note that all state values associated with X implied values are tagged with ⊠.

State value	Implied values of b, c and x		
S = *abcdx*	f(S, *b*)	f(S, *c*)	f(S, *x*)
00000	0	0	0
00001	X	X	X
00010	X	X	X
00011	X	X	X
00100	X	X	X
00101	X	X	X
00110	X	X	X
00111	0	1	1
01000	0	0	0
01001	1	0	0
01010	X	X	X
01011	1	0	1
01100	1	1	0
01101	X	X	X
01110	1	1	1
01111	0	1	1
10000	0	1	0
10001	1	0	1
10010	X	X	X
10011	1	0	1
10100	1	1	0
10101	X	X	X
10110	1	1	0
10111	0	0	1
11000	X	X	X
11001	1	0	1
11010	X	X	X
11011	1	0	1
11100	1	1	0
11101	X	X	X
11110	1	1	0
11111	X	X	X

Table 1. Implied values of b, c and x for all states in the state space.

b	*c*	*x*
☒ 00001	☒ 00001	☒ 00001
☒ 00010	☒ 00010	☒ 00010
☒ 00100	☒ 00100	☒ 00100
	10000	
☒ 00011	☒ 00011	☒ 00011
☒ 00101	☒ 00101	☒ 00101
☒ 00110	☒ 00110	☒ 00110
01001	☒ 01010	☒ 01010
☒ 01010	01100	10001
01100	☒ 10010	☒ 10010
10001	10100	☒ 11000
☒ 10010	☒ 11000	
10100		
☒ 11000		
01011	00111	00111
☒ 01101	☒ 01101	01011
01110	01110	☒ 01101
10011	☒ 10101	01110
☒ 10101	10110	10011
10110	☒ 11010	☒ 10101
11001	11100	11001
☒ 11010		☒ 11010
11100		
11011	01111	01111
☒ 11101	☒ 11101	10111
11110	11110	11011
		☒ 11101
☒ 11111	☒ 11111	☒ 11111

Table 2. Tabular K-maps for *b*, *c* and *x*.

Step 2: Quine-McCluskey tabular Karnaugh mapping.

Quine-McCluskey tabular Karnaugh mapping is now applied, giving the prime implicants[4] for *b*, *c* and *x* shown in Table 3.

[4] A prime implicant is a product term derived from a Karnaugh Map that is contiguous and maximum size.

b	*c*	*x*
$\bar{c}\cdot x$	$\bar{a}\cdot\bar{b}\cdot x$	$\bar{a}\cdot\bar{b}\cdot c$
$\bar{c}\cdot d$	$\bar{a}\cdot\bar{b}\cdot d$	$a\cdot b\cdot\bar{c}$
$\bar{d}\cdot x$	$\bar{a}\cdot c$	$\bar{a}\cdot d$
$c\cdot\bar{d}$	$c\cdot\bar{d}$	$\bar{b}\cdot x$
$d\cdot\bar{x}$	$d\cdot\bar{x}$	$\bar{c}\cdot d$
$c\cdot\bar{x}$	$c\cdot\bar{x}$	$d\cdot x$
$a\cdot b$	$a\cdot\bar{x}$	$c\cdot x$
	$b\cdot c$	$a\cdot x$

Table 3. List of prime implicants for *b*, *c* and *x* generated by step 2 of the synthesis procedure.

As stated above, states associated with X implied values are tagged in the K-maps. These tags are propagated up each stage in the formation of prime implicants when two tagged terms are merged. These terms are redundant and are removed in the last stage of the formation of prime implicants.

Step 3: Elimination of redundant terms with prime implicant tables.
In the final step of the synthesis procedure, redundant prime implicants are eliminated using prime implicant tables. In this example, simpler single output prime implicant tables are used. However, prime implicant tables for multiple-outputs [20 p230-236] may be used to achieve more efficient results for more complex problems.

Prime implicant table for ***b***:

paths	11001	10100	11100	01100	10100	11100	10110	11110	10011	11011	01011	10011	11011	11001
	01001	10110	11110	01110	11100	01100	11110	01110	10001	11001	01001	11011	01011	10001
Minterms														
$\bar{c}\cdot x$ √	×								×	×	×	×	×	×
$\bar{c}\cdot d$												×	×	
$\bar{d}\cdot x$	×													×
$c\cdot\bar{d}$					×	×								
$d\cdot\bar{x}$							×	×						
$c\cdot\bar{x}$ √		×	×	×	×	×	×	×						
$a\cdot b$			×							×				

giving $b = \bar{c}\cdot x + c\cdot\bar{x}$

Prime implicant table for ***c***:

paths	10100	10100	11100	10000	10110	11100	11110	01100	01110	01111
	10110	11100	01100	10100	11110	11110	01110	01110	01111	00111
Minterms										
$\bar{a}\cdot\bar{b}\cdot x$										
$\bar{a}\cdot\bar{b}\cdot d$										
$\bar{a}\cdot c$ √								×	×	×
$c\cdot\bar{d}$		×	×							
$d\cdot\bar{x}$					×		×			
$c\cdot\bar{x}$ √	×	×	×		×	×	×	×		
$a\cdot\bar{x}$ √	×	×		×	×	×				
$b\cdot c$			×			×	×	×	×	

giving $c = \bar{a}\cdot c + c\cdot\bar{x} + a\cdot\bar{x}$

Prime implicant table for ***x***:

paths	01110	01111	00111	10111	10011	11011	10011	11011	10001
	01111	00111	10111	10011	10001	11001	11011	01011	11001
Minterms									
$\bar{a}\cdot b\bar{\cdot}c$									
$a\cdot b\cdot\bar{c}$						×			
$\bar{a}\cdot d$ √	×	×							
$b\cdot x$			×	×	×				
$\bar{c}\cdot d$							×	×	
$d\cdot x$ √		×	×	×			×	×	
$c\cdot x$		×	×						
$a\cdot x$ √				×	×	×	×		×

giving $x = \bar{a}\cdot d + d\cdot x + a\cdot x$

After the elimination of redundant terms with prime implicant tables, the logic expressions for b, c and x are as follows:

$$b = \bar{c}\cdot x + c\cdot\bar{x}$$
$$c = \bar{a}\cdot c + c\cdot\bar{x} + a\cdot\bar{x}$$
$$x = \bar{a}\cdot d + d\cdot x + a\cdot x$$

In contrast, Chu's method produces a **less optimal** result on the same example, requiring an additional internal signal, *y,* to resolve a state assignment problem in the contracted STG for output signal *c* as follows [18 p16-24]:

$$b = \bar{c}\cdot x + c\cdot\bar{x}$$
$$c = \bar{a}\cdot c + c\cdot\bar{y} + a\cdot\bar{y}$$
$$x = \bar{a}\cdot d + d\cdot x + a\cdot x$$
$$y = \bar{a}\cdot c + c\cdot y + a\cdot y$$

5. The Validity of the New Approach.

In this section it is shown that network functions produced from uncontracted state graphs using the new approach are correct and **minimal** in the sense of the number of

gates and inputs to the gates. The discussion is based on a sum of products form, but can be also applied to the product of sums form with appropriate modifications.

5.1. Basic Concepts.

In a state graph description of a circuit with n inputs, outputs and internal signals, the state value is obtained by concatenating the logic values of all the n signals in a predefined sequence. The logic value of each of the n signals can be considered as a coordinate in an n dimensional Boolean state space [19]. Each state in the state graph can be associated with a cell in a Karnaugh map of all the n signals. The important property of the Karnaugh map is that adjacent cells differ in one coordinate only, and therefore all state transitions are between neighbouring cells. Prime implicants obtained from Karnaugh maps can be considered subspaces within the Karnaugh map.

5.2. Proof of Validity.

Theorem 1
The introduction of the X (don't care) implied value for states never visited in the state graph does not modify the state graph description, and as a consequence, does not affect the definition of network functions.

□

With logic functions being defined around the definition of implied value ($f_j(s) = f(s, j)\ \forall\ s \in S$), it is important to prove that the introduction of X (don't care) implied value for all states never visited by the state graph will not affect the definition of f_j. And since network functions obtained from two identical state graphs are identical, to prove the validity for the new definition of $f(s, j)$, it is only required to prove that the introduction of X implied value for all states never visited by the state graph will not modify the state graph description by either inserting or eliminating paths into or from the state graph.

In a state graph description, the absence or presence of a path $s[j^*\rangle s'$ is reflected in the implied value of signal j in state s. $f(s, j) = s(j)$ when $s[j^*\rangle s'$ is absent while $f(s, j) = s'(j)$ when $s[j^*\rangle s'$ is present. As such, to add or eliminate a path $s[j^*\rangle s'$ into or from the state graph, the implied value of j in state s will have to be modified. However, since the introduction of a X implied value does not change the implied value of any state traversed by the state graph, the introduction of these X implied

value will not add to or eliminate path(s) from the state graph, proving the validity of the New Definition of Implied Value and Theorem 1 above.

Definition of n-Space.
A multi-dimensional space of n dimension space with discrete coordinates values of 0 and 1 on each of its dimensions is called an n-space.

□

Definition of p-Subspace.
A space of p dimension within an n-space ($p \leq n$) is called a p-subspace.

□

Having proved the validity of f_j with the new definition of $f(s, j)$, the correctness of the new method and the minimal results it produces are now examined. Firstly, the logic expression for a non-input signal j produced by the new technique is correct if and only if it is consistent with f_j (i.e., the logic expression should produce a logic 1 or a logic 0 in all states $s \in S$ where $f(s, j) \equiv$ logic 1 or logic 0 respectively). With the spatial concept of Karnaugh maps, the correct logic expression for the implementation of signal j is a collection of prime implicants that corresponds to a set of p-subspaces which covers all the 1s within the n-space of implied values. This logic expression is also a minimal hazard free implementation if the set of p-subspaces is a minimum set where all pairs of neighbouring states with the same implied value and associated with an edge in the state graph are contained within at least one p-subspace [20 p247-249].

Theorem 2
The minimal hazard free implementation of a non-input signal is a minimum collection of prime implicants which corresponds to a set of p-subspaces such that all 1s within the n-space of implied values are contained within these p-subspaces. All neighbouring 1s associated with an edge between two neighbouring states in the state graph are also contained within the space of at least one p-subspace.

□

To show that implementations produced by the new method are correct as well as minimal and hazard free, let us examine the new method in its three individual steps as follows. In the first step, tabular Karnaugh maps for each of the non-input signals are set up according to the definition of f_j. This is analogous to filling an n-space with the implied values of j for all the states in an n-space which contains the state graph. Quine-McCluskey tabular Karnaugh mapping in the second step then extracts prime

implicants from the Karnaugh maps which in effect corresponds to a set of the largest p-subspaces which cover all the 1s and *X*s in the n-space of implied values. After step 2, the logic expression for each of the non-input signal *j* then are consistent with f_j, proving the correctness of the result. Finally, in the third step of the new method, prime implicant tables are used to eliminate redundant prime implicant from the expressions obtained in step 2. With each column in these prime implicant tables equivalent to an *edge* in the state graph, the tables will eliminate all redundant terms (p-subspaces) such that all the selected essential prime implicants will then be a collection of a minimum set of p-subspaces. These p-subspaces will contain all the 1s in the n-space of implied values and all neighbouring 1s associated with all the edges in the state graph, proving that the results produced by the new method are also minimal hazard free [20 p247-249].

6. Conclusion.

In this paper, we have shown that Chu's definition of the input set used in net contraction is inadequate for selecting essential signals for implementation of non-input signals. Net contraction can result in state assignment problems in contracted STGs and consequently inefficient implementations. To overcome these flaws in net contraction, we have proposed a new technique for implementing circuits. In the technique, a state assignment problem free uncontracted state graph description of a circuit is used to derive minimal network functions as follows:

(1) Generation of tabular Karnaugh maps for all of the non-input signals.
(2) Application of Quine-McCluskey tabular Karnaugh mapping to determine the logic expressions of all the non-input signals.
(3) Elimination of all the redundant terms in the logic expressions obtained from step (2) using prime implicant tables.

This *single pass* well defined algorithm has been proven to produce correct and minimal results, and suffers no difficulties related to net contraction. Also, unlike Chu's and Meng's techniques which rely on persistent and semi-modular properties to produce hazard free design, the implementation produced with the new technique satisfies other necessary criteria to ensure hazard free operation [21].

References

[1] Chu, T.A. and Glasser, L.A., "Synthesis of Self-timed Control Circuits from Graphs: An Example", In *proc. IEEE International Conference on Computer Design*, pp. 565-571, New York, USA, October 6-9, 1986.

[2] Chu, T.A., "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications", In *proc. IEEE International Conference on Computer Design*, pp. 220-223, New York, USA, October 5-8, 1987.

[3] Chu, T.A., "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications", Ph.D. dissertation Department of Electrical and Computer Science, Massachusetts Institute of Technology, June 1987.

[4] Seitz, C.L., "System Timing" in Mead, C and Conway, L, *Introduction to VLSI Systems*, Chapter 7, pp. 218-262, Addison-Wesley, Reading, Massachusetts, 1980.

[5] Chaney, T.J. and Molnar, C.E., "Anomalous Behaviour of Synchronizer and Arbiter Circuits", *IEEE Trans. on Computers*, Vol. C-22, No. 4, pp. 421-422, April 1973.

[6] Kleeman, L. and Cantoni, A., "Metastable Behavior in Digital Systems", *IEEE Design & Test of Computers*, Vol. 4, No. 6, pp. 4-19, December 1987.

[7] Kleeman, L. and Cantoni, A., "On the Unavoidability of Metastable Behavior in Digital Systems", *IEEE Trans. on Computers*, Vol. C-36, No. 1, pp. 109-112, January 1987.

[8] Lau, C.H., "SELF: A Self-timed Systems Design Technique", *Electronics Letters*, Vol. 23, No. 6, pp. 269-270, March 1987.

[9] Lau, C.H., Renshaw, D. and Mavor, J., "Data Flow Approach to Self-timed Logic in VLSI", In *proc. IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 479-482, Pennsylvania, USA, May 4-7, 1988.

[10] Rosenberger, F.U., Molnar, C.E., Chaney, T.J. and Fang, T-P., "Q-Modules: Internally Clocked Delay-Insensitive Modules", *IEEE Trans. on Computers*, Vol. 37, No. 9, pp. 1005-1018, September 1988.

[11] David, I., Ginosar, R. and Yoeli, M., "Implementing Sequential Machines as Self-Timed Circuits", *IEEE Trans. on Computers*, Vol. 41, No. 1, pp. 12-17, January 1992.

[12] van de Snepscheut, J.L.A., *Trace Theory and VLSI Design*, (Lecture notes in Computer Science 200), Springer-Verlag, New York, 1985.

[13] Molnar, C.E. and Fang, T-P., "An Asynchronous System Design Methodology", Technical Memorandum No. 287, Computer Systems Laboratory, Washington University, St. Louis, Missouri, June 1981.

[14] Molnar, C.E., Fang, T-P. and Rosenberger, F.U., "Synthesis of Delay-Insensitive Modules", In *proc. 1985 Chapel Hill Conference on VLSI*, pp. 67-86, North Carolina, USA, May 15-17, 1985.

[15] Martin, A.J., "A Synthesis Method for Self-timed VLSI Circuits", In *proc. IEEE International Conference on Computer Design*, pp.224-229, New York, USA, October 5-8, 1987.

[16] Meng, T.H.-Y., Brodersen, R.W. and Messerschmitt, D.G., "Automatic Synthesis of Asynchronous Circuits from High-Level Specification", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 8, No. 11, pp. 1185-1205, November 1989.

[17] Horan, P., "A Software for the Design of Delay Insensitive Asynchronous Logic Circuits", In *proc. Microelectronics Conference*, pp. 11-15, Melbourne, Australia, June 24-25, 1991.

[18] Gopalakrishnan, G. and Jain, P., "Some Recent Asynchronous System Design Methodologies", Technical Report UU-CS-TR-90-016, Department of Computer Science, University of Utah, October 1990.

[19] Chung, E. and Kleeman, L., "Circuit Realisation with Uncontracted State Graphs", Technical Report MECSE-90-3, Department of Electrical and Computer Systems Engineering, Monash University, December 1990.

[20] McCluskey, E.J., *Logic Design Principles: with emphasis on testable semicustom circuits*, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[21] McCluskey, E.J., "Transients in Combinational Logic Circuits" in *Redundancy Techniques for Computing Systems*, Wilcox, R.H. and Mann, W.C. (Eds) Spartan Books, Washington 12 DC, 1962.