Realizing Burstmode Circuits via STG Speed Independent Synthesis

Hans Jacobson
Technical Report
Department of Computer Science
University of Utah

Abstract

This report discusses the similarities and differences of STG and Burstmode specifications and synthesis methods. The first part of the report examines the applicability and efficiency of STG's single controller fork-join concurrency ability versus Burstmode's partitioned fork-join concurrency approach. Results comparing the synthesis results for designs using the same level of concurrency in the controllers (STG and Burstmode), as well as the different methods of realizing fork-join concurrency, are presented.

The second part compares the timing assumptions being made by the SI synthesis algorithms and if they can generate a hazard-free solution under Burstmode burst property and fundamental mode assumptions. This comparison shows that speed independent generalized C-element implementations exhibit hazards under the burst property assumption model and can thus not be used to implement Burstmode controllers. It also shows that the SI standard C-element approach, while complying with the burst property of a legal Burstmode specification, may not generate - from a Burstmode point of view - minimum covers. In addition, timed circuits are analyzed for the same hazard considerations. Timed circuits have the same problems as SI when it comes to Burstmode hazard considerations. An extension to timed circuit synthesis that potentially can reduce the number of entrance violations in a standard C-element implementation significantly is also presented.

1 Introduction

In the context of high level synthesis, the control style targeted by the handshake expansion step is important in that it dictates what high level constructs can be supported efficiently at the state machine level. Two main styles of asynchronous control realizations are STG's and Burstmode circuits. These two have different advantages and disadvantages at both the specification and synthesis level. The asynchronous circuit compiler ACK [3] allows fork-join concurrency to be specified in its high level input language. Since it is currently targeting Burstmode style of control implementation though, such concurrency must be realized by implementing each fork-join thread as a separate sub controller that are invoked by a sequential main controller when they are supposed to execute. Since this way of realizing fork-join concurrency introduces some overhead due to start/stop handshaking and signal sharing arrangements, evaluating the more straight forward way of realizing fork-join concurrency in one and the same controller using STG synthesis is of interest. This issue is discussed in section 2.

Another interesting comparison of STG and Burstmode synthesis is to examine the implementations for controllers with the same kind of concurrency. This would potentially help in deciding what synthesis algorithms to use under which circumstances. An interesting aspect of such a comparison is also to compare hazard equalities, i.e. the hazard covers made by the different synthesis algorithms, to determine if Burstmode controllers can be synthesized using speed independent algorithms and the implementations still be hazard-free under burst property and fundamental mode assumptions. This will be discussed in section 4.2. Similar considerations regarding timed circuits will also be discussed in sections 4.4 and 5. This report will discuss the similarities and differences of these methods and present some synthesis benchmark comparisons of the resulting implementations.

2 Specification

Burstmode. Burstmode controllers [6, 8] were developed as a logic extension for single input change asynchronous state machines [7]. The restriction that only one input could change at a time took care of dynamic

hazard problems but was too severe and resulted in significant limitations of concurrency. One answer to this was to allow input signals to arrive in bursts and outputs to be generated as a burst thus allowing a higher degree of concurrency. This specification method still put quite tight restrictions on the concurrency in that it is only allowed in bursts of the same signal type (inputs or outputs). Burstmode has later been extended to allow non-monotonic input level signals and directed don't cares which further extends the concurrency by allowing input signals to be enabled to change over a period of several bursts.

As far as specification is concerned, Burstmode's main advantage is its similarity to synchronous state machines which, by nature, also exhibits a "bursty" nature between clock edges. Its support of non-monotonic input level signals also facilitates interfacing to synchronous or other level based environments.

STG. STG controllers [2] have a higher degree of concurrency than Burstmode in that they allow specification of fork-join type of concurrency. Each thread in such a fork-join can then operate on inputs and outputs independently of eachother while being part of one and the same controller. STG's in their normal form however, do not support level signals which makes it harder to integrate them to a synchronous environment. Recent developments of timed circuits [5], based on STG's, not only allows timing bounds to be associated with each signal transition allowing pruning of unreachable states, but also incorporates level signals. Addressing issues about level signals however, is out of scope for this report.

2.1 Fork-join concurrency using Burstmode

One of the major problems with the burst property of a Burstmode specification is the lack of fork-join type of concurrency where the threads are allowed to contain arbitrary sequences of input/output signal events.

While STG's are general enough to allow specification of this type of concurrency, Burstmode effectively restricts the allowed concurrency to signal bursts of either inputs or outputs. To perform handshake expansion from a high level description containing fork-join concurrency that targets Burstmode type of control is therefore not as straight forward as for the STG case and requires partitioning of incompletely specified controllers [4]. To allow such concurrency, the threads must be separated into subcontrollers which then can be made to execute in parallel by requests from the sequential main controller. While this method of realizing fork-join concurrency generally reduce the complexity of the individual controllers, there is added overhead in starting and detecting the completion of each such subcontroller. In addition, when signals are shared between such incompletely specified subcontrollers, extra logic must be added to resolve the sharing. For input signals this can be done by using a *blocker gate* approach to only allow the signal to reach the subcontrollers that are currently executing and supposed to see the change on the input. For each controller containing the shared signal, a blocker gate is allocated. A control signal associated with each controller then controls if the event should be propagated through the blocker gate (to the controller) or not. The blocker gate can be implemented as an AND gate in the case of four phase protocol or a SELECT element in case of two phase. For output signal sharing, merge elements such as OR and XOR gates can be used.

This logic must be added *sequentially* to the critical signal path through the controller as illustrated in figure 1. Subsequently it may significantly reduce controller performance unless the complexity of the subcontrollers themselves have been reduced by the partitioning sufficently to outweigh this fact. Something that is becoming less likely with recent advances in automated technology mapping of large controller circuits [1].

The question then is, if STG's with their ability to specify fork-join signal concurrency within the same controller can generate a potentially better solution that, with application of technology mapping, can yield better performance on average.

Since issues such as technology mapping and controller placement and routing are affected by the decision to partition or not, this study can not give a definitive answer for the general case. The results obtained however indicates that STG's are better to implement short fork-join threads. while partitioned Burstmode is better suited for implementing larger fork-joins due to the reduced logic complexity of non-shared signals thanks to partitioning. One example that compares the efficiency of the two implementation approaches more in detail is presented in section 6.

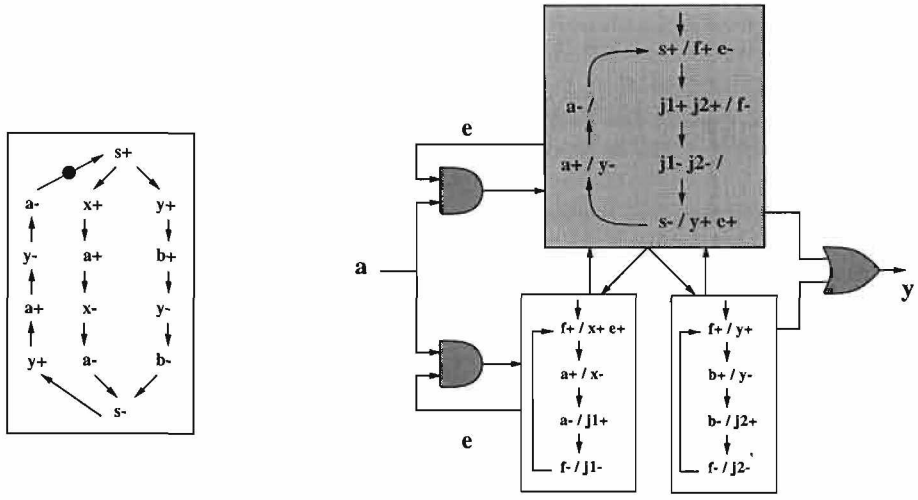


Figure 1: Fork-join realization using STG and Burstmode

3 Converting Burstmode to STG

In the process of comparing the Burstmode and STG approaches, it is important to realize specifications that are exactly the same. We therefore discuss how to implement an automatic translator from a Burstmode specification to an STG, or more specific, Event Rule (ER) specification.

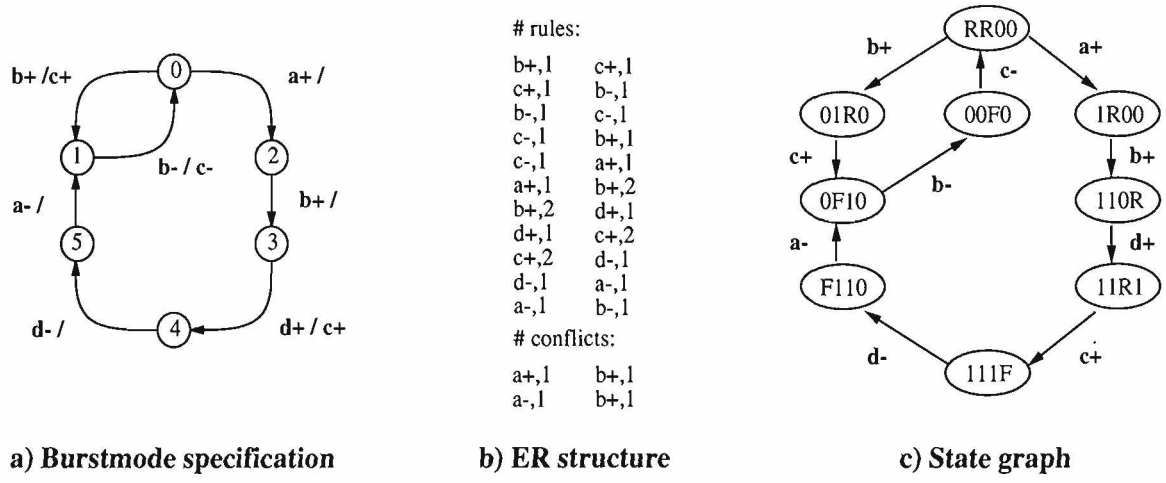
One main difference in the specification of STG's compared to Burstmode controllers is that STG's synthesis methods require the environment to be specified in order to find all reachable states of the design. AFSM specifications like Burstmode do not explicitly model the environments behavior, but due to the controllers single threaded nature the environment can unambiguously be derived from the controller specification.

In order to find the enabling and enabled signals, or rules, which then describe the behavior of both controller and environment, of an ER, it is therefore necessary to find the environment of the Burstmode controller to be translated. To find the environment we must *mirror* the behavior of the controller. By observing that a Burstmode controller at any given time only executes a single thread, albeit with burst concurrency, the mirror behavior can be informally defined as follows.

1. An input burst of the controller is an output burst of the environment.
2. An output burst of the controller is an input burst of the environment.
3. An output burst of the environment *enables* an output burst of the controller if and only if the current state of the controller has a non-empty output burst. Otherwise an output burst of the environment *enables* an input burst of the controller.
4. An input burst of the controller *enables* an input burst of the environment.

Note that this simple definition only holds true for Burstmode specifications. Extended Burstmode's addition of directed don't cares requires that the enabling signals for such a directed don't care is the output (or input) burst of the previous state of where the directed don't care is first encountered. Also a directed don't care can only enable the output burst in the state it is forced to evaluate (or the input burst of the next state). It is also worth noting that while the basic definition for Burstmode mirroring given above always will guarantee persistency in the ER, the introduction of Extended Burstmode and its directed don't cares may result in persistency violations of the trigger signals of the directed don't care signal.

To disambiguate concurrency from choice and merge we also need to define conflicts. In a choice place, each signal in one branch of the choice conflicts with the signals of every other branch. The conflicts in a merge place is similarly defined but with the "branches" going into the place rather than out of it.



a) Burstmode specification **b) ER structure** **c) State graph**

Figure 2: Burstmode translation to ER structure

A translation example is illustrated in figure 2. As can be observed in the definition of the mirror behavior above, we can directly translate the input burst of state 3 of the controller to enable the output burst of the same state. Similarly the output burst of this state enables the input burst of the next state. Notice that in state 2 the output burst is empty wherefore the input burst, rather than the output burst, of this state enables the input burst of the next state.

4 Synthesis

The question now that we have a way to convert a Burstmode specification to a STG specification is, can a Burstmode controller be implemented using speed independent synthesis algorithms and still be hazard-free under the Burstmode timing assumptions? This section will try to answer that question by analyzing what hazard covers are made by the SI algorithms and how they correspond to Burstmode hazards.

4.1 Burstmode circuits

When a Burstmode specification is entered into the synthesis algorithms it is first converted into a flow table. This flow table is then minimized to reduce the number of states. This minimized flow table then forms an initial hazard-free implementation of single minterm excitation region covers. After state assignment, logic minimization is performed to reduce the complexity of the circuit. During logic minimization, special care must be taken as to not introduce any dynamic hazards by expanding cubes such that they intersect a dynamic transition trajectory without including the corresponding start state of the transition. In a Burstmode controller all signals within a specified input burst may arrive in arbitrary order and with arbitrary times in between. This timing property will be refered to as the Burstmode burst property. Burstmode circuits also operate under fundamental mode assumption, meaning that the circuit as a whole must stabilize after an input burst before the next input burst is allowed to arrive at its inputs.

4.2 Speed independent circuits

When the STG specification, is entered into the synthesis algorithms it is first converted into a state graph representing all reachable states of the circuit. The resulting state graph is then synthesized using different assumptions on gate implementation structure, i.e. single generalized C-element (gC) gate or multiple gates of a standard C-element (sC) implementation. Each separate gate though is still considered to be atomic and have an arbitrary propagation time. Each excitation region is then synthesized separately. The general algorithm, also called multi-cube, then proceeds by finding all prime implicants that can be used to implement the current excitation region. Since the multicube algorithm allows covering of an excitation region with an arbitrary SOP

cover rather than one single cube, we will sometimes refer to such SOP covers as gate-blocks since they are considered to be one atomic block.

Under the generalized C-element model, which assumes the entire complex gate is atomic and thus evaluates immediately upon arrival of any input signal, any set of prime implicants that completely covers the excitation region can be used to implement a cover.

The standard C-element model assumes that individual gates are atomic. The implementation structure for this model can be seen in figure 3. Since the circuit consists of more than one level of gates, while gates are atomic the propagation of a gates new value may take arbitrarily long time, and thus the whole set or reset *function* may not have finished evaluating before new inputs arrive. This could potentially create a hazard when evaluation of the set and reset functions overlap. The synthesis algorithm must in this case remove the cause of such hazards. This is done by introducing the concept of *entrance violations* which are solved by first finding the *implied states* of every prime implicant. An implied state is a state from which we illegally may enter the cube cover of a prime implicant, thus causing a hazard. Second we split the prime implicants that have implied states into *candidate implicants* that have less entrance violations. A *CC-table* containing columns for both the states of the excitation region that must be covered by some implicants, and the implied states of the implicants is then constructed. A solution covering all columns of the table is then found.

4.3 Hazard equivalence

This section will explore the similarities between hazards in SI and Burstmode realizations. The important questions are what Burstmode hazards under burst property and fundamental mode assumptions may manifest in a circuit synthesized using speed independent algorithms under atomic gC and sC implementation assumptions. We assume that a SI sC can in fact be implemented as a gC Burstmode circuit. Such a transformation is legal under Burstmode burst property and fundamental mode assumptions. Since these Burstmode assumptions do not make any assumptions about gate atomicity any hazards exhibited by the SI standard gate implementation will be preserved in the transformation to a Burstmode generalized C-element.

4.3.1 Burstmode dynamic hazards.

Take the example in figure 3 which models the introduction of a possible dynamic hazard in a Burstmode cover. For the sake of the discussion, assume that cubes A, B, and C are prime implicants found by the Burstmode logic minimizer and SI multi-cube algorithm respectively. The transitions we will consider are $t1$ followed by $t2$. Consider two different scenarios.

1. If for dynamic transition $t1$ signal a arrives before signal b, then the transition trajectory passes briefly through state $s1$ and cube A, before reaching its end state in cube C.

   Under the SI atomic gate assumption, this transition is hazard-free in the gC approach since the gate evaluates immediately as it enters each state. This means that the A cube will always have switched on before the state covered by the C cube is entered and subsequently there will be no hazard. Since selecting such a cover would introduce a dynamic hazard under Burstmode burst property and fundamental mode assumptions, where gates are not assumed to be atomic, SI gC implementations can not be used to synthesize hazard-free Burstmode circuits. (Consider the case when C starts pulling down the output and is interrupted by a late conducting A transistor stack.)

   Under the SI atomic gate assumption, this transition is not hazard-free in the sC approach since the gate implementing cube A, while evaluating immediately, may take an arbitrary long time to propagate its value. As illustrated in figure 3, the internal node X can then exhibit a glitch while transition $t2$ is being performed, resulting in an incorrect final value of the output. This potential hazard however, is removed by the SI sC algorithm since moving over cube A in this fashion indicates an entrance violation which is solved by reducing cube A to, in this case, only cover the excitation region minterm. The SI sC hazard removal addresses the Burstmode dynamic hazard issue even under the burst property and fundamental mode assumptions and can thus generate a hazard-free cover for the specified transitions.

2. If for dynamic transition $t1$ signal b arrives before signal a, then the transition trajectory passes briefly through state $s2$, before reaching its end state in cube C.

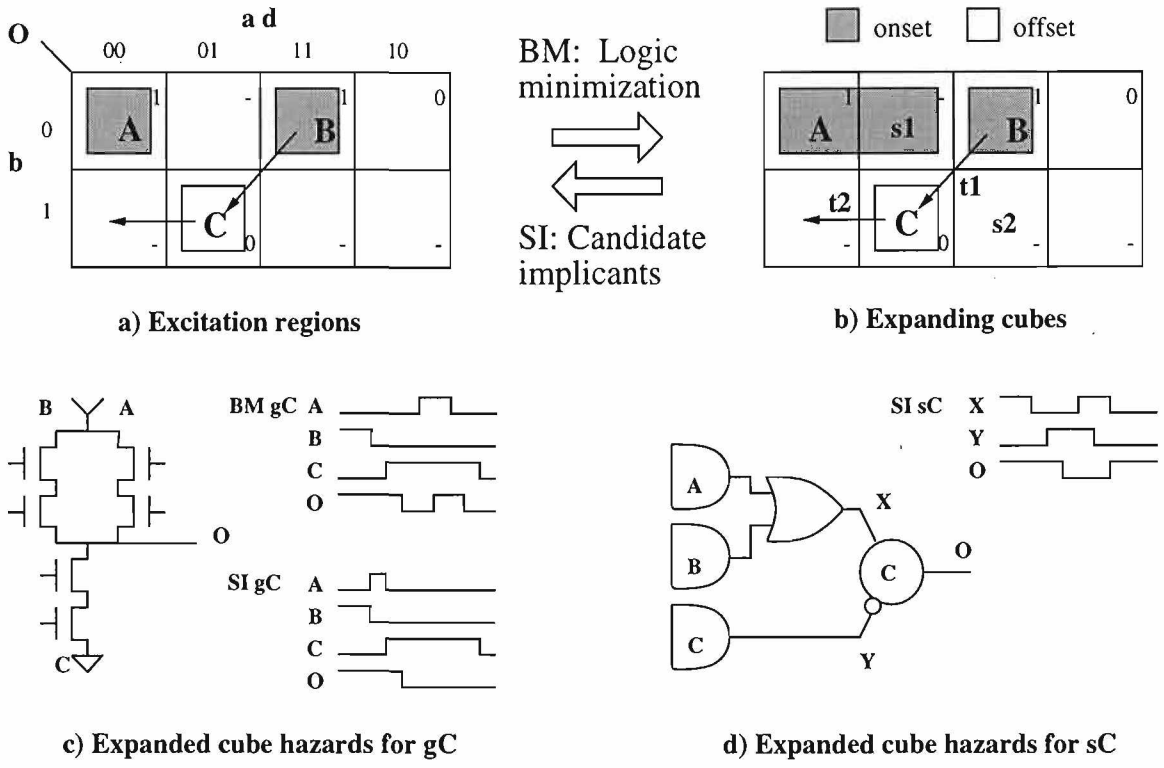


Figure 3: Comparison between Burstmode and SI hazards

Under the SI atomic gate assumption, this transition is hazard-free in the gC approach since the gate evaluates immediately as it enters each state. This means that the transition is guaranteed not to turn on A during the transition. When the end state C is reached, the gate evaluates and eventually propagates a monotonic transition to the output. While this holds true for the SI gC approach due to the atomic gate assumption, it does not hold true under the Burstmode burst property assumption. This is due to the gate not being considered atomic and thus even if signal b arrives before a at the perimeter of the gate, they may arrive at the different stacks inside the gate in arbitrary order thus making it possible to enter state $s1$, switching on cube A. If A is slow and C switches on first, it may cause a hazard on the output.

Under the SI atomic gate assumption, the $t1$ transition is hazard-free in the sC approach since the gate implementing cube A evaluates immediately due to signal c disabling it from any further changes. A will therefore remain off during the transition, and when entering the end state of $t1$, C will evaluate and eventually propagate a monotonic transition to the output. While this holds true for the SI sC approach due to the atomic gate assumption, it does not hold true under the Burstmode burst property assumption. This is due to the gate for A not being considered atomic and thus even if signal b arrives before a at the perimeter of the gate, they may arrive at the different stacks inside the gate in arbitrary order thus making it possible to enter state $s1$. This may cause C to switch on before A, causing a hazard on the output.

4.3.2 Burstmode static hazards.

Take the example in figure 4 which models the introduction of a possible static hazard in a Burstmode cover. (Note however that this hazard will not be seen at the output of the circuit due to the staticizer on the output.) For the sake of the discussion, assume that cubes A, B, and C are prime implicants found by the Burstmode logic minimizer and SI multi-cube algorithm respectively. The transitions we will consider are $t1$ followed by $t2$ followed by $t3$. Consider two different scenarios.

1. If for static transition $t1$ signal a arrives before signal b, then the transition trajectory passes briefly through state $s1$ and cube A, before reaching its end state in state $s3$. The following dynamic transition $t2$ then

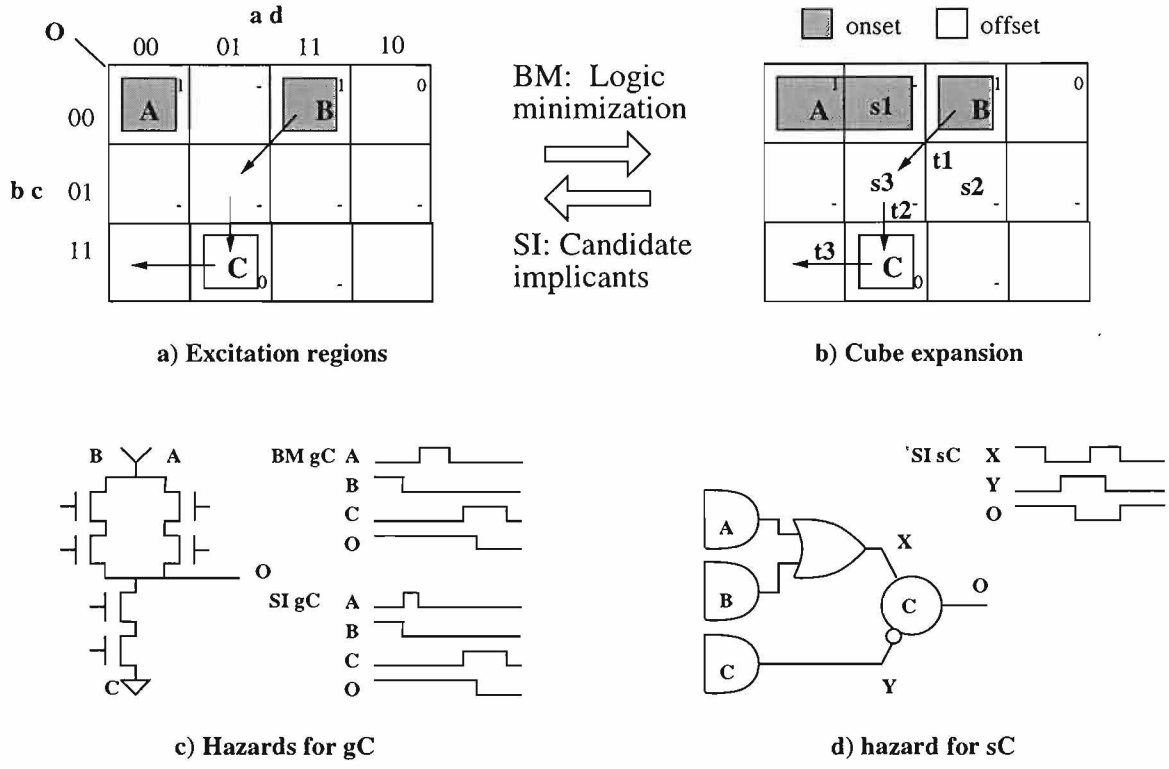


Figure 4: Comparison between Burstmode and SI hazards

moves to its end state covered by cube C.

Under the SI atomic gate assumption, transitions $t1 + t2$ are hazard-free in the gC approach since the gate evaluates immediately as it enters each state. This means that the A cube will always have switched on before the state covered by the C cube is entered and subsequently there will be no hazard. This cover is also hazard-free under Burstmode burst property and fundamental mode assumptions since, by the assumptions, it is guaranteed that the circuit will attain quiescence after $t1$ before the input burst of $t2$ will arrive. Static hazards are therefore of no concern in a Burstmode gC implementation.

Under the SI atomic gate assumption, transitions $t1+t2$ are not hazard-free in the sC approach since the gate implementing cube A, while evaluating immediately, may take an arbitrary long time to propagate its value. As illustrated in figure 3, the internal node X can then exhibit a glitch while transitions $t2 + t3$ are being performed, resulting in an incorrect final value of the output. This potential hazard however, is removed by the SI sC algorithm since moving over cube A in this fashion indicates an entrance violation which is solved by reducing cube A to only cover the excitation region minterm. The SI sC hazard removal addresses the Burstmode static hazard issue although such hazards can not manifest in the corresponding Burstmode gC implementation. The cover produced by the SI sC implementation is therefore not minimal in the number of literals compared to the hazard-cover requirements to achieve a hazard-free Burstmode cover under burst property and fundamental mode assumptions.

2. If for static transition $t1$ signal a arrives before signal b, then the transition trajectory passes briefly through state $s1$ and cube A, before reaching its end state in state $s3$. The following dynamic transition $t2$ then moves to its end state covered by cube C.

   Under the SI atomic gate assumption, transitions $t1 + t2$ are hazard-free in the gC approach since the gate evaluates immediately as it enters each state. This means that the transition is guaranteed not to turn on A. When $t2$ reaches the state covered by C, the gate evaluates and eventually propagates a monotonic transition to the output. Since $t1$ and $t2$ are separate transitions and a Burstmode circuit is guaranteed to

attain quiescence under burst property and fundamental mode assumptions in between burst transitions, regardless of the arrival order of a and b to internal stacks, the output will be hazard-free.

Under the SI atomic gate assumption, transition $t1 + t2$ are hazard-free in the sC approach since the gate implementing cube A evaluates immediately due to signal c disabling it from any further changes. A will therefore remain off during the transition, and when entering the end state of $t2$, C will evaluate and eventually propagate a monotonic transition to the output. Since $t1$ and $t2$ are separate transitions and a Burstmode circuit is guaranteed to attain quiescence under burst property and fundamental mode assumptions in between burst transitions, regardless of the arrival order of a and b to internal stacks, the output will be hazard-free.

4.3.3 Conclusions

From the discussion in previous subsections, it is clear that a SI gC cover is not enough to guarantee hazard-freeness under Burstmode burst property and fundamental mode assumptions. SI sC covers on the other hand not only provide required hazard-free covers, but also introduces extra hazard covers of hazards that cannot manifest in a controller restricted to a legal Burstmode behavior. Subsequently speed independent synthesis can not implement circuits obeying Burstmode burst property and fundamental mode assumptions as well as Burstmode synthesis can.

The following equations then informally express these observations in a clear and straight forward manner.
Literal count: $SI\ gC \leq BM\ gC \leq SI\ sC$
BM hazards: $SI\ gC \geq BM\ gC = SI\ sC = 0$

4.4 Timed circuits

As discussed in the previous section, a SI model cannot deal with the fundamantal mode issue. Since no fundamental mode assumption is made the algorithm cannot assume that the circuit has stabilized before the next input signal event occur. This problem is inherent to the timing model of speed independent circuits and cannot be solved without violating the SI assumption.

There is another approach called *timed circuits* [5] that also starts from a STG specification. This approach annotates signal transitions with timing bounds in an attempt to prune the state graph of unreachable states. This approach does not require a SI model but rather extends the specification to allow putting bounds on the delay of a circuit. Gates are still considered atomic however.

For example, one may specify that the response time for a given output to react to a given input event will be within a certain timing bound. When the STG specification, or rather an ER structure, is entered into the synthesis algorithms, a timing analysis step is first performed. This timing analysis use the timing information from the specification and tries to find unreachable states in the state graph. Such states can then be removed from the state graph reducing the complexity of the specification to be synthesized without changing its external behavior. The resulting state graph is then synthesized using the same speed independent assumptions on gate behavior as discussed in section 4.2.

Once unreachable states due to timing specifications in the state graph have been removed, the synthesis proceeds exactly the same as in speed independent synthesis. Using the same figures as in the SI case, the difference in the interpretation of hazards that were discussed in section 4.2 then will then change to the following.

Burstmode dynamic hazards.

1. If for transition $t1$ signal a is always guaranteed to arrive before signal b, then state $s1$ is unreachable and thus removed from the state graph. One might believe that this would solve the problem of cube A causing a hazard since it is not reachable. While this holds true for both the gC and sC SI approaches due to atomic gate assumptions, it does not hold true under the Burstmode burst property. This is due to the gate not being considered atomic and thus even if signal a arrives before b at the perimeter of the circuit, they may arrive at the different stacks inside the circuits in arbitrary order thus making it possible to enter state $s1$. This may cause C to switch on before A causing a hazard on the output. (Observe that state $s1$ may still be covered by a prime implicant as a don't care despite it no longer being in the state graph.)

2. If for transition $t1$ signal b is always guaranteed to arrive before signal a, then state $s2$ is unreachable and thus removed from the state graph. This however, does not in any way affect the possibility of $t1$ to briefly enter cube A followed by C. The hazard considerations will therefore be exactly the same as for dynamic hazards in section 4.2.

Burstmode static hazards. Similar observations as those made for the Burstmode dynamic hazards above can be made to the static hazard considerations in section 4.2.

As we can see from the comparison, timed circuit synthesis algorithms in their current form cannot deal with Burstmode hazards any better than the speed independent synthesis (since the synthesis algorithms after state graph pruning are, in fact, the same). The problem is that while timing analysis is used on the state graph to remove unreachable states, no timing is used by the synthesis algorithms when finding implied states.

5 Extended timed circuits

In this section an extension to timed circuit synthesis is proposed that will help reduce the number of implied states caused by static transitions.

Due to the lack of timing considerations during the actual synthesis phase, timed circuits can not take advantage of a property fundamental to achieving less entrance violations, namely the assumption that the circuit may have stabilized between static transitions. (Observe though that during a dynamic transition under SI assumption, the circuit is assumed to have stabilized as soon as the output has been propagated since only one gate-block may be on at the same time.) Entrance violations for such cases must therefore still be considered during the covering step. The key to remove implied states thus is being able to determine which static transitions will obey the fundamental mode assumption and which will not. For every static transition that will obey the fundamental mode assumption we do not have to find any implied states. We informally define a transition obeying the fundamental mode assumption as a transition that after reaching its end state, the circuits internal nodes will stabilize before the next transition out of this state commences. Note that the assumption of atomic gates in sC implementations still is valid under the new fundamental mode assumption. In this case, the fundamental mode assumption refers to all internal gate outputs (nodes) of the circuit having reached their final values for the given state.

By using timing information conveying how long the maximum time the circuit may take to stabilize after receiving a certain input and compare this to the minimum time until the next input signal arrives to the circuit, we can determine if the fundamental mode assumption is violated or not. If it is violated, then we must consider implied states, but if it is met there is no reason to find implied states. This approach then removes the constraint that two gate-blocks can not be on at the same time, as two covers now not only can be on at the same time due to different gate delays, but their cubes are actually allowed to overlap eachother if they do not cross any transition trajectory that does not obey the fundamental mode assumption. Note that this method can also be used to annotate static subtrajectories of a transition to find out if indeed the transition can be broken up into two transitions, the first of which obeys the fundamental mode assumption. This method can of course also be used to model settling time due to state variable feedbacks by treating them as input transitions. The assumption that the circuit has stabilized after a dynamic transition as soon as the output event has occurred is also true under this new assumption since although several cubes may be on at the same time, all internal nodes are stable at the beginning of the transition. In a reset region, that means that all set region cubes that are initially on must monotonically go off before the C-element is enabled to respond to the reset region cube going on. Note that while several cubes can switch off, but only one cube can switch on. Otherwise the circuit would not be guaranteed to have stabilized at the time the output changes value. A gate-block can thus not cover *subsets* of other excitation regions. It is allowed however, to extend into another excitation region if it will cover it completely, thus enabling us to remove the other cover, resulting in only one gate-block covering each excitation region (thus several excitation regions may be covered by the same gate-block).

ER rule extension. In order to derive how long time it will take the circuit to stabilize due to a static transition we must extend the rule concept of the ER structure. A rule can now also be used to specify enablings of non-enabled signals. I.e. the enabling signals in this case are input signals that are making transitions in the output signals quiescent states.

A "non-enabled" (fundamental mode) rule can then also be annotated with the maximum time it will take the cover of the excitation region of the non-enabled signal to settle after the enabling signal(s) have arrived. A fundamental mode rule will therefore have the following syntax:
$signal_{enabling}$ $signal_{non-enabled}$ $marking_{initial}$ $time_{min}$ $time_{max}$ $time_{stabilize}$
The extended multi-cube algorithm then becomes the following:

1. Find *prime implicants*
2. Find *implied states*
3. Find *candidate implicants*
4. Construct CC table
5. Solve CC table

Where the extended definition of the "find implied states" step will be the following:

- A state s is an implied state of an implicant c for the excitation region $ER(u*, k)$ if s is not covered by c and is a predecessor of a state s' that is both covered by c and in the quiescent set, and the transition $s_i \rightarrow s'$ does not obey the fundamental mode assumption, i.e.
$IS(c) = s | s \notin c \wedge \exists s'[(s, s') \in \Gamma \wedge s' \in c \wedge s' \in QS(u*) \wedge !FM(s, s')]$

Where compliance with the fundamental mode assumption can be expressed as follows:

- A cover c obeys the fundamental mode assumption for a transition t if its maximum settling time s_t for t is greater than the minimum arrival time of the next input event i.

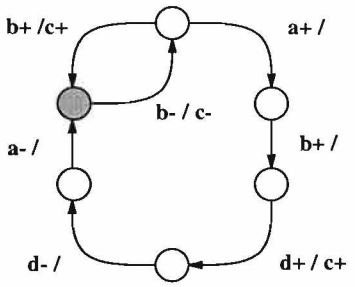
The rest of the synthesis steps are not changed. In the following example we abstract the settling time annotation to just indicate that the circuit indeed is guaranteed to have settled before the next input transition can occur. If the full scope of the timing analysis were to be used, the maximum settling time would have to be compared to the minimum arrival time of the next input transitions to determine if the fundamental mode assumption is violated or not.

An example of using this method is illustrated in figure 5. As shown the ER structure has been extended with two fundamental mode rules (markings, min, and max timing not shown) that lets the synthesis algorithms know that these static transitions ($a-, 1$ and $d-, 1$) for excitation region $c+, 1$ can be treated as if they obey the fundamental mode assumption. In other words we are guaranteed that transition $a-, 1$ will not occur before the logic covering the excitation region for $c+, 1$ has stabilized after transition $d-, 1$. Similarly we are guaranteed that $b-, 1$ does not occur until the logic has settled after transition $a-, 1$. Without using this fundamental mode annotation, prime implicant $a'b$ has $F110$ as an implied state forcing the implicant to be split searching for candidate implicants. Using the fundamental mode annotation, the prime implicant does not have any implied state and can be used in the cover without introducing extra implied state columns that must be covered by other implicants. The cover using the fundamental mode annotation then becomes $a'b$, and when not using it becomes $a'bc'$.

As with the minimum and maximum timing bound annotation in a timed circuit specification, the fundamental mode settling time annotations in the extended timed circuits can be gradually introduced and tightened. This allows great flexibility in how much timing information the designer wants to enter into the specification.

Extended timed circuits relation to Burstmode. By using timed circuit synthesis with the proposed fundamental mode annotation, The properties of Burstmode controllers can be modeled more precisely in the STG synthesis. Since unnecessary covers due to implied states of static transitions that actually obey the fundamental mode constraint are no longer included in the solution, the literal count compared to that achieved by Burstmode synthesis should be the same, or very close.

We must observe however that when several cubes are allowed to be on at the same time in the sC implementation, when translated to a Burstmode gC implementation the cirucit is not guaranteed to have settled by the time the output changes. This is due to transistor stacks having different drive strength which may result in one stack changing the output value while a weaker stack in the opposite transistor network remains conducting.



a) Burstmode specification

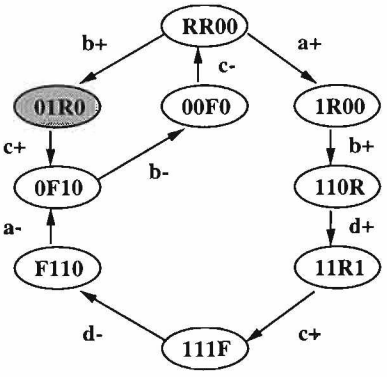
normal rules:

b+,1	c+,1
c+,1	b-,1
b-,1	c-,1
c-,1	b+,1
c-,1	a+,1
a+,1	b+,2
b+,2	d+,1
d+,1	c+,2
c+,2	d-,1
d-,1	a-,1
a-,1	b-,1

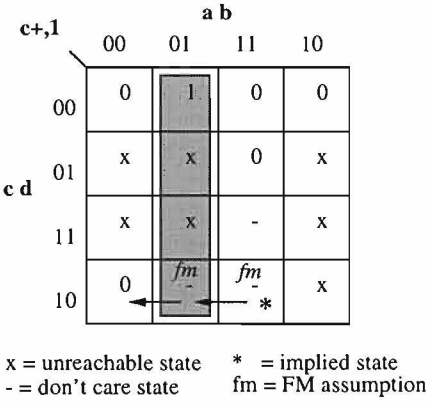
fundamental mode rules:

a-,1	c+,1	*fm*
d-,1	c+,1	*fm*

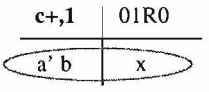
b) Extended ER structure



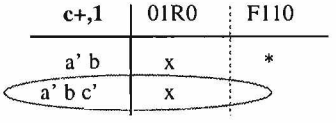
c) State graph specification



d) Prime implicant



e) CC table using FM timing



f) CC table not using FM timing

Figure 5: Example of extended timed circuit synthesis

By also annotating enabled signals (as opposed to non-enabled), i.e. the rules of an output's excitation region, with a maximum settling time, we can decide if the resulting cover can be safely translated into a gC implementation. Note that if the fundamental mode assumption annotation method is extended like so, the minimum arrival time for the next input transition can always be annotated with a value large enough to satisfy this constraint. By inserting delays on signal wires we can therefore always ensure that the fundamental mode assumptions are met and that the sC circuit can be safely translated into a hazard-free gC circuit seen from a Burstmode point of view. Note however that this extra extension to fundamental mode annotation has no meaning for sC implementations since gates are assumed to be atomic.

6 Results

The similar results for the SI gC and Burstmode gC implementations illustrated in figure 6 are a bit surprising since the SI gC should potentially be able to find better covers since it does not consider any kind of logic hazards. Observe that the SI gC implementations are not guaranteed hazard-free under Burstmode burst property and fundamental mode assumptions. As expected, the SI sC implementations sometimes have to remove hazards for static transitions, hazards that cannot actually occur under the Burstmode assumptions, and thus result in larger covers. The large difference in the synthesis time is due to the SI synthesis using a very efficient single cube algorithm based on graph traversal rather than finding prime implicants as in the Burstmode case.

As can be seen in figure 7, the introduction of fork-join concurrency in the SI design did result in a complexity increase. It is hard to make a fair comparison though since not the same state variable assignments can be used (as

		Rules	I/O	Time	Literals
GCD 2p	**3D gC**	320	18	840	140
	ATACS gC			11	140
	sC			15	247 *
Factorial 2p	**3D gC**	92	16	88	46
	ATACS gC			1	46
	sC			1	54 *
Factorial 4p	**3D gC**	95	16	83	42
	ATACS gC			1	42
	sC			1	42
scsi_isend	**3D gC**	81	10	30	48
	ATACS gC			1	51 *
	sC			1	60 *
diffeq_mul1	**3D gC**	48	7	30	31
	ATACS gC			1	31
	sC			1	33 *
sbuf_send_ctl	**3D gC**	34	8	24	30
	ATACS gC			1	30
	sC			1	35 *
mp_fwd_pkt	**3D gC**	24	7	27	14
	ATACS gC			1	16
	sC			1	16

Figure 6: Comparison between Burstmode and SI implementations

in the comparison of the same-concurrency implementations). Since the state variable assignment in the SI case was fairly naive, better results are to be expected when a good state assignment algorithm is used. The figure also illustrates the added complexity of the same partitioned Burstmode implementation due to signal sharing logic. To get a fair comparison of the number of transistors an input signal must take through the circuit before generating an output, the figure also illustrates the average "transistor-depth" of each output signal. This also takes into account the signal sharing logic for the partitioned Burstmode case. The similar "gate-depth" of the signal paths in the partitioned Burstmode and centralized SI realizations indicates that the centralized approach would have an advantage due to its greater ability to take advantage of technology mapping. The dissimilar gate-depth for the *fab1req* and *fab2req* signals in the fork-join can to a large part be explained by the naive state assignment in the SI case.

References

[1] BEEREL, P. A., CHUN CHOU, W., AND YUN, K. Y. A heuristic covering technique for optimizing average-case delay in the technology mapping of asynchronous burst-mode circuits. In *Proc. European Design Automation Conference (EURO-DAC)* (Sept. 1996).

[2] CHU, T.-A. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications.* PhD thesis, MIT Laboratory for Computer Science, June 1987.

[3] KUDVA, P. *Synthesis of Asynchronous Systems Targeting Finite State Machines.* PhD thesis, Computer Science Department, University of Utah, 1995.

[4] KUDVA, P., GOPALAKRISHNAN, G., AND JACOBSON, H. A technique for synthesizing distributed burst-mode circuits. In *Proc. ACM/IEEE Design Automation Conference* (1996).

[5] MYERS, C. J. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits.* PhD thesis, Dept. of Elec. Eng., Stanford University, Oct. 1995.

[6] NOWICK, S. M. *Automatic synthesis of burst-mode asynchronous controllers.* PhD thesis, Computer Systems Laboratory, Stanford University, 1993.

		Rules	I/O		Time	Literals
ForkJoin	**3D gC** *main*		18		56	42
	fj1		7		24	13
	fj2		10		35	26
	sharing		9		-	6
	ATACS gC	109	26		76	93
	sC				-	-

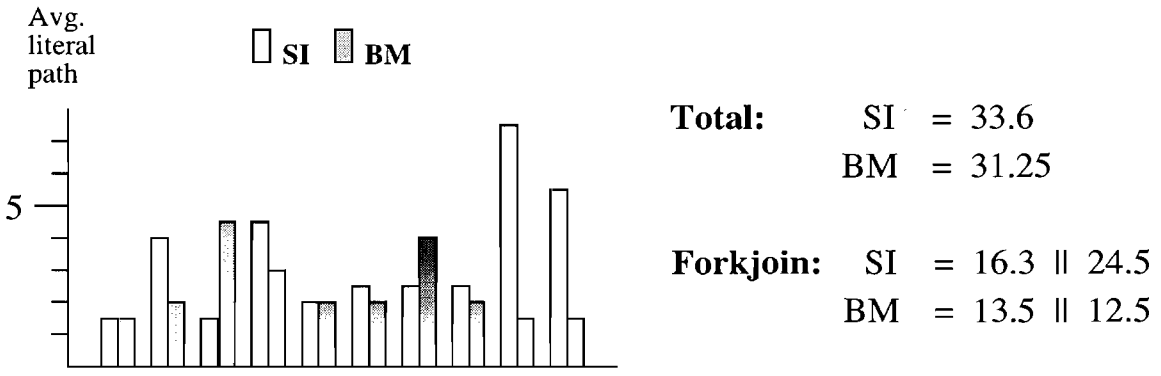


Figure 7: Comparison between Burstmode and SI fork-join concurrency

[7] UNGER, S. H. *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[8] YUN, K. Y. *Synthesis of asynchronous controllers for heterogeneous systems.* PhD thesis, Stanford University, Aug. 1994.