Achieving Fast and Exact Hazard-Free Logic Minimization of Extended Burst-Mode gC Finite State Machines

Hans Jacobson*
Department of Computer Science
University of Utah
hans@cs.utah.edu

Chris Myers†
Department of Electrical Engineering
University of Utah
myers@ee.utah.edu

Ganesh Gopalakrishnan‡
Department of Computer Science
University of Utah
ganesh@cs.utah.edu

Abstract

This paper presents a new approach to two-level hazard-free logic minimization in the context of extended burst-mode finite state machine synthesis targeting generalized C-elements (gC). No currently available minimizers for literal-exact two-level hazard-free logic minimization of extended burst-mode gC controllers can handle large circuits without synthesis times ranging up over thousands of seconds. Even existing heuristic approaches take too much time when iterative exploration over a large design space is required and do not yield minimum results. The logic minimization approach presented in this paper is based on state graph exploration in conjunction with single-cube cover algorithms, an approach that has not been considered for minimization of extended burst-mode finite state machines previously. Our algorithm achieves very fast logic minimization by introducing compacted state graphs and cover tables and an efficient single-cube cover algorithm for single-output minimization. Our exact logic minimizer finds minimal number of literal solutions to all currently available benchmarks, in less than one second on a 333 MHz microprocessor — more than three orders of magnitude faster than existing literal exact methods, and over an order of magnitude faster than existing heuristic methods for the largest benchmarks. This includes a benchmark that has never been possible to solve exactly in number of literals before.

1 Introduction

Burst-mode and extended burst-mode asynchronous finite state machine controllers have been successfully used in designing several efficient real-life asynchronous circuits [1, 2, 3, 4, 5]. For such large designs, many iterations through the synthesis process are needed to reach a good circuit implementation. For example, different protocols and state assignments could lead to substantial improvements in circuit area and delay. Since existing exact synthesis methods are slow, and may not even complete for large circuits, designers are often forced to use heuristic methods, or partitioning [6], to interactively explore the design space in a reasonable amount of time often yielding sub-optimal circuits. The bottleneck of extended burst-mode finite state machine synthesis is typically in the hazard-free logic minimization step. This paper will address a new method that allows fast algorithms based on state graph exploration to produce solutions that are hazard-free under the extended burst-mode hazard model. This new method allows very fast logic minimization that is exact in number of literals, even for controllers that previously have been impossible to solve exactly in number of literals due to their large size.

Hazard-free logic minimization of two-level sum-of-products burst-mode circuits has been pursued by Nowick et al. Literal exact, cube exact, and heuristic solutions have been explored in the tools `HFMIN` [7], `IMPYMIN` [8], and `Espresso-HF` [8]. While the two later algorithms have managed to reduce logic minimization time, algorithms for higher quality literal-exact solutions still takes thousands of seconds for large controllers, effectively setting a limit on exploration of the design space.

Algorithms based on state graph exploration have traditionally been used for synthesis of speed-independent [9, 10] and timed [11] circuits. Some researchers [9, 10] have proposed to use speed-independent methods for logic minimization of burst-mode gC controllers by first translating them into Petri-net specifications. The generated solutions are hazard-free under the constrained gC decomposition rules imposed by the relaxed burst-mode hazard model presented in [12]. However, these methods are not hazard-free for *extended* burst-mode controllers.

Structural logic minimization methods to combat the problem of state explosion in state graph traversal have been explored [13, 14, 15]. These methods perform logic minimization while traversing the input Petri-net specification avoiding generation of potentially large state graphs. These methods, however, experience a substantial increase in Petri-net size for non-monotonic level signals (frequently used in extended burst-mode specifications), and are not hazard-free for extended burst-mode controllers.

Recent efforts have been made to produce optimal output solutions over all state reductions and encodings [16], thus avoiding manual design exploration of different state assignments. While these methods are intriguing, the sheer algorithm complexity currently limits the size of controllers that can be automatically explored.

In more recent designs, such as Yun's differential equation solver [4] and the burst-mode portions of Intel's RAPPID [17], it has been found that substantial performance improvements can be achieved through the use of generalized C-element (gC) circuit implementations over two-level standard gate implementations. In [18], state minimization and state assignment methods are developed for extended burst-mode controllers based on excitation region covers allowing for gC circuit implementations. These synthesis methods have been implemented in the `3D` tool [19]. `3D` leverages the `HFMIN` tool to perform hazard-free logic minimiza-

*Supported by the University of Utah Graduate Research Fellowship award.
†Supported in part by NSF CAREER award MIP-9625014, SRC contract 97-TJ-487, and a grant from Intel Corporation.
‡Supported in part by NSF Award MIP-9622587.

tion and [20] to perform technology mapping. The hazard-free logic minimization method presented in this paper performs the same function as the HFMIN tool in the 3D synthesis flow.

Contribution

This paper addresses the important milestone of achieving fast and exact hazard-free logic minimization. It does so in the context of hazard-free logic minimization of extended burst-mode gC finite state machines. By combating state explosion and dividing the minimization problem into more easily solved sub-problems, our new algorithm allows very fast logic minimization of even the largest burst-mode benchmarks to date and yields per-output literal-exact solutions, enabling truly interactive and iterative exact exploration of design alternatives.

Our approach is orthogonal to structural Petri-net based minimization methods by leveraging standard synthesis techniques traditionally used in speed-independent and timed circuit synthesis, namely state graph exploration and derivation of single-cube covers [11, 21, 22, 23, 24]. Our method introduces significant enhancements and extensions to these techniques that are necessary to reach the aggressive goal of allowing interactive exploration of large designs. More specifically, this paper introduces a new approach that allows state graphs to be represented very compactly, transforming the graph traversal complexity from potentially exponential, to linear. Compacted states also allow reduced synthesis time in the subsequent binate and unate cover problems by generating smaller cover tables. A new efficient algorithm to derive single-output covers that are minimal in number of literals is also introduced. By dividing the cover problem into more easily solved sub-problems and then merging the results, the time spent in finding a minimal solution is significantly reduced. Also part of the logic minimization methodology has been the development of methods to handle the concept of extended burst-mode hazards during state graph exploration and cover minimization.

Section 2 describes extended burst-mode specifications, hazard models, and required covers. Section 3 gives background on state graphs and the single-cube cover algorithm leveraged by our new method. Section 4 presents compacted state graphs along with a new exact algorithm that achieves per-output literal-exact cover solutions. Section 5 presents benchmark comparisons between the algorithm presented in this paper and other available extended burst-mode logic minimization tools. Conclusions can be found in Section 6.

2 Extended burst-mode background

This section starts out by giving a background view of extended burst-mode specifications. This is followed by an overview of delay assumptions, hazard models, and requirements for hazard-free synthesis.

2.1 Extended burst-mode specification

Extended burst-mode controllers [19] belong to the multiple input change (MIC) classification of asynchronous finite state machines. The controllers work under a fundamental mode assumption that requires the logic to stabilize between two distinct sets of input changes. Inputs are allowed to change concurrently in an extended form of MIC called *bursts*. The signals within a burst may arrive in arbitrary order. Every input burst is followed by (a possibly empty) concurrent burst of output and state signal changes. After the output and state burst, the internal nodes of the circuit must attain quiescence before the fed back state signals are allowed to reach the inputs of the circuit. The circuit must then

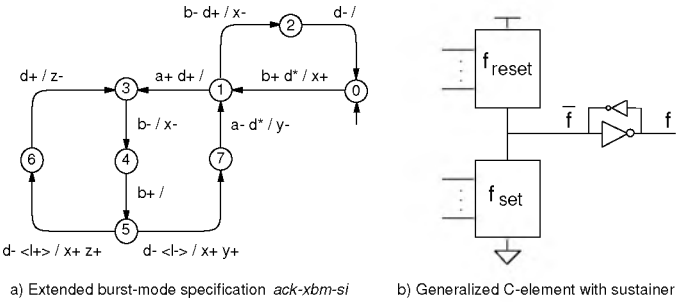

Figure 1. Extended burst-mode specification and gC implementation structure.

stabilize in response to these state signals before the next input burst may arrive. This cyclic operation of extended burst-mode circuits is similar to the way synchronous state machines operate, which, by nature, also exhibit a "bursty" nature between clock edges.

The extended burst-mode specification shown in Figure 1(a) is used as an example in later sections. Each transition between states contains an input burst and a possibly empty output burst. If there is more than one outgoing transition from a state, a deterministic choice is implied. Signals annotated with $+$ and $-$ signs and not enclosed in angle brackets imply a rising and falling transition, also known as a *terminating edge*. Input signals annotated with a $*$ are called *directed don't cares* and are free to change monotonically at any time during a sequence of specified directed don't care states but must change by the time the signal is next specified as a terminating edge. A terminating edge not preceded by a directed don't care is called a *compulsory edge*. Signals enclosed within angle brackets are *conditionals* (level signals) and are free to change non-monotonically whenever not specified. When specified, such a conditional is assumed to reach a stable value in time to be sampled correctly by the arriving compulsory edges of the burst. State transitions occur only when all conditionals are met and all terminating edges pertaining to a burst have appeared.

2.2 Extended burst-mode synthesis

Classic extended burst-mode synthesis follows a path similar to synchronous finite state machine synthesis. An extended burst-mode controller is first state assigned to implement a next state function corresponding to the initial specification. The state assignment step must ensure that the resulting function is free of function hazards and must also take steps to ensure that an implementation free of logic hazards exists. The next step is to find a minimal next state circuit implementation of the state assigned specification that is free of logic hazards through logic minimization. Logic minimization produces a set of minimized logic equations which can then be technology mapped into a real circuit and analyzed for timing to determine required delays on feedback paths. The work in this paper focuses on the hazard-free logic minimization step of this synthesis procedure. This section presents the theory behind extended burst-mode transitions and hazards so that these concepts can later be discussed in the context of logic minimization using our approach which is based on state graph exploration and the single-cube algorithm.

Timing requirements. Extended burst-mode assumes a delay model with arbitrary delays in gates and wires. To guarantee a correct implementation, three timing requirements are imposed by the extended burst-mode synthesis method. *Fundamental-mode environmental constraint*: no compulsory edge of the next input

burst may appear until the circuit has attained quiescence. *Feedback delay requirement*: fed back variable changes must not reach the inputs until all enabled outputs and state variables have completed. *Setup and hold time requirements*: all conditional signals specified to be stable must stabilize before any compulsory edge appears and must remain stable until the output and state burst has been completed.

gC implementation structure. In an extended burst-mode gC implementation of the output function f, the set logic (f_{set}) and reset logic (f_{reset}) are both implemented as two-level AND-OR circuits. The on-set of f (f_{set}) is implemented by the N-stack transistor network and the off-set (f_{reset}) is implemented by the P-stacks. The arguments for hazard-free covers are analogous for f_{set} and f_{reset}. Figure 1(b) illustrates the structure of an extended burst-mode gC circuit.

Extended burst-mode transitions. When inputs are allowed to change non-monotonically during multiple-input changes, the classical definition of hazards is insufficient. In [19], Yun introduces the notion of *generalized transitions* to enable a framework for discussing the new hazards that can occur in an extended burst-mode environment. A generalized transition defines the set of all legal trajectories in transition cube $[S,E]$, where S is a *start cube* and E is an *end cube*. A generalized transition can have three types of inputs: *rising-edge*, *falling-edge*, and *level*. Edge inputs change monotonically (i.e., at most once in a legal trajectory). If specified, level inputs must hold the same value in S and E, where the value is either a constant 0 or 1, to comply with the setup and hold time requirements. Level signals, if they are not specified in the transition, may change non-monotonically. The *start subcube*, s, is a maximal subcube of S such that each signal undergoing a directed don't care transition is set to its initial value (i.e., 0 for a rising-edge and 1 for a falling-edge). The *end subcube*, e, is a maximal subcube of E such that each signal undergoing a directed don't care transition is set to its final value (i.e., 1 for a rising-edge and 0 for a falling-edge). These subcubes fill an important role in the definition of hazard-free covers. An extended burst-mode transition is a *static* generalized transition for a given output if all minterms in [S,E] yield the same value for the output. It is a *dynamic* $1 \rightarrow 0$ generalized transition if all minterms in [S,E) (i.e., all minterms in the transition excluding those in E) yield 1 and all minterms in E yield 0 on the output. A dynamic $0 \rightarrow 1$ transition is similarly defined.

Combinational logic hazards. A logic implementation of a function f free of function hazards is said to have a logic hazard for an input transition if for some assignment of delays to gates and wires the output can exhibit a glitch during the transition interval. There are two types of combinational logic hazards that must be considered in an extended burst-mode circuit: *static* and *dynamic* hazards. However, in a gC implementation, static hazards cannot manifest themselves on the output [12]. Also, note that an extended burst-mode specification is free of function hazards after state minimization and assignment [12], a requirement necessary to find an implementation free of logic hazards.

In [12], a combinational network is said to contain a dynamic logic hazard during a function-hazard-free input change from start cube S to end cube E when $f(S) \neq f(E)$ and the circuit's output can potentially change non-monotonically during the input change from S to E. It is proven in [12] that when a product term contains the start subcube s (or the end subcube e), then the product term changes monotonically during the corresponding extended burst-mode transition. Finally, it is shown that the output of a two-level combinational AND-OR circuit is hazard-free during a $1 \rightarrow 0$ or $0 \rightarrow 1$ extended burst-mode transition $[S,E]$ if and only if every product term intersecting the subtransition $[S,E)$ contains the start subcube s and every product term intersecting E contains end subcube e.

Assuming that the transistor stacks internal to a complex gate switch simultaneously, an output function implemented as a monolithic gC circuit (gC core) cannot exhibit hazards on its output. However, due to the weak current drive strength of high transistor stacks, it is often desirable to partially decompose a gC core into a two-level network of smaller gates with higher drive strength. Such decomposition is also used to decrease the delay of the most frequent paths through the controller, making the common case faster. When allowing arbitrary decomposition of a gC core, the hazard constraints for a two-level combinational AND-OR circuit, presented above, are required since delays introduced by decomposed gates may cause stacks to no longer switch simultaneously. However, by avoiding short-circuits (i.e., not allowing the decomposition of trigger signals which, during the same transition, enable and disable a P and N stack pair), the hazard constraints can be relaxed to no longer require the inclusion of the start subcube s when a product term intersects the subtransition $[S,E)$. Note, however, that a product term intersecting the end cube E of a dynamic transition $[S,E]$ still must contain the end subcube e. Further discussion on extended burst-mode hazards and short-circuit avoidance can be found in [12, 25, 20]. This paper uses the hazard assumptions described above.

Extended burst-mode required cover. Given the hazard requirements discussed above, the hazard-free cover requirements for an extended burst-mode gC set output function f_{set} becomes:

1. Each set cube of f_{set} must not intersect the off-set.
2. For every dynamic $0 \rightarrow 1$ transition $[S,E]$ in f_{set}, the end cube, E, (i.e., the *excitation region*) must be completely covered by some product term.
3. Any product term of f_{set} intersecting the cube E of a dynamic $0 \rightarrow 1$ transition $[S,E]$ must also contain the end subcube e.

The second requirement describes the product terms, also called *required cubes*, that are required for the cover to turn on when it is supposed to. The first and third requirements describe the constraints that the required product terms must satisfy for the cover to be hazard-free. As shown in Section 4.3, all of these requirements are satisfied by the logic minimization algorithm presented in this paper. Hazard-freedom requirements for f_{reset} are analogous to f_{set}.

3 State graph and single-cube background

As the logic minimization approach presented in this paper is based on state graph exploration and the single-cube cover algorithm, this section first gives some background definitions used for these approaches.

State graph definitions. A *state graph* is a graph in which nodes are labeled with the current state vector. Signals in the state vector that are enabled to rise or fall are annotated with a superscript *. The edges of the state graph are optionally labeled with the names of the signals that are enabled to change. An excitation region for an output signal is a maximal connected set of states in a state graph in which the signal is enabled to change. If the signal is rising (falling) in the region, it is called a *set region* (*reset region*). Each excitation region in an extended burst-mode circuit can be implemented with a single conjunction of literals (i.e., a single *cube*) which corresponds to a *cover* of the excitation region. The cover of an excitation region is a set of states for which the corresponding cube in the implementation evaluates to

one. Each cube in the implementation is composed of *trigger signals* and *context signals*. For an excitation region, a trigger signal is a signal whose firing can cause the circuit to enter the excitation region. The set of trigger signals for an excitation region can be represented with a cube called a *trigger cube*.

The single-cube algorithm. The single-cube algorithm [11] is a fast algorithm for finding a solution to controllers where each excitation region can be covered by a single cube. An important property of extended burst-mode controllers is that an excitation region can always be represented by a single cube. In a burst-mode specification, every excitation region spans only one minterm due to the delay constraint imposed on fed-back state and output signals (i.e., no changes on inputs are visible throughout the excitation region). In an extended burst-mode specification, directed don't cares and level signals can expand this minterm into a cube, but since all input and output signals that can be used as context signals are persistent, the excitation region can still be covered by a single cube. Therefore, the single-cube algorithm is guaranteed to always find a solution for extended burst-mode controllers.

The input to the single-cube algorithm is a complete state coded state graph and the output is a set of minimized logic equations for each output signal implementing each set and reset region in the state graph specification. The single-cube algorithm finds a hazard-free cover by first analyzing the state graph to derive an initial cover based on trigger signals for each excitation region. Since this initial cover may include minterms of the opposite set, called *cover violations* (CV), as well as hazards, called *intersection violations* (IV), a binate covering problem is then formulated to remove such violations by adding context signals to the cover.

4 Extended burst-mode logic minimization using the single-cube algorithm

Although the single-cube algorithm is very fast for medium sized state graphs (several hundred states), large and highly concurrent extended burst-mode controllers can yield very large state graphs, making state graph traversal and solving of cover tables a time-consuming process. This section introduces the concepts of compacted state graphs and compacted cover tables to enable very fast logic minimization of large and complex controllers.

Extended burst-mode gC controllers are generally best implemented with a minimal literal count *per output* cover (also known as single-output minimization). Such a cover criteria exploits product term sharing while at the same time guarding against excessive sharing which could result in product terms with too high literal counts (as often happens in the simpler to derive minimal cube covers). However, since the standard single-cube algorithm by default generates minimal covers *per excitation region*, it cannot be directly used to find high quality extended burst-mode implementations. This section therefore also presents a new algorithm that extends the single-cube algorithm to find per-output minimal literal covers, a much harder problem than the single-cube algorithm default of finding per-excitation-region minimal literal covers.

4.1 Algorithm overview

Figure 2 illustrates the high-level outline of our logic minimization algorithm. The logic minimization starts out by first traversing the state graph, deriving the set of regions where the output currently being minimized is enabled to change (excitation regions). The algorithm then proceeds by finding the initial

```
cost_function = minimum literal cardinality;
// Divide:
foreach output set or reset function in StateGraph {
  ERs = each excitation region of output;
  foreach er in ERs {
    TCs = find_trigger_cubes(er, StateGraph);
    CSs = find_context_signals(er, StateGraph);
    // Resolve:
    foreach (tc, cs) in (TCs, CSs) {
      foreach state in StateGraph {
        CVs = CVs ∪ find_cover_violations(tc, state);
        IVs = IVs ∪ find_intersection_violations(tc, cs, state); }
      table_binate = build_binate_cover_table(tc, cs, CVs, IVs);
      Solutions_local = Solutions_local ∪ solve_binate_cover_table(table_binate); }}
  // Merge:
  table_unate = build_unate_cover_table(ERs, Solutions_local);
  Solution_output = solve_unate_cover_table(table_unate, cost_function);
  Solution_controller = Solution_controller ∪ Solution_output; }
```

Figure 2. Extended single-cube algorithm for per-output minimal literal count solutions.

trigger cubes and context signals for each excitation region separately. Eventual cover and intersection violations are then found, and a binate cover table is built and solved in order to remove any such violations from the initial cover. Except for the concept of intersection violations which was developed by the authors to handle extended burst-mode hazards, the steps described so far form the original single cube algorithm which finds per-excitation-region minimal covers. In order to achieve higher quality covers, the single cube algorithm is extended to generate per-output minimal covers. The extensions are implemented by the last five lines in the algorithm in Figure 2. Rather than finding a single cover cube per excitation region, the extended single cube algorithm finds the set of minimal and unique cubes that could potentially be part of a final minimal cover. These sets of local solutions are then merged into a final minimal solution by setting up and solving a unate cover problem. The following subsections will discuss the compacted state graphs and the extended single cube algorithm in more detail.

4.2 Compacted state graphs

The underlying problem of state explosion of highly concurrent controllers is that every extended burst-mode burst must be expanded into all possible interleavings of signal edges. This is because signals pertaining to a burst may arrive in arbitrary order. The number of states caused by a burst grows exponentially with the number of signals in the burst. The situation is made even worse when non-monotonic level signals are present in the specification. Since any unspecified level signal may take on an arbitrary value, each state in the state graph corresponding to a terminating edge must be split into 2^n states, where n is the number of level signals defined in the specification, to model all possible combinations of values that the level signals may currently exhibit. As a high degree of signal concurrency is often required to extract as much parallelism from a design as possible and level signals are frequently used to read the status of datapaths, clearly we do not wish the minimization algorithm to be exponentially dependent on the degree of signal concurrency and level signals present in the specification. This state explosion can be combated by introducing the concept of *compacted state graphs*. A compacted state represents the set of states reachable by all possible interleavings of a set of concurrently enabled signals. An important property of compacted states is that they must not contain

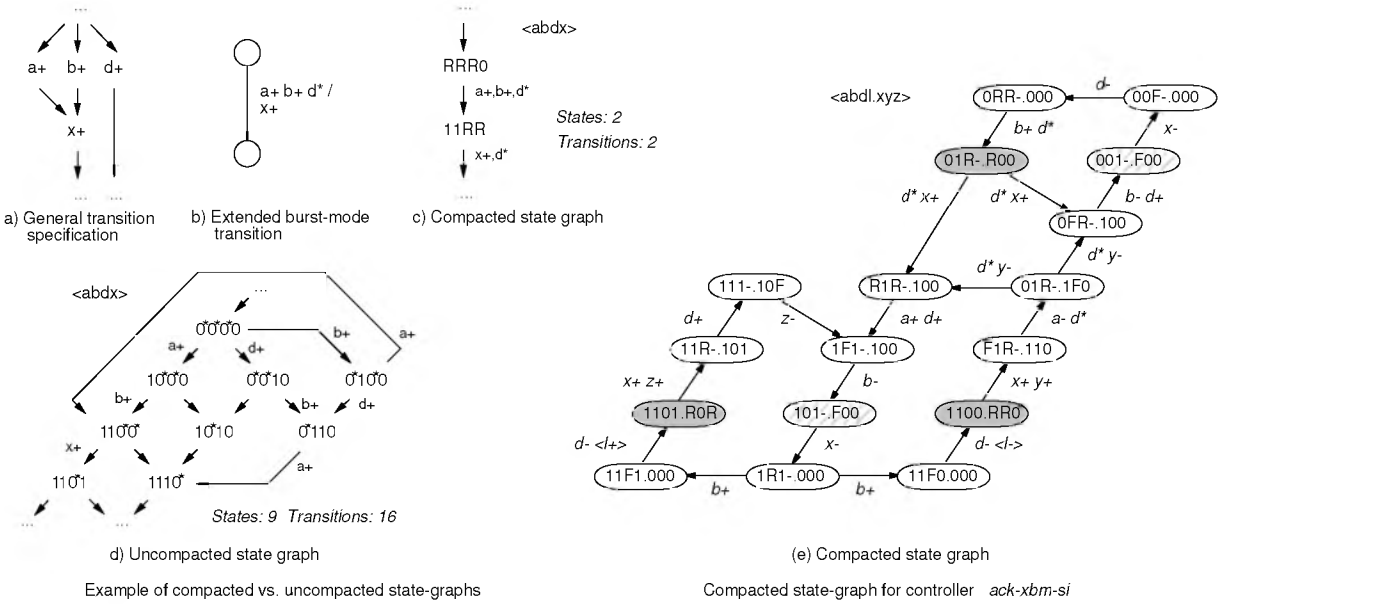

Figure 3. Compacted state graphs.

unreachable states or else synthesis results may not be optimal.

A compacted state is a cube consisting of all states reachable by all possible interleavings of the set of currently enabled monotonic (edge) and non-monotonic (level) input and output signals. A compacted state is represented by a set of 1's and 0's for non-enabled signals, a set of R's and F's denoting rising and falling edges of enabled monotonic signals (terminating edges as well as directed don't cares), and a set of -'s denoting the unknown value of enabled non-monotonic level signals.

When building a compacted state graph from a generic signal transition based specification, a new compacted state is only added to the compacted state graph when a new signal becomes enabled or when a signal in the current state becomes enabled in the opposite direction. Note that the definition of a compacted state may allow compacted states to overlap. As an example consider the generic transition based specification shown in Figure 3(a). This specification corresponds to the extended burst-mode burst $a+,b+,d*/x+$ in Figure 3(b). When entering the specified portion of the graph, three signals $a+$,$b+$, and $d+$ are enabled. A compacted state $abdx$:RRR0 is created which covers all states reachable by all possible interleavings of the enabled signals. A state that is not part of this compacted state is not reached until $a+$ and $b+$ have both fired and $x+$ becomes enabled. The enabled signals are now $x+$, and $d+$ and a compacted state 11RR is created. Note that these two compacted states partially overlap and indeed they need to do so since directed don't care $d+$ is free to fire at any time both before and after $a+$ and $b+$ fire. Subsequently, both compacted states must cover the possibility of $d+$ firing inside either of them. In the compacted state graph, a transition from compacted state RRR0 to compacted state 11RR takes place when both $a+$ and $b+$ have fired (illustrated by the $a+,b+,d*$ label in the figure), regardless if $d+$ has fired or not. Figure 3(d) shows the uncompacted state graph for the same transition specification.

For an extended burst-mode controller specification there exists a one-to-one correspondence between an extended burst-mode burst and a compacted state graph state. The compacted state graph size therefore grows linearly with the number of bursts in the original extended burst-mode specification and is completely independent of the number of concurrently enabled signals and the number of declared level signals. The right side of Figure 3(e) shows the compacted state graph for the *ack-xbm-si* example from Figure 1. The compacted state graph has 17 states and 20 transitions. An uncompacted state graph would have 50 states and 114 transitions. In real, more complex, extended burst-mode controllers, state savings are usually well over two orders of magnitude significantly reducing state graph analysis time and memory consumption.

4.3 Compacted cover tables

The single-cube algorithm starts out by first deriving an initial cover, or trigger cube, for each excitation region consisting of only the trigger signals of that excitation region. Trigger signals in an extended burst-mode controller correspond to the set of terminating edges of an input burst. Note that a signal specified as a directed don't care in a burst cannot be a trigger signal for that burst as its value is a don't care throughout the excitation region. A level signal is also not a trigger signal since the setup time requirement forces it to stabilize before any compulsory edges arrive at the inputs. As an example, consider the compacted state graph in Figure 3(e). The excitation regions for output x are indicated by the shaded (set) and striped (reset) compacted states. The trigger cube for signal x in the set region $abdl.xyz$:1100.RR0 of our example would be $--0-.---$, or d', since $d-$ is the trigger signal transition. The other trigger cubes (TC) for the signal x are shown in Figure 4(a). By starting with a trigger cube to cover each excitation region, we satisfy the second requirement for a hazard-free extended burst-mode cover (see Section 2.2) in that the excitation region is completely covered by some product term. Subsequent steps to remove violations from the initial trigger cube cover must, of course, ensure that the whole excitation region remains completely covered.

Next, a set of potential context signals, signals that are stable throughout the excitation region, are derived for each excitation region. Such context signals are later used to remove violating states. Note that because of the feedback delay constraint of extended burst-mode, concurrently changing outputs that are fed back remain stable at the inputs throughout an excitation region

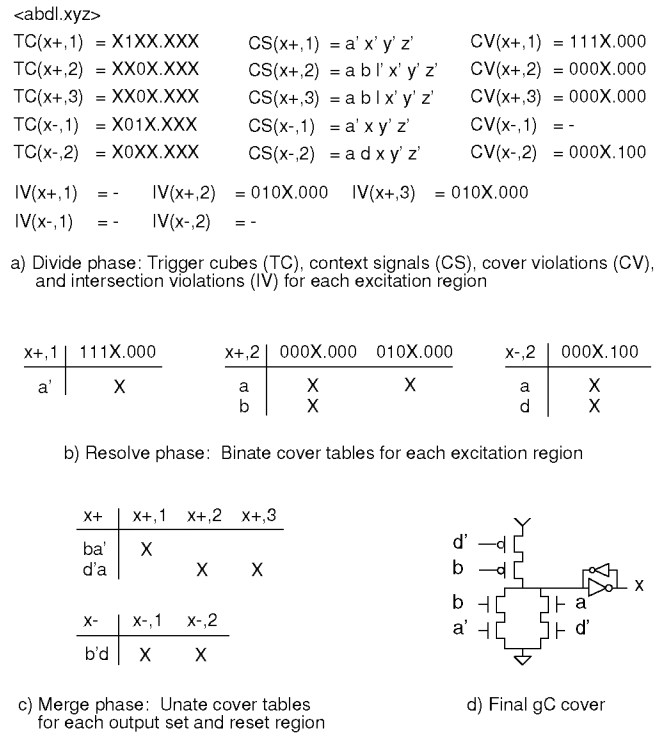

Figure 4. Logic minimization of output x of controller *ack-xbm-si*.

and can thus be used as context signals. The potential context signals for output x in excitation region 1100.RR0 of our example are a, b, l', x', y', and z'. The other potential context signal sets (CS) for excitation regions of signal x are also shown in Figure 4(a).

Since the initial cover may cover states where the output in question is defined to have the opposite value, such *cover violation* (CV) states must be removed from the initial cover. Note that special care must be taken while detecting cover violations when using compacted state graphs since compacted states may overlap. By removing all cover violation states we satisfy the first requirement for a hazard-free extended burst-mode cover in that each set cube does not intersect the off-set minterms.

For extended burst-mode gC controllers, a cube must not intersect an excitation region unless it also includes its end subcube. An example of such an *intersection violation* (IV) is cube d' which intersects excitation region `01R-.R00` without including its end subcube `011-.000`. The intersection state-cube `010-.000` = `01R-.R00` ∩ `--0-.---` must therefore be removed for the cover to be hazard-free. The only possible context signal that can be used to remove this violation is a. All intersection violations (IV) for signal x are shown in Figure 4(a). Note that even if the trigger cube contains the end subcube, a selected context signal may still exclude the end subcube. If such a context signal is selected, then some other context signal must also be selected to remove the entire intersection cube. This condition is what makes our covering problem binate. By removing all intersection violation states (i.e., parts of the trigger cube which may intersect a dynamic transition without containing its end subcube), we satisfy the third requirement for a hazard-free extended burst-mode cover in that no dynamic hazard is present in the final cover.

To minimize the logic output functions, all covering and intersection violations are collected along with any context signal that can remove the violations from the initial cover and a binate cover table is constructed and solved. Columns in the table then represent violations that must be removed before the function becomes hazard-free, and rows represent context signals that can be added to the cover to remove violating states. By using cubes to represent violations in combination with compacted state graphs, the size of the cover table is reduced considerably allowing the cover problem to be solved much faster than previously possible. The binate cover tables for signal x are shown in Figure 4(b).

4.4 Exact minimization

The basic single-cube algorithm implements the per-excitation-region literal-exact minimization approach. It does not perform sharing of product terms which is the minimization criteria of choice for extended burst-mode gC circuits. We present an extension to the single-cube algorithm which essentially casts the problem of finding a per-output literal-exact cover as a simpler three step problem of divide, resolve, and merge. Our proposed method is general enough such that *any* cost function based on literal and product term counts can be easily implemented, both for single and multi-output minimization. This paper, however, considers only single-output literal-exact solutions as this suits extended burst-mode gC implementations best.

Classic single-output logic minimization for extended burst-mode controllers is a two step problem where each output function is considered separately. An output function is composed of all excitation regions for that output's on-set (or off-set). The classic algorithms first find all prime implicants of the output function using tools such as `Espresso` or `Scherzo`. Since prime implicants may potentially be hazardous, the prime implicants are then checked for hazards and split into hazard-free implicants when necessary. A cover is then formed by selecting hazard-free implicants until the entire function is covered. As the number of excitation regions per output grows, Espresso and Scherzo runtimes increase quite notably as deriving prime implicants for a large on-set is a very complex problem. In contrast to the approach described above, the single-cube approach to logic minimization is to consider each excitation region separately, thus dividing the problem into smaller, more easily solved sub-problems. Since we are interested in achieving a minimal literal count per output however, the basic single-cube algorithm must be extended to find a broader set of sub-solutions and then merge these sub-solutions into a final minimal cover.

Finding a per-output literal-exact solution through the use of the single-cube algorithm is achieved by employing a strategy consisting of three phases, divide, resolve, and merge. The divide phase splits the dataset into manageable pieces, the resolve phase then finds the local solutions for each piece, and the merge phase finds the total solution by considering all local solutions. This algorithm is illustrated in Figure 2.

The basic single-cube algorithm first divides the cover problem into the sub-problems of finding, for each excitation region separately, a minimal hazard-free cube that covers the excitation region. To find a minimal cover for the whole output function however, we must extend the single-cube algorithm to return the set of minimal cubes, where each cube covers a unique combination of excitation regions. Finding these sets of local hazard-free solutions for each excitation region then constitutes the resolve phase of the algorithm. The second extension of the single-cube algorithm is then to find a minimal literal cover for the output as a whole from these local solutions. This is posed as a unate cover problem where a minimal literal solution is found using classical reduction techniques followed by a branch and bound on the cyclic core. This forms the merge phase of the single-cube algorithm and produces a final cover for the output that is minimal in the number of literals.

Return to Figure 4(b). Since both a and d can remove the violating state-cube 000-.100 from trigger cube b' of excitation region $(x-,2)$, two minimal literal solutions $b'a$ and $b'd$ are found for this excitation region. Cube $b'a$ covers only excitation region $(x-,2)$ while cube $b'd$ covers both $(x-,1)$ and $(x-,2)$. Since cube $b'a$ covers only a subset of the excitation regions of cube $b'd$ and does not have a smaller literal count, only cube $b'd$ is unique and needs to be kept. Although excitation region $(x+,2)$ has two possible solutions $d'a$ and $d'ab$, only $d'a$ needs to be kept since $d'ab$ is not minimal in number of literals and both solutions cover the same excitation regions (the same cover table and solutions are found for excitation region $(x+,3)$). The merge phase then finds a final solution for the set and reset regions of the output by selecting a minimal cover from the local solutions found during the the resolve phase. This unate cover problem is illustrated in Figure 4(c). Both cubes ba' and $d'a$ must be used in the cover for the set region of x. For the reset region of x only one cube, $b'd$, is needed as it covers both excitation regions. The final complex gate for output x is shown in Figure 4(d).

5 Results

The algorithm described in this paper has been completely incorporated and automated in the ATACS [11] synthesis tool. The ATACS extended burst-mode logic minimizer is exact in number of literals. We compare against the publicly available state of the art hazard-free logic minimization tools developed by Nowick et al. The HFMIN [7] minimizer is exact in number of literals. The IMPYMIN [8] minimizer uses implicit BDD algorithms for exact number of cubes minimization but does not perform literal exact minimization. The ESPRESSO-HF [8] minimizer is heuristic in both number of cubes and number of literals. It should be noted that IMPYMIN and ESPRESSO-HF are optimized for the more difficult problem of multi-output covers. All minimizers are run in single-output minimization configurations as multi-output minimization is not suitable for gC circuits in terms of performance. The 3D tool [19] was used to find a state assignment for the benchmarks.

5.1 Benchmark results

The runtime and literal comparisons shown in Table 1 contains the largest benchmarks that have been built in the burst-mode community to date. The *postoffice* [1], *cache-ctrl* [26], *stetson* [3], *hp-ir* [3], *diffeq* [4], *cd-player* [27], *pscsi* [28], *sscsi* [2], *xscsi* [19], *dram-ctrl* [2], and *barcode* [29] are all derived from real-life designs. The other examples are classic burst-mode benchmarks from various publications. The burst-mode controllers *ack-cd-player, ack-fibonacci, ack-diffeq, ack-barcode, ack-gcd,* and *ack-factorial* are all generated automatically from a procedural language description by the high-level synthesis framework ACK [30, 6, 31, 5]. All benchmarks are run on a 333 MHz Ultrasparc-2 processor with 1 GB of physical memory and 840 MB of virtual memory running the Solaris operating system. All benchmarks that finished ran completely in physical memory. Hence the results should show the true runtime potential of the respective minimizers.

As can be seen in Table 1, our logic minimization method is not only exact for all benchmarks but also extremely fast. Our method achieves sub-second synthesis even for the largest benchmarks and is well over three orders of magnitude faster than the closest literal-exact solution for these benchmarks. Our exact method is also over one order of magnitude faster than heuristic solutions for the majority of the benchmarks. In addition, our method can perform literal-exact minimization for designs where this has previously been impossible. For the *ack-cd-player-p1* benchmark which contains four level signals, HFMIN runs out of memory despite the 1.84 GB of total available memory while our minimizer ATACS completes in only 0.42 seconds using less than 15 MB of memory. Although the heuristic and cube-exact methods of ESPRESSO-HF and IMPYMIN yield results comparable to literal-exact solutions for many of the benchmarks, their literal counts deviate by as much as 13.5% and 10.5% respectively for some of the benchmarks, underscoring the importance of the ability to perform literal-exact exploration of design alternatives.

			ATACS (literal-exact)		HFMIN (literal-exact)		IMPYMIN (cube-exact)		ESPRESSO-HF (heuristic)	
Design	CS	IO	Time	Lit	Time	Lit	Time	Lit	Time	Lit
ack-cdplayer-p1	74	45	0.42	274	impossible		150.86	276	20.02	277
ack-cdplayer-p2	32	24	0.32	129	44.93	129	29.04	133	10.56	130
ack-fibonacci	50	35	0.34	194	1675.65	194	85.74	204	13.61	198
ack-diffeq	32	34	0.33	177	120.16	177	45.93	184	14.32	185
ack-barcode	34	28	0.31	167	45.90	167	31.54	168	11.15	168
ack-gcd	34	20	0.33	80	19.26	80	16.44	80	6.48	80
ack-factoriai	23	17	0.27	42	16.01	42	13.26	42	5.77	42
ack-xbm-si	17	7	0.25	16	7.68	16	4.48	16	2.09	16
cache-ctrl	98	36	0.79	510	2524.96	510	273.40	534	14.62	521
chu-ad-opt-e	8	6	0.25	12	5.53	12	4.68	12	2.19	12
dme-e	20	8	0.28	22	9.33	22	8.50	22	3.69	22
dme-fast-e	20	8	0.26	27	9.27	27	7.74	29	3.66	28
dram-ctrl	25	13	0.31	38	12.93	38	10.25	42	4.37	42
hp-ir-sc-ctrl	76	30	0.39	205	81.73	205	41.90	216	12.26	213
hp-ir-sd-ctri	54	24	0.50	137	39.42	137	30.24	144	11.28	144
hp-ir-it-ctri	23	13	0.26	46	15.25	46	12.85	48	5.73	47
hp-ir-rf-ctrl	26	13	0.31	33	13.18	33	10.92	35	5.06	35
hp-ir-two-tick	15	6	0.25	8	3.82	8	3.10	8	1.59	8
hp-ir	16	5	0.26	11	3.74	11	3.09	11	1.53	11
postoffice-pesnd	28	10	0.31	57	9.33	57	8.01	58	3.64	58
postoffice-bfsnd	17	8	0.33	30	9.22	30	7.61	31	3.55	31
postoffice-bfrd	14	7	0.26	15	7.32	15	6.03	15	2.88	15
pscsi-pscsi	119	20	0.58	270	30.76	270	22.49	282	7.42	280
pscsi-isend	22	10	0.31	48	11.09	48	9.24	48	4.25	48
pscsi-itcv	13	9	0.32	26	9.15	26	7.56	26	3.56	26
pscsi-trcv-bm	18	10	0.27	30	11.06	30	10.01	30	4.26	30
pscsi-trcv	14	8	0.32	23	7.33	23	6.09	23	2.83	23
pscsi-tsend-bm	24	10	0.33	45	11.26	45	9.22	45	4.21	45
pscsi-tsend	24	10	0.32	50	11.14	50	9.23	50	4.26	50
sscsi-isend-bm	24	11	0.31	52	11.39	52	9.35	52	4.24	52
sscsi-isend-csm	18	11	0.28	39	11.05	39	9.28	41	4.31	39
sscsi-trcv-bm	24	11	0.27	50	11.14	50	9.39	51	4.25	51
sscsi-trcv-csm	18	11	0.26	37	11.04	37	9.28	39	4.23	37
sscsi-tsend-bm	26	11	0.30	52	11.92	52	9.32	52	4.27	52
sscsi-tsend-csm	22	11	0.30	38	11.10	38	9.27	39	4.22	40
stetson-p1	84	30	0.43	225	81.06	225	39.80	236	11.95	235
stetson-p2	56	24	0.52	148	38.41	148	28.47	154	11.10	168
stetson-p3	16	6	0.25	8	3.74	8	3.03	8	1.53	8
vanbek-ad-opt-e	6	6	0.25	13	5.47	13	4.61	13	2.17	13
xscsi-fifo2scsi	22	11	0.32	45	12.02	45	9.38	46	4.12	45
xscsi-dma2fifo	18	9	0.28	37	9.53	37	7.62	37	3.50	37
xscsi-fifo2dma	14	8	0.29	20	7.87	20	6.01	20	2.78	20
xscsi-fifoceilcti	6	5	0.25	10	5.48	10	4.51	10	2.10	10
yun-diffeq-aiu2	32	14	0.32	89	20.02	89	16.21	90	6.19	89
yun-diffeq-aiu1	18	10	0.29	38	13.28	38	10.61	38	4.67	38
yun-diffeq-mul1	8	7	0.26	32	7.43	32	6.04	32	2.73	32
yun-diffeq-mul2	6	6	0.25	13	5.74	13	4.46	13	2.06	13

Table 1. Benchmark comparisons between existing extended burst-mode logic minimizers ATACS (the method presented in this paper), HFMIN, IMPYMIN, and ESPRESSO-HF. (*CS* - number of compacted states in state graph, *IO* - number of input, output, and state signals, *Time* - logic minimization run-time in seconds, *Lit* - number of literals in solution.)

5.2 Discussion

The effectiveness of the presented logic minimizer can be mainly attributed to two factors. First, the signal concurrency properties of extended burst-mode controllers allows them to be expressed very efficiently in the form of compacted state graphs. Using compacted state graphs to represent extended burst-mode finite state machines, time spent in state graph exploration grows linearly, rather than exponentially, with the complexity (amount of signal concurrency and number of level signals) of the specification. The notion of compacted states is also exploited to significantly reduce the time spent in solving the binate and unate cover

problems necessary to find hazard-free minimal solutions, as the size of the cover tables is significantly reduced.

Second, the presented per-output single-cube minimization algorithm naturally divides the problem of finding a solution into smaller sub-problems of finding local unique solutions for each excitation region separately, which are then merged into a final minimal solution for the entire output function. This divide and merge strategy works especially well for gC controllers due to the relatively few required cubes (excitation regions), and subsequently local solutions, present in extended burst-mode controllers. The number of local solutions for a given output function is the limiting factor of the unate cover problem in the merge step of the extended single cube algorithm, and presently also the limiting factor for the whole minimization algorithm.

6 Conclusions

A very fast algorithm for per-output literal-exact logic minimization of extended burst-mode gC finite state machines is presented. The method is based on state graph exploration and the efficient single-cube cover algorithm which is extended to handle the extended burst-mode hazard model and generate per-output minimized covers. Synthesis time for very large controllers has been significantly reduced by introducing the concept of compacted state graphs and compacted binate cover tables. New extensions to the original single-cube algorithm allowing extremely fast generation of per-output literal-exact solutions have also been developed. With the achievement of sub-second exact logic minimization of even the largest burst-mode benchmarks to date on a 333 MHz microprocessor, the method presented in this paper has opened up the possibility of truly interactive and iterative design space exploration of exact solutions for large controllers.

Acknowledgement: The authors would like to thank Michael Theobald for his invaluable help with benchmarks and tools. Also thanks to Prabhakar Kudva and Eric Mercer for helpful comments and suggestions.

References

[1] Bill Coates, Al Davis, and Ken Stevens, "The Post Office experience: Designing a large asynchronous chip," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 341–366, Oct. 1993.

[2] Steven M. Nowick, Kenneth Y. Yun, and David L. Dill, "Practical asynchronous controller design," in *Proc. International Conf. Computer Design (ICCD)*. Oct. 1992, pp. 341–345, IEEE Computer Society Press.

[3] Alan Marshall, Bill Coates, and Polly Siegel, "Designing an asynchronous communications chip," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 8–21, 1994.

[4] Kenneth Y. Yun, Peter A. Beerel, Vida Vakilotojar, Ayoob E. Dooply, and Julio Arceo, "The design and verification of a high-performance low-control-overhead asynchronous differential equation solver," *IEEE Transactions on VLSI Systems*, vol. 6, no. 4, pp. 643–655, Dec. 1998.

[5] Hans M. Jacobson and Ganesh Gopalakrishnan, "Application-specific programmable control for high-performance asynchronous circuits," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 319–331, Feb. 1999.

[6] Prabhakar Kudva, Ganesh Gopalakrishnan, and Hans Jacobson, "A technique for synthesizing distributed burst-mode circuits," in *Proc. ACM/IEEE Design Automation Conference*, 1996.

[7] Robert M. Fuhrer, *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines*, Ph.D. thesis, Department of Computer Science, Columbia University, 1999.

[8] Michael Theobald and Steven M. Nowick, "Fast heuristic and exact algorithms for two-level hazard-free logic minimization," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 11, pp. 1130–1147, Nov. 1998.

[9] J. Cortadella, M. Kishinevsky, A.Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.

[10] Chantal Ykman-Couvreur, Bill Lin, and Hugo de Man, "Assassin: A synthesis system for asynchronous control circuits," Tech. Rep., IMEC, Sept. 1994, User and Tutorial manual.

[11] Chris J. Myers, *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*, Ph.D. thesis, Dept. of Elec. Eng., Stanford University, Oct. 1995.

[12] Kenneth Y. Yun and David L. Dill, "Automatic synthesis of extended burst-mode circuits: Part I (specification and hazard-free implementation)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 101–117, Feb. 1999.

[13] Kuan-Jen Lin, Chi-Wen Kuo, and Chen-Shang Lin, "Synthesis of hazard-free asynchronous circuits based on characteristic graph," *IEEE Transactions on Computers*, vol. 46, no. 11, pp. 1246–1263, Nov. 1997.

[14] Enric Pastor, Jordi Cortadella, Alex Kondratyev, and Oriol Roig, "Structural methods for the synthesis of speed-independent circuits," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 11, pp. 1108–1129, Nov. 1998.

[15] Sung Tae Jung and Chris J. Myers, "Direct synthesis of timed asynchronous circuits," in *Proc. International Conf Computer-Aided Design (ICCAD)*, Nov. 1999, pp. 332–337.

[16] Robert M. Fuhrer and Steven M. Nowick, "OPTIMISTA: State minimization of asynchronous FSMs for optimum output logic," in *Proc. International Conf Computer-Aided Design (ICCAD)*, 1999.

[17] Shai Rotem, Ken Stevens, Ran Ginosar, Peter Beerel, Chris Myers, Kenneth Yun, Rakefet Kol, Charles Dike, Marly Roncken, and Boris Agapiev, "RAPPID: An asynchronous instruction length decoder," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1999, pp. 60–70.

[18] Kenneth Y. Yun, "Automatic synthesis of extended burst-mode circuits using generalized C-elements," in *Proc. European Design Automation Conference (EURO-DAC)*, Sept. 1996, pp. 290–295.

[19] Kenneth Yi Yun, *Synthesis of Asynchronous Controllers for Heterogeneous Systems*, Ph.D. thesis, Stanford University, Aug. 1994.

[20] K. W. James and K. Y. Yun, "Average-case optimized transistor-level technology mapping of extended burst-mode circuits," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 70–79.

[21] P. Beerel and T.H.-Y. Meng, "Automatic gate-level synthesis of speed-independent circuits," in *Proc. International Conf Computer-Aided Design (ICCAD)*. Nov. 1992, pp. 581–587, IEEE Computer Society Press.

[22] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng, "Covering conditions and algorithms for the synthesis of speed-independent circuits," *IEEE Transactions on Computer-Aided Design*, Mar. 1998.

[23] Alex Kondratyev, Michael Kishinevsky, Bill Lin, Peter Vanbekbergen, and Alex Yakovlev, "Basic gate implementation of speed-independent circuits," in *Proc. ACM/IEEE Design Automation Conference*, June 1994, pp. 56–62.

[24] Alex Kondratyev, Michael Kishinevsky, and Alex Yakovlev, "Hazard-free implementation of speed-independent circuits," *IEEE Transactions on Computer-Aided Design*, vol. 17, no. 9, pp. 749–771, Sept. 1998.

[25] Kenneth Y. Yun and David L. Dill, "Automatic synthesis of extended burst-mode circuits: Part II (automatic synthesis)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 118–132, Feb. 1999.

[26] Steven M. Nowick, Mark E. Dean, David L. Dill, and Mark Horowitz, "The design of a high-performance cache controller: a case study in asynchronous synthesis," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 241–262, Oct. 1993.

[27] Joep Kessels, Kees van Berkel, Ronan Burgess, Marly Roncken, and Frits Schalij, "An error decoder for the compact disc player as an example of VLSI programming," Tech. Rep., Philips Research Laboratories, Eindhoven, The Netherlands, 1992.

[28] Kenneth Y. Yun and David L. Dill, "Automatic synthesis of 3D asynchronous state machines," in *Proc. International Conf Computer-Aided Design (ICCAD)*. Nov. 1992, pp. 576–580, IEEE Computer Society Press.

[29] P R Panda and N Dutt, "1995 high level synthesis design repository," Tech. Rep. 95-04, University of California, Irvine, U.S.A., 1995.

[30] Prabhakar Kudva, *Synthesis of Asynchronous Systems Targeting Finite State Machines*, Ph.D. thesis, Computer Science Department, University of Utah, 1995.

[31] Prabhakar Kudva, Ganesh Gopalakrishnan, Hans Jacobson, and Steven M. Nowick, "Synthesis of hazard-free customized CMOS complex-gate networks under multiple-input changes," in *Proc. ACM/IEEE Design Automation Conference*, 1996.