Speed-Independent PRs			Simplified Timed PRs		
$lsro \wedge lsri$	$\mapsto$	$mdlo^+$	$lsri$	$\mapsto$	$mdlo^+$
$\neg lsro \wedge \neg mdli$	$\mapsto$	$mdlo^-$	$\neg mdli$	$\mapsto$	$mdlo^-$
$\neg mdlo \wedge \neg lsro \wedge \neg rai \wedge mdli$	$\mapsto$	rao^+	$\neg mdlo \wedge \neg lsro \wedge mdli$	$\mapsto$	rao^+, bo^+
$lsro$	$\mapsto$	rao^-	$lsro$	$\mapsto$	rao^-, bo^-
$\neg mdlo \wedge \neg lsro \wedge \neg bi \wedge mdli$	$\mapsto$	bo^+			
$lsro$	$\mapsto$	bo^-			
$rao \wedge bo \wedge \neg lsri \wedge rai \wedge bi$	$\mapsto$	$lsro^+$	$rai \wedge bi$	$\mapsto$	$lsro^+$
$\neg rao \wedge \neg bo \wedge mdlo$	$\mapsto$	$lsro^-$	$mdlo$	$\mapsto$	$lsro^-$

Table 1: Comparision of production rules for speed-independent and timed versions of the MMU.

Examples Optimized with Timing				
	Speed-Independent		Timed	
Example	Trigger Signals	Context Signals	Trigger Signals	Context Signals
Pipeline Handshake [12]	8	0	6	0
MMU unopt [6]	13	3	9	2
MMU opt	15	7	9/7	4/2
Microprocessor [7] [3]				
Fetch	8	sv	6	6
PCAdd	10	sv	7	4
Exec	42	sv	26	4
ALU	26	sv	17	sv

Table 2: An entry of "sv" under context rules indicates that a state variable is needed for synthesis. For the timed implementation of the optimized MMU, the first number is without sharing gates for *rao* and *bo*, and the second number is with the gate shared.

[8] Alain J. Martin. "Programming in VLSI: From Communicating Processes to Delay-Insensitive VLSI Circuits". In C.A.R. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1990.

[9] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, Massachusetts Institute of Technology, 1987.

[10] Alain J. Martin. "Formal Program Transformations for VLSI Circuit Synthesis". In E.W. Dijkstra, editor, *UT Year of Programming Institute on Formal Developments of Programs and Proofs*. Addison-Wesley, 1989.

[11] Dražen Borković. *Private Communications*, Summer 1991. Dražen Borković is a graduate student at Caltech.

[12] Andy Hung and Teresa H. Y. Meng. "Asynchronous Self-timed Circuit Synthesis with Timing Constraints". In *Proceedings of the IEEE ISCAS 90*, May 1990.

Q	:	$mdli$	$mdlo$	$lsri$	$lsro$
Enabled state					
$mdlo^+$	:	1	X	1	1
$mdlo^-$	:	0	X	X	0
Timed enabled state					
$mdlo^+$	:	1	X	1	1
$mdlo^-$	:	0	X	0	0

As we can see above, the uncertainty in the value of $lsri$ has been removed when timing is incorporated. From the specification given in figure 3b, we see that the transition $mdlo^+$ has only one rule enabling it from $lsri^+$ which thus cannot be redundant. Using just this signal to pull $mdlo$ up may cause a problem in the original enabled state, because when $mdlo$ is being pulled down, $lsri$ is in an indeterminate state. Therefore, interference between the pullup and the pulldown of the signal may be possible. This problem can be alleviated by using either the signal $mdli$ or $lsro$ as a context signal since they are definitely disabled when $mdlo$ is pulled down. If we look at the timed enabled state, however, we see that $lsri$ has indeed gone low before we want to pull $mdlo$ down. Therefore, taking the timing constraints into consideration, we do not need to add an extra context signal for this transition, simplifying the implementation. Using this analysis, we can remove 3 of the 7 context signals needed in the speed-independent implementation for our timed implementation.

4.3 Speed-Independent vs. Timed Implementation

After adding context signals to our original specification, we require 22 literals for a speed-independent implementation as shown in table 1. Given the timing constraints, we can reduce the circuit to only 9 literals. Thus, we have reduced our circuit complexity from the speed-independent implementation by over 50 percent using conservative timing constraints. A complex gate implementation for both is shown in figure 4. Note that this reduction was possible not only because of removing redundant literals, but also because we could combine the gate needed for implementing rao and bo.

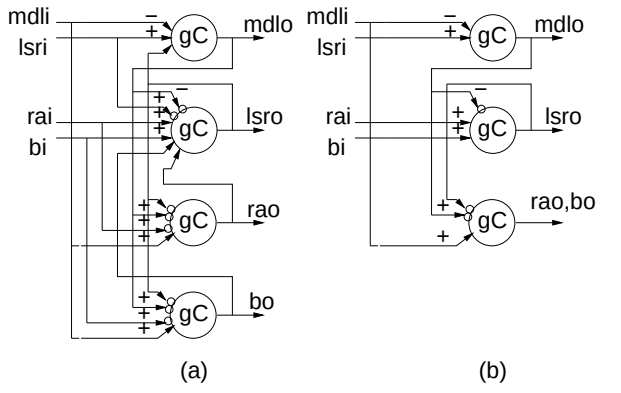

Figure 4: (a) The speed-independent implementation of the MMU. (b) The timed implementation of the MMU.

4.4 Other Examples

We have also synthesized many other circuit examples as shown in table 2. In the pipeline handshake example, we used the timing constraints given in [12] which are shown in figure 1. In this example, we notice that we can remove two trigger signals. For the MMU, we see that the version optimized for concurrency has a more complex speed-independent implementation, but a less complex timed implementation. The specifications for the microprocessor described in [7] are taken from [3]. For these circuits, we used similar timing assumptions as those for the MMU. All of the microprocessor specifications needed state variables for a speed-independent implementation; however, three of the four could be implemented without state variables if timing constraints were added.

5 Conclusions and Future Research

We have developed a synthesis procedure for timed circuits based on a timing analysis on infinite acyclic graphs. We show that a sufficient condition for the removal of redundancy in a specification can be found by examining a finite portion of the graph. To implement a timed circuit without circuit hazards under given timing constraints, we apply the timing analysis to optimizing a circuit implementation. Our results indicate that by using conservative timing constraints, we can significantly reduce a circuit's complexity. While it is obvious that area is reduced, we also have an increase in performance because we are able to synthesize more concurrent specifications than can often be considered practical using speed-independent design techniques.

Future work includes the investigation of the necessary condition for removal of redundant rules. We believe that all legal redundant rules should be detectable by tracing only a finite portion of an unfolded cyclic graph. Other future goals include automation of the timing analysis and the synthesis procedure, and the IC design of interesting timed circuits to assess the area and performance gain.

Acknowledgments

The authors would like to thank Professor David Dill of Stanford University who dedicated a considerable amount of time in assisting us in formalizing our work. The authors would also like to thank Peter Beerel of Stanford University and Professor Steve Burns of the University of Washington for many valuable discussions on the synthesis of timed circuits. We would also like to express our appreciation of the work done in Professor Alain Martin's group at the California Institute of Technology, for their insight in the design of asynchronous circuits. Their assistance in deriving the specification and speed-independent implementation of the MMU example is also gratefully acknowledged.

References

[1] P. Vanbekbergen, G. Goossens, and H. De Man. "Specification and Analysis of Timing Constraints in Signal Transition Graphs", 1992. Accepted to EDAC.

[2] Kenneth McMillan and David L. Dill. "Algorithms for Interface Timing Verification", 1992. To appear in ICCD 92.

[3] Steve Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.

[4] Gaetano Borriello and Randy H. Katz. "Synthesis and Optimization of Interface Transducer Logic". In *Proceedings IEEE 1987 ICCAD* Digest of Papers, pages 274–277, 1987.

[5] L. Lavagno, K. Keutzers, and A. Sangiovanni-Vincentelli. "Synthesis of Verifiably Hazard-Free Asynchronous Control Circuits". Technical report, University of California, Berkeley, 1990.

[6] Chris Myers and Alain Martin. "An Asynchronous Memory Management Unit". To appear in a *Caltech* technical report.

[7] A. J. Martin, S. M. Burns, T. K. Lee, D. Borković, and P. J. Hazewindus. "The Design of an Asynchronous Microprocessor". In *Decennial Caltech Conference on VLSI*, pages 226–234, 1989.

$q_{s+,k}$ is determined as follows: for $k = 1, 2, \ldots, n$, if while the transition s^+ is enabled, the signal s_k is stable low, then $q_{s+,k} \equiv 0$; if the signal s_k is stable high, then $q_{s+,k} \equiv 1$; and if the signal s_k is changing, then $q_{s+,k} \equiv X$. The enabled state for the transition s^- is similarly defined.

An algorithm to derive the enabled state and to determine a minimal set of *context signals* which need to be added into the circuit implementation to prevent the problems described above has been developed [11]. Using our timing analysis, we can remove many of these context signals. After we detect the redundant rules, we construct an enabled state from the original specification. We then remove much of the uncertainty, or X's, from the enabled state by checking the covering of the paths between events in a similar fashion as in the previous section. Using this *timed enabled state* and the remaining non-redundant rules, we determine a minimal set of context signals. The set usually contains fewer context signals than that needed in a speed-independent implementation.

3.2 Complex Gate Circuit Implementation

For each output signal, s, the *trigger* signals (i.e. those given in the rules) and the context signals for s^+ are implemented in series in a pullup network, and similarly, the signals needed for s^- are implemented in series in a pulldown network. The resulting circuit is a state-holding element called a *generalized C-element* [8], and some examples are shown in figure 2a with the rules that they implement. If a signal appears only in the pullup, but not in the pulldown, then it is annotated with a "+". If a signal appears only in the pulldown then it is annotated with a "-". Otherwise, the signal has no annotation. A CMOS implementation for the third element is shown in figure 2b.

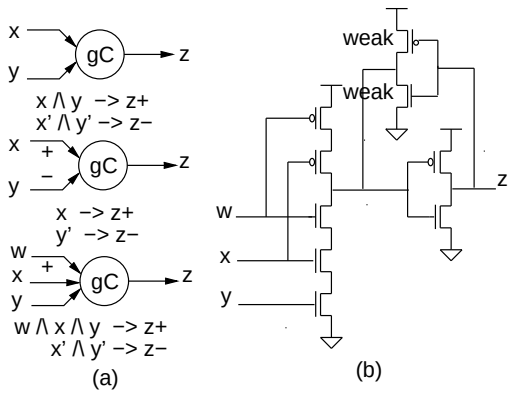

Figure 2: (a) Schematic symbols for a generalized C-element. The first is the Muller C-element; the second is a set/reset flip-flop; and the implementation of the third is given in (b) (Courtesy of [3]).

The degree of complexity reduction in a timed circuit compared to a speed-independent one is dependent on the amount of concurrency in the specification. While this reduction of circuit complexity does not reduce the number of gate transitions in the critical path (because we use a complex-gate approach for implementation), increase in speed can be realized in that a complex gate with smaller numbers of fanin and fanout will switch faster. We also believe performance will be enhanced due to the fact that timed circuits allow more concurrency to be specified.

4 Example: A Memory Management Unit

In this section, we present an example of a simple memory management unit (MMU) designed for use with a 16-bit asynchronous microprocessor [7]. The original implementation was derived using Martin's synthesis method [6]. A version of this specification which has been optimized for concurrency will be presented here. There are six possible cycles that the MMU control circuit can enter, depending on the whims of the environment. The 6 cycles can be independently designed and merged together to get the overall implementation. For simplicity, we discuss only the design of one cycle: loading from the segmentation read register, which is shown in figure 3. For our timed implementation, we have taken all the internal timing constraints to be negligible and the environment timing constraints to be large. We also assume that all signals are initially reset low. A solid dot on an arc indicates the initial marking for which the corresponding rule has $\varepsilon = 1$.

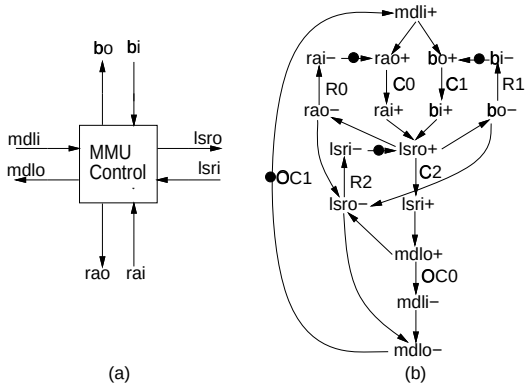

Figure 3: (a) Block diagram for part of the MMU controller. (b) The STG specification for the MMU with timing constraints added. The OCi, Ci, and Ri's on the graph indicate *bounded timing constraints* from off-chip, computation, and reseting a computation, respectively.

4.1 Removing Redundant Rules

To generate our timed implementation, we must first check all events associated with multiple rules (i.e. all transitions with multiple arcs leading into them) for redundant rules given the specified timing constraints. We make the following delay assumptions: off-chip delays are greater than computational delays ($l(OC_i) > u(C_i)$), computational delays are greater than delays from reseting a computation ($l(C_i) > u(R_i)$), and delays from reseting a computation are much greater than internal delays ($l(R_i) \gg \ internal\ delays$). Let us examine the rules on the event $\langle rao^+, i\rangle$ from the events $\langle rai^-, i-1\rangle$ and $\langle mdli^+, i\rangle$. We see that the path $\langle\langle rao^-, i-1\rangle, \langle rai^-, i-1\rangle, \langle rao^+, i\rangle\rangle$ has length of R_0 and the path $\langle\langle rao^-, i-1\rangle, \langle lsro^-, i-1\rangle, \langle mdlo^-, i-1\rangle, \langle mdli^+, i\rangle, \langle rao^+, i\rangle\rangle$ has length OC_1. Therefore, the second path covers the first so all paths with a tail $\langle\langle rai^-, i-1\rangle, \langle rao^+, i\rangle\rangle$ are not critical. Since the timing on event $\langle rao^+, 0\rangle$ is trivially satisfied because the signal rai is initially low, the rule from rai^- to rao^+ is redundant. This procedure is repeated for all other events with multiple rules, and we find that 6 out of the 15 rules on output signals in the original specification are redundant.

4.2 Adding Context Signals

To determine which context signals must be added, we must first determine the timed enabled state for each signal using the delay assumptions. Part of the resulting enabled state with no timing constraints and the timed enabled state for the signal $mdlo$ are shown below:

Proof: Since a rule is to specify that one event occurs before another, we need to show that the rule is not needed to prevent the enabled event from firing too early (from Definition 2.1). Given a redundant rule $\langle e, f, i, \varepsilon, \tau \rangle$, let P be the set of all paths between the two events $\langle reset, 0 \rangle$ and $\langle f, i \rangle$. Let P_r be the set of all paths with the tail $\langle \langle e, i-\varepsilon \rangle, \langle f, i \rangle \rangle$. All paths in the set P_r are not critical (from Definition 2.8). If we remove the redundant rule, we also remove all paths P_r from P. Since they are not critical, their removal does not change the earlist firing time of $\langle f, i \rangle$ after $\langle reset, 0 \rangle$; i.e. $max[l(P - P_r)] = max[l(P)]$ (from Definition 2.7). ∎

In Lemma 2.2, we have shown that it is possible to simplify our timed circuits by detecting and removing redundant rules. To check if a rule is redundant, we must show that it never appears on a critical path.

2.3 Algorithm for Detecting Redundant Rules

In our ER systems, each rule will appear an infinite number of times; however, it is not necessary to check each occurrence of each rule to verify that all are not on a critical path due to the repetitive nature of the ER system. In the following algorithms, we provide a sufficient condition under which only two occurrences of each rule need to be checked (proven in Theorem 2.1). Lemma 2.1 further simplifies our analysis in that we need not examine the whole graph to show that a path is not critical since it is sufficient to show this by finding a tail of this path which is covered.

Algorithm 2.1 *On_Critical_Path (G, source, $\langle e_k, f, i, \varepsilon_k, \tau_k \rangle$)*
For each path which starts with the event $\langle source, i-1 \rangle$ and ends with the rule $\langle e_k, f, i, \varepsilon_k, \tau_k \rangle$ or $\langle source, f, i, 0, [0, \infty] \rangle$
If there exists a tail of this path covered by a path that ends in a different rule $\langle e_j, f, i, \varepsilon_j, \tau_j \rangle$ where $e_k \neq e_j$
then return(false)
else return(true)

Algorithm 2.2 *Find_Redundant_Rules*
Given an ER system
Construct graph G excluding reset rules which includes only events with indicies $i-1$ or i for an arbitrary $i > 0$
Add an event $\langle source, i-1 \rangle$ to set of vertices
For each rule of the form $\langle e, f, i-1, 1, \tau \rangle$ (i.e. $\varepsilon = 1$)
Add a rule of the form $\langle source, f, i-1, 0, [0, \infty] \rangle$
For each event $\langle f, i \rangle$ with multiple rules
For each rule of the form $\langle e_k, f, i, \varepsilon_k, \tau_k \rangle$
If not (On_Critical_Path (G, source, $\langle e_k, f, i, \varepsilon_k, \tau_k \rangle$)))
then construct a graph G' including reset rules which includes only events with $i = 0$
If not (On_Critical_Path (G', reset, $\langle e_k, f, 0, \varepsilon_k, \tau_k \rangle$)))
then add rule to set of redundant rules

Theorem 2.1 *Algorithm 2.2 is sufficient to determine if a rule can be removed.*

Proof: (by induction) Base case: Algorithm 2.2 explicitly checks that for $i = 0$ the rule is not on a critical path. This is done by using Algorithm 2.1 which checks that there is some tail of this path which is covered by another path (from Lemma 2.1). Inductive step: Given that the algorithm also determines if some arbitrary value of i is not on a critical path, we construct a graph that includes only occurrence indices of i and $i+1$. This graph would have the same form as G from Algorithm 2.2. Since we found that the i^{th} occurrence was not on a critical path when examining G, this later occurrence will also be found not to appear on a critical path. Since no occurrence of a rule appears on a critical path, the rule is redundant and can be removed (from Definition 2.8 and Lemma 2.2). ∎

To identify redundant rules, we use Algorithm 2.2 to construct a finite acyclic graph and use Algorithm 2.1 to detect when an occurrence of a rule is not on a critical path. Each time Algorithm 2.1 is called, if we are given that $\langle e_k, i - \varepsilon_k \rangle$ is one of the enabling events of $\langle f, i \rangle$, we use the polynomial time algorithms described in [1] or [2] to find the minimum time difference between $\langle e_k, i - \varepsilon_k \rangle$ and $\langle f, i \rangle$. If it is greater than u_k, where u_k is the upper bound on the timing constraint of the rule between these events, then the rule between these events cannot be on a critical path.

We also propose an alternative algorithm which detects redundant rules by looking for covered paths as described in Algorithm 2.1. This algorithm we will outline with an example. Referring back to figure 1, we first construct the graph G as described in Algorithm 2.2. We will analyze the enabling rules for the event $\langle Rout^-, i \rangle$: $\langle Aout^-, Rout^-, i, 0, [2, 5] \rangle$ and $\langle Ain^+, Rout^-, i, 0, [5, 7] \rangle$. Earlier, it was shown that the path $\langle \langle Rout^+, i \rangle, \langle Aout^-, i \rangle, \langle Rout^-, i \rangle \rangle$ was covered. Therefore, from Lemma 2.1, all paths with this path as a tail, such as $\langle \langle Aout^+, i \rangle, \langle Rout+, i \rangle, \langle Aout^-, i \rangle, \langle Rout^-, i \rangle \rangle$, are not critical. There are other paths, however, that do not have this path as a tail but have the path $\langle \langle Aout^+, i \rangle, \langle Rin^-, i \rangle, \langle Aout^-, i \rangle, \langle Rout^-, i \rangle \rangle$ as a tail. This path has a maximum length of 40, so it can be covered by the path $\langle \langle Aout^+, i \rangle, \langle Rout^+, i \rangle, \langle Ain^+, i \rangle, \langle Rout^-, i \rangle \rangle$ which has a minimum length of 1010. Therefore, all paths with a tail $\langle \langle Aout^-, i \rangle, \langle Rout^-, i \rangle \rangle$ with $i > 0$ are not critical (from Theorem 2.1). To show that all paths with $\langle \langle Aout^-, 0 \rangle, \langle Rout^-, 0 \rangle \rangle$ as a tail are not critical, we construct a graph G' by removing any events with occurrence index $i - 1$, and add the reset rules. In this case, we did not rely on any events with an occurrence index $i - 1$ in our analysis so the same covering relations hold. Therefore, all occurrences of the rule $\langle Aout^-, Rout^-, i, 0, [2, 5] \rangle$ are not on a critical path which means the rule is redundant and can be removed.

2.4 Complexity of the Algorithm

Although the condition used in Algorithm 2.2 is only sufficient and not necessary, meaning that it may be possible that not all redundant rules are found, in all the examples that we have synthesized using this algorithm, we were able to identify all the redundant rules. Therefore, given a cyclic specification with v vertices and e edges, we only need to analyze two acyclic graphs, one with $2v$ vertices and $2e$ edges and one with v vertices and e edges.

3 Synthesis of Timed Circuits

Once we have determined the redundant rules, we can then use this information to synthesize a timed circuit. This section describes a procedure for doing so which utilizes timing constraints for further reductions.

3.1 Adding Context Signals

Before we can generate a timed circuit, we must determine which signals need to be added to prevent transitions from *interfering* or *misfiring*. Interference occurs when a circuit is trying to both pullup and pulldown a signal simultaneously. A misfire, on the other hand, is when a signal that is enabled to be low (or high) is pulled up (or down) prematurely. One method often used to prevent these problems from occurring is to determine the state of all the signals when we want a transition to occur. To this state information, Boolean minimization is applied to derive an implementation. We chose to use another method, *guard strengthening*, developed by Martin [10] and implemented by Borković [11], because we can add a timing analysis to it to achieve even further complexity reduction. This algorithm uses information we call the *enabled state* described below:

Definition 3.1 *(Enabled State) For each transition s^+, the enabled state is of the form $Q_{s+} \equiv q_{s+,1}, \ldots, q_{s+,k}, \ldots, q_{s+,n}$, where n is the number of signals in the specification. Each*

For each rule in which $\varepsilon = 1$, we must add a special reset rule of the form $\langle reset, f, 0, 0, \tau_0 \rangle$ where the event $\langle reset, 0 \rangle$ is the initialization event for the system. This will allow us to specify the timing on initial occurrences of events.

The last parameter to consider is the bounded timing constraint τ. Timing constraints for an input rule (i.e., those with the enabling event from the environment) are usually determined by interface specifications or datapath delay estimates. Timing constraints for an internal event-rule can be estimated using standard procedures based on fanin and fanout. After a circuit is generated, more accurate circuit models can be used to deduce a more accurate estimate. If this new estimate is outside the range of the previous estimate, then the synthesis procedure requires rerunning with the revised timing constraints. In many applications, internal delays are negligible compared to the long environment delays, thus allowing us to avoid the problem of calculating the internal delays which are usually very sensitive to our optimizations.

A circuit example specified by its STG and part of its corresponding ER system are shown in figure 1. The solid dots in figure 1a indicate the initial marking; i.e., these rules will have $\varepsilon = 1$. The numbers on an arc represent the lower and upper bounds of the timing constraint on the enabled transition. This cyclic graph is unfolded into an infinite acyclic graph, or an ER system, as shown in figure 1b. An example of a rule in this ER system is between the two events $\langle aout^-, i-1 \rangle$ and $\langle rin^+, i \rangle$ which is of the form $\langle aout^-, rin^+, i, 1, [1000, *] \rangle$, where * represents an infinite delay.

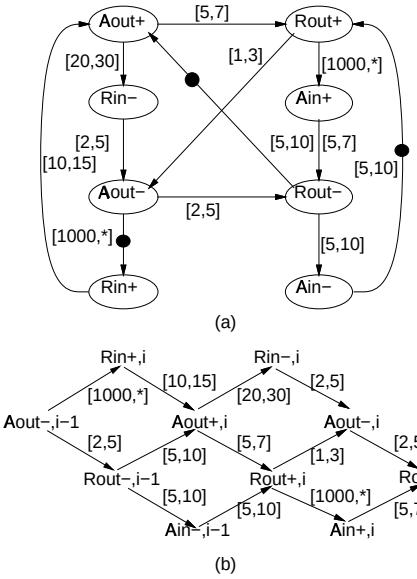

Figure 1: (a) The STG for an interconnection circuit (courtesy of [12]). (b) A subgraph of the infinite ER system for this STG.

2.2 Sufficient Condition for Rule Removal

Before we synthesize a timed circuit, we first detect which rules are *redundant*. A rule is redundant if it is implied by the timing constraints. In other words, a redundant rule can be omitted without changing the set of possible interleavings of events. Only when there are multiple enabling rules for an enabled event can some of the enabling rules be redundant. In this section, we will give the formalism needed to determine when a rule is redundant and to prove that this is a sufficient condition to remove a rule.

Definition 2.2 *(Path) A path p is an ordered sequence of events $\langle e_0, i_0 \rangle$, $\langle e_1, i_1 \rangle$, ..., $\langle e_n, i_n \rangle$ and their corresponding rules $\langle e_0, e_1, i_1, i_1 - i_0, \tau_1 \rangle$, $\langle e_1, e_2, i_2, i_2 - i_1, \tau_2 \rangle$, ..., $\langle e_{n-1}, e_n, i_n, i_n - i_{n-1}, \tau_n \rangle$, abbreviated as $\langle \langle e_0, i_0 \rangle, \langle e_1, i_1 \rangle, \ldots, \langle e_n, i_n \rangle \rangle$.*

Definition 2.3 *(Tail) Given a path $p = \langle \langle e_0, i_0 \rangle, \ldots, \langle e_n, i_n \rangle \rangle$, a tail of p is any path $\langle \langle e_k, i_k \rangle, \ldots, \langle e_n, i_n \rangle \rangle$ which begins with an event in the path p, follows the same sequence as p after that event, and ends in the same final event.*

Definition 2.4 *(Path Length) The minimum length of a path $p = \langle \langle e_0, i_0 \rangle, \ldots, \langle e_n, i_n \rangle \rangle$ is defined as $l(p) \equiv \sum_{k=1}^{n} l_k$, and the maximum length of the path is defined as $u(p) \equiv \sum_{k=1}^{n} u_k$, given that $\tau_k \equiv [l_k, u_k]$ is the timing constraint between the events $\langle e_{k-1}, i_{k-1} \rangle$ and $\langle e_k, i_k \rangle$.*

Definition 2.5 *(Cover) Given two paths p_1 and p_2 which are disjoint except for the initial event $\langle e_0, i_0 \rangle$ and final event $\langle e_n, i_n \rangle$ which they have in common (referred to as reconvergent paths). If $l(p_1) > u(p_2)$, then p_1 covers p_2.*

Definition 2.6 *(Earliest Firing Time) Given the set of all paths P between two events $\langle e_0, i_0 \rangle$ and $\langle e_n, i_n \rangle$, the earliest firing time of $\langle e_n, i_n \rangle$ after $\langle e_0, i_0 \rangle$ is the maximum over all the minimum path lengths, denoted as $max[l(P)]$.*

Definition 2.7 *(Critical Path) Given the set of all paths P between two events $\langle e_0, i_0 \rangle$ and $\langle e_n, i_n \rangle$, a critical path p is any path whose removal from this set would change the earliest firing time of $\langle e_n, i_n \rangle$ after event $\langle e_0, i_0 \rangle$; that is, $max[l(P - \{p\})] \neq max[l(P)]$.*

In the example depicted in figure 1, there are two reconvergent paths from event $\langle Rout^+, i \rangle$ to event $\langle Rout^-, i \rangle$, namely, $\langle \langle Rout^+, i \rangle, \langle Aout^-, i \rangle, \langle Rout^-, i \rangle \rangle$ and $\langle \langle Rout^+, i \rangle, \langle Ain^+, i \rangle, \langle Rout^-, i \rangle \rangle$. Since the minimum path length along the second one is 1005 while the maximum path length along the first is 8, the second path covers the first. In addition, we see that the earliest firing time of $\langle Rout^-, i \rangle$ after $\langle Rout^+, i \rangle$ is $max[3, 1005] = 1005$.

Lemma 2.1 *Let p_1 be a path in the set of all paths P between the events $\langle e_0, i_0 \rangle$ and $\langle e_n, i_n \rangle$. If there exists a path p_2 which covers a path p_3, a tail of the path p_1, then p_1 is not a critical path.*

Proof: Given that p_2 covers p_3, they have an initial event $\langle e_k, i_k \rangle$ in common (from definition 2.5). Let p_4 be a path composed of the events from $\langle e_0, i_0 \rangle$ to $\langle e_k, i_k \rangle$ along the path p_1. There exists a path p_5 in the set P which is composed of the two paths p_4 and p_2. Therefore, $l(p_5) = l(p_4) + l(p_2)$ (1) and $l(p_1) = l(p_4) + l(p_3)$ (2) (from definition 2.4). We also know that $l(p_2) > u(p_3)$ since p_2 covers p_3 (from definition 2.5). We observe that $u(p_3) \geq l(p_3)$ (from definition 2.1). By transitivity we see that $l(p_2) > l(p_3)$ (3) which gives us that $l(p_5) > l(p_1)$ (4) (from (1), (2), and (3)). The earliest firing time of $\langle e_0, i_0 \rangle$ after $\langle e_n, i_n \rangle$ is given by $max[l(P)]$ which is equivalent to $max[l(p_1), l(p_5), l(P - \{p_1\} - \{p_5\})]$ (from definition 2.6). From (4), we see $max[l(P)] = max[l(p_5), l(P - \{p_1\} - \{p_5\})] = max[l(P - \{p_1\})]$. Therefore, the path p_1 is not a critical path (from definition 2.7). ∎

Definition 2.8 *(Redundant Rule) A redundant rule is one which never appears on a critical path.*

Lemma 2.2 *If a rule is redundant it can be removed from an ER system without changing the specified behavior.*

Synthesis of Timed Asynchronous Circuits

Chris Myers and Teresa H. -Y. Meng
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Abstract

In this paper we present a synthesis method that utilizes timing constraints to generate timed asynchronous circuits. By unfolding the cyclic graph specification of an asynchronous circuit into an infinite acyclic graph, we are able to use efficient algorithms to analyze the given timing constraints. We derive a sufficient condition for the removal of redundancy in the specification. Based on this condition, we only need to analyze a finite subgraph of the infinite acyclic graph for derivation of a correct implementation. To the reduced specification, we apply a systematic synthesis procedure that further optimizes the implementation based on the timing constraints. Using realistic circuit examples, we demonstrate that the resulting timed implementation can be significantly reduced in complexity from its speed-independent counterpart while remaining hazard-free under the given timing constraints.

1 Introduction

Speed-independent asynchronous circuits have recently gained much attention because they are robust to variations in delays. They are sometimes overly conservative since in many cases some knowledge of the circuit delays is available. The purpose of this paper is to present a methodology that takes advantage of these timing constraints (gate, wire, and environment delay information) to synthesize *timed circuits*. As long as the given bounded timing constraints hold, a timed circuit retains the same behavior with less circuit complexity than its speed-independent counterpart.

In order to synthesize timed circuits, it is necessary to determine the difference in the possible firing times between two transitions in a circuit specification. Recently, polynomial time algorithms have been developed by Vanbekbergen [1] and Dill [2] to determine this value for acyclic graphs. To apply these algorithms to circuit synthesis, these results must be expanded to handle cyclic specifications. One approach to this problem is to unfold the cyclic graph into an infinite acyclic graph [3]. We have derived a sufficient condition which breaks the infinite acyclic graph into a finite acyclic graph so that these algorithms can be used to correctly detect redundant circuitry.

Using timing constraints to simplify asynchronous circuits is not a new idea [4] [5]; however, most of these techniques were based on adding delay elements to avoid circuit hazards. We will show that hazard-free timed circuits can be synthesized without suffering any further delay compared with speed-independent circuits, while using the timing constraints to *reduce* the circuit complexity. In one practical memory interface [6] which was designed for use with an asynchronous microprocessor [7], we were able to reduce the circuit complexity by 50 percent using very conservative timing constraints.

This paper contains five sections. In section 2, we describe the specification language and the algorithm used to detect redundancy in a timed specification. In section 3, we discuss the synthesis procedure of timed circuits based on the redundancy found from the algorithm. In section 4, we present the memory interface example, and other synthesis examples. In section 5, we give our conclusions.

[0]This research was supported in part by an NSF Fellowship, ONR Grant no. N00014-89-J-3036, and a research grant from the Center for Integrated Systems, Stanford University.

2 Timing Analysis on Timed Specifications

Many methodologies have been proposed to specify asynchronous circuits. We have chosen to use Burns' *event-rule (ER) system* [3] for specification as it facilitates our timing analysis. *Events* are occurrences of transitions. There are two *transitions* associated with each signal s in a specification, namely, s^+ where $+$ denotes that the signal s is changing from low to high, and s^- where $-$ denotes that the signal s is changing from high to low. *Rules* are relationships governing the ordering of events. An ER system is an *infinite weighted marked directed acyclic graph* where the *vertices* are the events, the *edges* are the rules, and the *weights* are ranges of possible delays.

It is shown in [3] that specifications that are not *data dependent* or *disjunctive* can be transformed into an ER system. A specification is *data dependent* if one of two or more possible paths of action is chosen by the environment. Specifications with data dependencies can often be decomposed into several subgraphs which can be analyzed individually. A specification is *disjunctive* if there exists a transition in the specification that is specified to occur after either one transition or another, but not necessarily both. In addition to the above restriction on specifications, we presently allow only *one rising and one falling transition* of each signal in the specification. Specifications compiled using Martin's *communicating sequential processes* (CSP) synthesis method [8] and in the form of Chu's *signal transition graphs* (STG) [9] have both been transformed into event-rule systems.

2.1 The Event-Rule System

In this section, we briefly describe the ER system. For a more complete description please refer to [3].

Definition 2.1 *(Event-Rule System) An event-rule system is a pair $\langle E, R\rangle$ where E is a set of events and R is a set of rules. Each rule is denoted as $\langle e, f, i, \varepsilon, \tau\rangle$ for $i \geq \varepsilon$, where $\langle e, i-\varepsilon\rangle$ and $\langle f, i\rangle$ are two events of two transitions e and f, i is the* occurrence-index, *ε is the* occurrence-index offset, *and $\tau \equiv [l, u]$ where l is the lower-bound and u is the upper-bound of the timing constraint on the rule. A rule can be written in the form $\langle e, i-\varepsilon\rangle \overset{\tau}{\mapsto} \langle f, i\rangle$ which represents two vertices connected by an edge in the infinite graph of the ER system.*

In a rule the occurrence-index i is used to denote each separate instance of a particular transition. The first occurrence of a transition has $i = 0$, and i increments with each following occurrence.

The value of ε is the difference in the occurrence-index of the enabled transition f and the enabling transition e. Under the assumption that we only have one rising and one falling transition of each signal, ε can only take on the values 0 or 1.