Global Critical Path: A Tool for System-Level Timing Analysis

Girish Venkataramani[†] Mihai Budiu[‡] Tiberiu Chelcea[†] Seth C. Goldstein[†]
[†]Carnegie Mellon University
Pittsburgh, PA
{girish,tibi,seth}@cs.cmu.edu
[‡] Microsoft Research
Mountain View, CA
mbudiu@microsoft.com

ABSTRACT

An effective method for focusing optimization effort on the most important parts of a design is to examine those elements on the critical path. Traditionally, the critical path is defined at the RTL level, as the longest path in the combinational logic between clocked registers. In this paper, we present a system-level timing analysis technique to define the concept of a Global Critical Path (GCP), for predicting system-level performance. We show how the GCP can be used as a theoretical and practical tool for understanding, summarizing and optimizing the behavior of highly concurrent self-timed circuits. We formally define the GCP and show how it can be constructed using a discrete event model and hardware profiling techniques. The GCP provides valuable insight into the control-path behavior of circuits and in finding system-level bottlenecks. We have incorporated the GCP construction and analysis framework into a high-level synthesis and simulation toolchain, thus enabling complete automation in modeling, analysis and optimization.

Categories and Subject Descriptors

B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Design, Measurement, Performance

Keywords

Global critical path, System modeling, Hardware profiling

1. INTRODUCTION

An effective method for optimizing the most important parts of a design is to focus on the elements on the critical path. Typically, the critical path is defined as the longest path in a directed acyclic graph (DAG). In synchronous circuits, for example, the critical path usually refers to the longest path in the combinational logic (which is a DAG) between two clocked registers. This "local" notion of critical path has been the backbone of many CAD techniques.

Computing the critical path at the system-level is difficult due to the presence of cycles. Several approaches have been proposed which use Petri-Nets and marked graph based models [3, 10, 9]. However, they all assume that the system is completely deterministic and has no conditional behavior (called choice). However, most realistic systems cannot be built based on these simplified assumptions. For example, without choice, we cannot support MUX-like behavior, and without non-determinism we cannot support arbitration or loops with dynamic bounds.

We propose a framework for computing a system-level Global Critical Path (GCP) for realistic systems exhibiting all forms of conditional choice and non-determinism. We introduce profiling techniques, used with great success in the software world [8], for analyzing relative timing relations between signal transitions during hardware simulation. The challenge in hardware profiling is scalability, since there are potentially millions of signal transitions in a realistic system. We address this by modeling only key control events in the circuit. We show that this only slightly increases simulation time (about 7% in practice) and still captures the essential features of the circuit. Using the results of profiling, we formally define the concept of the GCP for the given circuit execution.

Using GCP to analyze a circuit has both strengths and limitations. A strength of our approach is that the GCP is computed deterministically, employing neither heuristics nor approximations. Our methods are light-weight and are fully integrated into an automated high-level synthesis flow [2]. Since GCP is a profiling-based approach, it inherits the weaknesses of all profiling approaches: the results are input-specific and changing the inputs may result in a different GCP. Thus, the users must be careful about chosing a representative input vector. In this paper, we focus on a particular control-path architecture based on asynchronous four-phase handshaking, but there are no fundamental reasons why our technique cannot be adapted to other design methodologies, including synchronous designs.

The main research contribution of this paper is a formal definition of the GCP for system-level timing analysis. We define the GCP and describe a simple, event-based model for capturing the timing relations between key control signal transitions (§ 3). The simplicity of the model results in an efficient and scalable approach to computing the GCP. We present a methodology to construct the GCP which combines the system-level event model with hardware profiling (§ 4). We show how this methodology is incorporated into a high-level synthesis toolchain by instrumenting a gate-level simulator to automatically compute the GCP for large asynchronous circuits generated by the toolchain. We show how the GCP can be analyzed and used within an optimization toolflow (§ 5). Through an exhaustive analysis, we show that there are very few possible classes of critical paths, and this knowledge can be used in simplifying the complexity of existing optimizations.

2. RELATED WORK

At the system-level, the *critical cycle* of execution has been shown to be the longest cycle in the system [3, 10, 9]. But this result is

DAC 2007, June 4–8, 2007, San Diego, CA, USA.

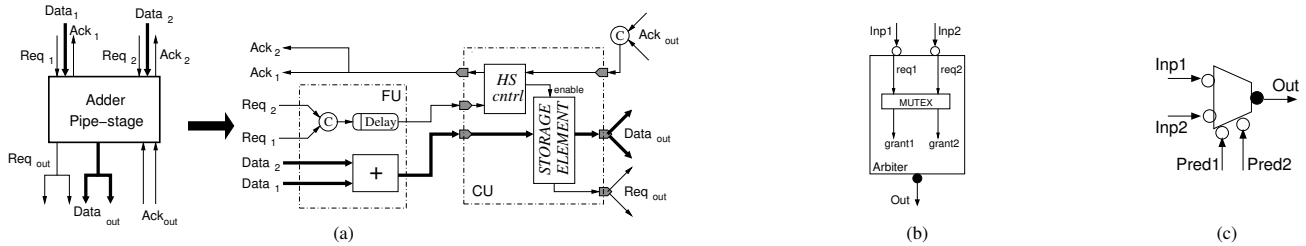


Figure 1: *Typical asynchronous pipeline stage interfaces: (a) an adder with a fanout of two and its circuit implementation, (b) an arbiter stage and (c) a 1-hot mux stage.*

applicable only to a limited class of circuits — those that exhibit no choice, and are completely deterministic systems. While some attempts to deal with deterministic chocie have been made [14, 6], analyzing systems with non-deterministic behavior is undecidable. In fact, in the presence of choice, Xie et. al. note that random simulation is the only known analysis technique [14].

Our work focuses on defining the GCP in the presence of non-determinism and arbitrary forms of choice, and we extend the concept of the *critical cycle* to the GCP, which may contain several critical cycles in sequence. We address non-determinism by employing profiling analysis during hardware simulation. Given a trace of event firing times, we show how the GCP can be efficiently and accurately constructed using an algorithm proposed by Fields et al. [4].

3. SYSTEM MODELING

Our system model is used for defining the timing relations between various circuit events and for capturing concepts such as choice and non-determinism. In § 4 we use these concepts to construct the GCP. We begin with a formal definition of the GCP.

3.1 GCP: A Formal Definition

Unlike most of the traditional concepts of critical path, which are static notions, the GCP summarizes the time-evolution of a circuit. We represent the circuit to be analyzed by a graph, $G = (V, E)$, which may include cycles (V are vertices, and $E \subseteq V \times V$ are edges). Time is denoted by T, and time steps by $t_i \in T$. A timed graph [14], $G \times T$, is a sequence of "snapshots" of the state of the circuit elements of G over time. The nodes of $G \times T$ are pairs (n, t_i), where $n \in V$.

The edges of the timed graph, $G \times T$, represent the set of signal transitions, denoted by $\mathcal{E} \subseteq (V \times T)^2$. If there is a signal transition leaving node n_1 at t_1 and reaching node n_2 at t_2—where $(n_1, n_2) \in E$—then $(n_1, t_1) \rightarrow (n_2, t_2) \in \mathcal{E}$. Observe that the timed graph $G \times T$ is acyclic, since $t_1 < t_2$ for every edge $(n_1, t_1) \rightarrow (n_2, t_2)$. We assign to each signal transition edge a length, which is the time difference between the two events: $||(n_1, t_1) \rightarrow (n_2, t_2)|| \stackrel{def}{=} (t_2 - t_1) > 0$. Finally, we **define** the *Global Critical Path (GCP)* as **the longest path of events** in the timed DAG $G \times T$, given by the sequence of edges, $(n_1^{GCP}, t_1^{GCP}) \rightarrow (n_2^{GCP}, t_2^{GCP}) \rightarrow \ldots \rightarrow (n_{last}^{GCP}, t_{last}^{GCP})$.

3.2 Events and Behaviors

This section describes how we model the behavior of self-timed circuits to capture the GCP. The salient features of the model are: (a) it is a precise, concise model for capturing timing relations between signal transitions, (b) it focuses only on the set of signals that determine overall circuit state, i.e., key control signals, and ignores the non-essential datapath and control signals, (c) it naturally captures choice and non-determinism, and (d) the primary goal of the model is to compute the GCP.

In this paper, we restrict ourselves to the class of asynchronous circuits built out of fully-decoupled pipeline stages, communicating using a 4-phase bundled-data handshake protocol [5][1]. In this protocol, each communication channel contains two control signals, *req* and *ack*, as shown in Fig. 1a. When the sender places new data on the channel, the *req* signal is raised. After consuming this data, the receiver raises the *ack* signal, after which both signals are lowered in the same order. Thus, data transfers are controlled by local flow-control instead of a global clock.

The key to efficiently computing the GCP for large circuits is to monitor only a subset of the circuit's control signals. The system model we now introduce captures dependence relations at the pipeline stage granularity. This is achieved by describing the dependence relation between the handshake events at the input and output of a given stage. Most of the signals internal to a stage (datapath and/or control) are ignored when building the model, significantly reducing the problem size.

Formally, we define the *Event Behavior Model* as $EBM = (\mathcal{E}, B, M, In, Out, R, X)$. At the heart of the model are events, $\mathcal{E}$, and behaviors, B. Each of the interesting signal transitions corresponds to an event, $e \in \mathcal{E}$. A behavior, $b \in B$, defines a partial execution ordering of the modeled events. Briefly, the model can be described as follows:

Live: A signal transition makes the corresponding event *live*. The set of live events at some point in time is denoted by M.

Kill: A signal transition may kill some events. For example, the rising transition of a signal $sig\uparrow$ will kill the falling transition of the same signal $sig\downarrow$. The relation R describes how events are killed. An event $e \in M$ is removed from M when $R(e) \cap M \neq \emptyset$. Thus, for a modeled signal, sig, $sig\downarrow \in R(sig\uparrow)$ and vice-versa.

Inputs: ($In : B \mapsto 2^{\mathcal{E}}$). This function describes *when* a behavior can occur. We say that a behavior b is *satisfied* when all of its input events are live: $In(b) \subseteq M$.

Outputs: ($Out : B \mapsto 2^{\mathcal{E}}$). Describes *the effect* of a behavior. In the absence of choice, a behavior can *fire* once it is satisfied. Firing a behavior makes its outputs alive. When a behavior b fires, $M := (M \cup Out(b)) - \{\, \forall e \mid (R(e) \cap Out(b)) \neq \emptyset\}$.

Exclusive: ($X : B \mapsto 2^{B}$). Defines behaviors whose outputs are mutually exclusive, and models non-determinism. This set is discussed in more detail in § 3.4.

3.3 Modeling A Simple Pipeline Stage

Fig. 1a is an asynchronous adder with a fanout of two. We abbreviate the (*req, ack*) signals from the input channel i with r_i and a_i respectively. The *req* signal of the output channel is r_o. The two *ack* signals from the two output consumers are a_{o_1} and a_{o_2}. The complete model for the adder is in Fig. 2a.

Behavior b_1 describes how the adder processes its data inputs. Once its input data channels are valid (indicated by $r_1\uparrow$ and $r_2\uparrow$)

[1]Many of these ideas are generalizable to other settings, including synchronous circuits.

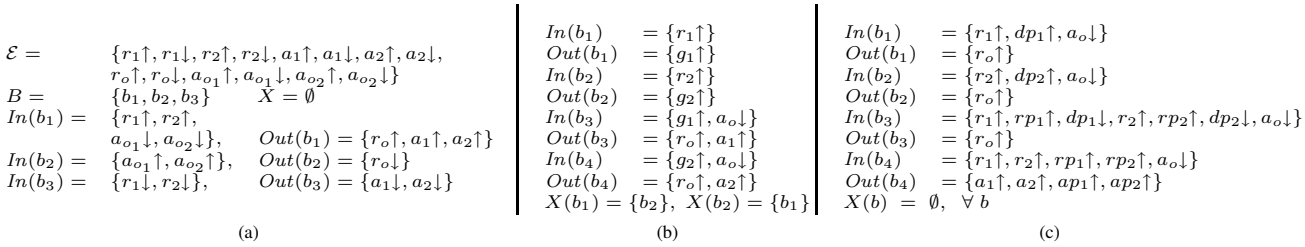


Figure 2: *Modeling (a) the adder in Fig. 1a, (b) the arbiter in Fig. 1b, and (c) the mux in Fig. 1c.*

and its previous output has been consumed ($a_{o_1}\downarrow$ and $a_{o_2}\downarrow$), the adder can process its inputs, generate a new output ($r_o\uparrow$), and acknowledge its inputs ($a_1\uparrow$ and $a_2\uparrow$). Behaviors b_2 and b_3 describe the reset phase of the handshake. Notice that In and Out only specify the control events at the pipeine stage interface, i.e., handshake events. Internal events, implementation details and datapath logic are usually abstracted away. This not only leads to models that are smaller than the actual circuits, but also decouples system modeling from system implementation.

3.4 Modeling Choice

We model choice by allowing multiple behaviors to fire the same event, i.e., $\exists b_1, b_2 \in B$, s.t., $Out(b_1) \cap Out(b_2) \neq \emptyset$. Depending on whether the choice is deterministic or not, additional constraints may be imposed.

Modeling non-determinism. Fig. 1b shows an arbiter stage, which exhibits non-deterministic choice. It arbitrates between two concurrent requests for access to a shared resource. The arbiter uses a Mutex (mutual exclusion element) [11] to determine which request is granted access. The winner's data is then transferred to the output port. The arbiter is modeled in Fig. 2b. In this example, we only show the behaviors related to the rising transition of the output, $r_o\uparrow$, which can be fired by two behaviors (b_3, b_4), and is illustrative of how the model encodes choice.

Non-determinism is modeled by the set X, which describes mutually exclusive behaviors. Two behaviors b and b', such that $b' \in X(b)$, may be satisfied simultaneously, but their outputs are mutually exclusive. Thus, b fires iff: $\forall b' \in X(b), Out(b') \cap M = \emptyset$. If two members of $X(b)$ are satisfied at the same instant, then one of them is randomly chosen to fire, reflecting the non-deterministic firing semantics of arbitration behaviors. In this example, behaviors b_1 and b_2 are mutually exclusive.

Modeling Unique Choice. In the presence of unique choice, an event may be generated by multiple behaviors, but there is a guarantee that the choice is deterministic. This is described by the invariant: $In(b_1) \cup In(b_2) \not\subset M$. Thus, at most one (out of several) behaviors is satisfied at any given instant, generating a unique choice event. Fig. 1c is a pipeline stage implementing a 1-hot encoded multiplexer, which exhibits unique choice due to early evaluation [2]. Fig. 2c shows the events and behaviors modeled for this mux. $\{r_i, a_i\}$ and $\{rp_i, ap_i\}$ are the handshake events of the data (Inp$i$) and predicate (Pred$i$) channels respectively ($i \in \{1, 2\}$). The dp_i events are control signals that specify the value of the predicate input, when it is valid (i.e., when the corresponding $rp_i\uparrow$ is live). When the data is invalid (i.e., $rp_i\downarrow \in M$), the $dp_i\downarrow$ events are live. A detailed exposition of this example can be found in [12].

4. DYNAMIC GCP CONSTRUCTION

In the presence of choice and non-determinism, finding the critical cycle of execution is an undecidable problem. In the past, two approaches have been proposed for dealing with choice — worst-case analysis [6] and stochastic analysis [14]. However, both these approaches deal only with unique choice and not non-determinism. Since our objective is to use the GCP for circuit optimization, we have chosen to focus on common-case execution behavior; in other words, hardware profiling. Using a collection of representative input-sets, we can simulate the gate-level circuit, and observe the firing times of the various system events during execution.

We have incorporated our GCP modeling and analysis methodology into the fully automated high-level synthesis (HLS) toolflow [2]. The Verilog back-end automatically infers the event model from the asynchronous pipeline structure, and translates the event model into Verilog PLI (Programming Language Interface) calls for profiling the model during post-layout simulation. Fig. 3a show the complexity of this methodology when analyzing kernels from Mediabench [7] suite. These are large designs containing on the order of tens of thousands of standard cell gates after tech-mapping. However, the fraction of events we model constitute about 2-3% of all circuit events; thus the overhead of profiling during gate-level simulation is low — on average, about 7% slowdown in simulation time.

Profiling generates a trace listing the times at which the model events fired during simulation. To compute the critical path from this trace we use essentially the Fields et al. algorithm [4], which processes the trace in reverse. For each executed behavior b_i, the last input event to fire, $e_k \in In(b_i)$, is deemed the locally critical input event that enables b_i to fire: $Crit(b_i) = e_k \in In(b_i)$. Starting from the last behavior, b_{last}^{GCP}, to fire in the trace (which by definition must be on the GCP), we recursively compute the behaviors that define the GCP: $\langle b_1^{GCP}, \ldots, b_i^{GCP}, b_{i+1}^{GCP}, \ldots, b_{last}^{GCP} \rangle$, such that for two adjacent behaviors, $Crit(b_{i+1}^{GCP}) \in Out(b_i^{GCP})$. The actual GCP is given by the event sequence corresponding to these behaviors: $e_i^{GCP} = Crit(b_i^{GCP})$.

Recall (from § 3.1) that GCP is a path on the timed graph, $G \times T$: $[(n_1^{GCP}, t_1^{GCP}) \rightarrow \ldots \rightarrow (n_i^{GCP}, t_i^{GCP}) \rightarrow \ldots (n_{last}^{GCP}, t_{last}^{GCP})]$. For long executions, millions of events could fire, thus making the GCP unwieldly. A practical way to summarize the information in the GCP is to *project* it on the untimed graph G, by discarding the time component t_i. The projection on G is a path: $[n_1^{GCP} \rightarrow n_2^{GCP} \rightarrow \ldots \rightarrow n_{last-1}^{GCP} \rightarrow n_{last}^{GCP}]$. Correspondingly, with each event $e \in E$, we associate the number of times it appears on the GCP.

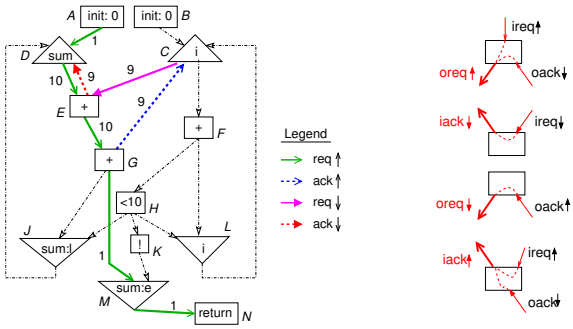
Fig. 3b shows the GCP for a circuit implementing the function, $sum = \sum_{i=1}^{10}(i + i)$. Each box is a pipeline stage, and edges between the boxes represent bundled data channels. An edge with special formatting represents a handshake event that falls on the GCP; the numerical annotation is frequency of occurence on the GCP. An edge with no number indicates that no handshake event of that channel is ever critical. A detailed description of this example can be found in [12].

5. USING THE GCP

The key to understanding the GCP is to determine the cause-and-effect relationships between the handshake control signals. Using

Benchmark	Events	Behaviors	Fraction Modeled	Simulation Slowdown
adpcm_d	1160	1273	0.029	11.22 %
adpcm_e	1360	1491	0.029	13.08 %
gsm_d	864	951	0.025	6.67 %
gsm_e	848	932	0.027	10.41 %
jpeg_d	4634	4998	0.023	7.27 %
jpeg_e	1510	1576	0.016	1.09 %
mpeg2_d	2062	2216	0.024	5.40 %
mpeg2_e	4566	4980	0.031	6.77 %

(a)



(b)

(c)

Figure 3: *(a) Model sizes for Mediabench kernels. The fourth column indicates what fraction of total observable events, the modeled events represent, and the last column shows the slowdown in simulation time due to hardware profiling. (b) GCP for* $sum = \sum_{i=1}^{10}(i+i)$ *is:* $req_{AD}\uparrow \rightarrow [req_{DE}\uparrow \rightarrow req_{EG}\uparrow \rightarrow ack_{GC}\uparrow \rightarrow req_{CE}\downarrow \rightarrow ack_{ED}\downarrow]^9 \rightarrow req_{DE}\uparrow \rightarrow req_{EG}\uparrow \rightarrow req_{GM}\uparrow \rightarrow req_{MN}\uparrow$*, (c) Causal relationships between adjacent critical events. Each of the four cases shows the input events that could be locally critical for generating the given output event. .*

the event model, we can reason about what event *could* have been critical before a given event, and why these two events would be chained on the GCP. For the types of controllers we have considered, which encompass a wide variety of real-world asynchronous circuits, Fig. 3c provides a complete summary of all possible dependence relations between the input and output handshake signals. For example, only ireq↑ and oack↓ can be the inputs producing oreq↑ as output, and so on.

From these patterns, we conclude that: if ack↓ is critical, then it is always the last event of the sequence ack↑ → req↓ → ack↓. The event preceding this sequence may be another ack↓ or a req↑. If a req↑ event is critical, then the event preceding it on the GCP may be another req↑ event or an ack↓ event. An exhaustive analysis of these event sequence patterns reveals that for any circuit implemented with 4-phase decoupled handshake, the topology of the GCP is always expressable as the following regular expression:

$$\begin{array}{ll} path_{data} & = [\text{req}\uparrow]* \\ path_{sync} & = [\text{ack}\uparrow \rightarrow \text{req}\downarrow \rightarrow \text{ack}\downarrow]* \\ \text{GCP} & = [path_{data} \rightarrow path_{sync}]* \end{array}$$

A $path_{data}$ sequence on the GCP reflects a condition where data production is slow and consumers are waiting for data to arrive. We refer to one or more consecutive $path_{data}$ sequences as a *data-delay path*. A $path_{sync}$ sequence on the GCP reflects a synchronization bottleneck, because the consumer is slow and is not ready to accept new data. We refer to one or more consecutive $path_{sync}$ sequences as a *sync-delay path*. The GCP can thus be summarized as a sequence of data-delay paths that are stitched together by sync-delay paths.

Such a classification of the GCP topology above allows us to reason about the optimization opportunities available. One such optimization is the slack matching problem which aims to eliminate sync-delay paths from the GCP. Venkataramani et. al. [13] used the GCP to formulate a quadratic time heuristic to this problem, which was previously shown to be NP-complete [1]. More examples of optimization opportunities and their implementation are described in [12].

6. CONCLUSIONS

Analogous to the concept of the critical path within combinational acyclic circuit graphs used in static timing analysis, we have presented the concept of a Global Critical Path (GCP) for complex, concurrent circuits containing cycles. We have described a formal model to analyze system-level timing in a self-timed circuit exhibiting all forms of choice and non-determinism. Using profile-based techniques, we have shown how to unambiguously construct the GCP from this model. The GCP topology provides valuable insights into the circuit's control architecture and can be used in rethinking existing optimizations. There is nothing intrinsic in the approach that prevents its adaptation to other asynchronous handshake protocols or synchronous circuits. The GCP has great potential for system-level analysis and optimization of digital circuits.

7. REFERENCES

[1] P. Beerel, M. Davies, et al. Slack matching asynchronous designs. In *ASYNC*, pp. 30–39, Mar. 2006.

[2] M. Budiu, G. Venkataramani, et al. Spatial computation. In *ASPLOS*, pp. 14–26, Oct. 2004.

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

[4] B. A. Fields, S. Rubin, et al. Focusing processor policies via critical-path prediction. In *ISCA*, 2001.

[5] S. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE Transactions on Very Large Scale Integration Systems*, 4-2:247–253, 1996.

[6] H. Hulgaard and T. Amon. Symbolic timing analysis of asynchronous systems. *IEEE Transactions on Computer Aided Design of Int. Circuits and Systems*, 19(10):1093–1104, October 2000.

[7] C. Lee, M. Potkonjak, et al. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *MICRO*, pp. 330–335, 1997.

[8] S. McFarling. Reality-based optimization. In *CGO*, pp. 59–68, 2003.

[9] P. McGee and S. Nowick. Efficient performance analysis of asynchronous systems based on periodicity. In *CODES+ISSS*, Sept. 2005.

[10] C. D. Nielsen and M. Kishinevsky. Performance analysis based on timing simulation. In *DAC*, pp. 70–76, 1994.

[11] I. Sutherland. Micropipelines: Turing award lecture. *CACM*, 32 (6):720–738, June 1989.

[12] G. Venkataramani, T. Chelcea, et al. Modeling the global critical path in concurrent systems. Technical Report CMU-CS-06-144, Carnegie Mellon University, Aug. 2006.

[13] G. Venkataramani and S. C. Goldstein. Leveraging protocol knowledge in slack matching. In *ICCAD*, Nov. 5-9 2006.

[14] A. Xie, S. Kim, et al. Bounding average time separations of events in stochastic timed Petri nets with choice. In *ASYNC*, pp. 94–107, April 1999.