Performance Analysis and Optimization of Asynchronous Circuits

Prabhakar Kudva, Ganesh Gopalakrishnan*, Erik Brunvand*

Department of Computer Science
University of Utah
Salt Lake City, UT 84112

Venkatesh Akella

Department of ECE
University of California,
Davis, CA 95616

Abstract

Asynchronous/Self-timed circuits are beginning to attract renewed attention as promising means of dealing with the complexity of modern VLSI designs. Very few analysis techniques or tools are available for estimating their performance. In this paper we adapt the theory of Generalized Timed Petri-nets (GTPN) for analyzing and comparing asynchronous circuits ranging from purely control-oriented circuits to those with data dependent control. Experiments with the GTPN analyzer are found to track the observed performance of actual asynchronous circuits, thereby offering empirical evidence towards the soundness of the modeling approach.

1 Introduction

Asynchronous (or Self-timed [13]) circuits are beginning to attract renewed attention as promising means of dealing with the complexity of modern VLSI designs. One of the widely touted advantages of asynchronous circuits is that they exhibit better average case performance (here, and elsewhere, performance means *overall speed* or *throughput*). In this paper, we propose performance analysis techniques for asynchronous circuits based on timed Petri nets that might help corroborate such claims. Features of our work include the ability to handle circuits with data dependent execution times, as well as the consideration of arbitration and resource usages.

Performance evaluation of synchronous circuits is almost always carried out in terms of the clock cycle time. The performance analysis problem is much more difficult for asynchronous circuits as they do not employ a global clock. In addition, the execution of asynchronous circuits is concurrent as well as data- and resource-dependent in nature. We believe that *timed Petri nets* can model systems with these characteristics.

Related Work and Overview of Approach

Asynchronous circuit performance analysis has been conducted at an analytical level by Sparsoe [19] and Williams [21], who study asynchronous pipelines. Petri Nets have been widely used for studying asynchronous computations [16] as well as for modeling as well as synthesizing synchronous and asynchronous circuits. They have also been widely used in performance analysis studies at the system level. Basically one can observe two categories of works in this area. In the first category, decision-free Petri net structures are used to model computations [18] while in the second category, Petri nets with decisions (or "choices") are allowed [22, 10]. Recent work on asynchronous circuit performance analysis falling under the first category include the works of Burns and and Williams [5, 21]. These authors model the computation as a constraint-graph and solve performance equations using linear programming techniques. Work by Hulgaard et al. [8] adapt the work by Burns and provide an algorithm to find exact bounds on the time separation of events in a branch-free process graph. Nielson and Kishinevsky [17] address the problem of determining the cycle time and critical paths through timing simulation. These works do not consider data dependent behaviors, arbitration for resources, and/or conditional branches.

The main contribution of this paper is to demonstrate that using an existing Generalized Timed Petri Net (GTPN) model [10] (detailed in Section 3), it is possible to hierarchically model a wide class of asynchronous circuits and systems in a uniform manner, while taking into account arbitration, resource usages, as well as data dependent behavior.

2 Modeling Self-timed Circuits

We assume two-phase transition signaling (up-going and down-going transition of a signal have the same meaning) and assume one wire per bit of data with the data-bundling assumption[20]. For concreteness, we assume macromodules-style circuits [20, 3, 2]. Macromodule circuits can be classified as control blocks (CB), datapath blocks (DB), and data-dependent control modules, or predicate blocks, PB (the latter two are illustrated in Figure 1). Examples of primitive CBs include the XOR gate, the Muller C-ELEMENT, and the CALL module. In Figure 1, the delay line associated with DB models its worst-case delay. PB implements data-dependent control flow, *e.g.*, conditional branches.

*Supported in part by NSF Award MIP-9215878

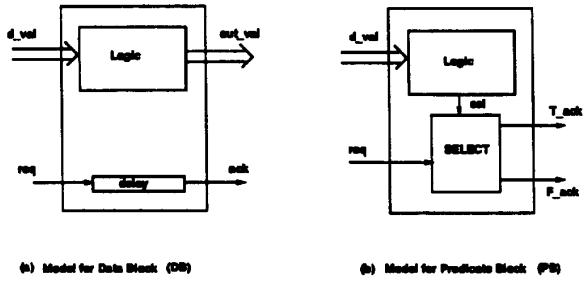

Figure 1: Models for DB and PB

A self-timed circuit can be represented by a one-safe Petri net [14, 6] which is obtainable by composing the Petri nets corresponding to various circuit elements. The name of a wire is attached to each Petri-net transition, and the firing of the Petri-net transition represents a signal transition on this wire. Each Petri net has *initial places* (IP) and *final places* (FP) that are disjoint, and correspond to the input- and output-ports of the circuit. The operation of a self-timed circuit is modeled by placing tokens in IP (corresponds to a *request*) and *waiting* for tokens to appear in FP (corresponds to an *acknowledge*). The environment of the circuit is responsible for placing the tokens in IP and removing the tokens from FP.

3 Generalized Timed Petri-net Theory

Timed Petri Nets

A GTPN (P, T, A, M_0, D, F, R) [10] is a Petri net which has been augmented to include a set of firing durations (D), a set of firing frequencies (F), and a possibly empty set of resources (R) associated with each transition, in addition to the usual fields of the set of places P, the set of Transitions T, the flow-relation $A \subseteq ((P \times T) \cup (T \times P))$, and the initial marking M_0. Each transition is associated with a *start firing* event and an *end firing* event in between which the firing of the transition is in progess for a deterministic firing duration of D units of time. The removal of tokens from a transition's input places occurs at *start firing* while the placement of tokens on a transitions's output places occurs at *end firing*. While the firing of a transition is in progress, the time to *end firing*, called the *remaining firing time* (RFT), decreases from D to zero. Thus, the state of a GTPN includes the distribution of tokens and the RFT values of currently active transitions. Each transition of a GTPN is generated by a set of start firing or a set of end firing events that occur simultaneously. The maximal set of transitions that can start firing simultaneously defines the next state of a GTPN. The *frequencies*, F, associated with each transition are used in assigning probabilities to the next state of the Petri net.

Modeling of delay and firing frequencies

The key timing parameter employed in our work is D, the delay of a transition. D can be obtained from a library, through experiments, or through known timing estimation techniques. Additional conventions in delay modeling are now explained.

A transition in the Petri net indicating a signal transition at the output of a gate is annotated by the delay of the gate, while a Petri-net transition indicating a signal transition at the input of a gate is annotated by the delay of the wire from the output of the driving gate to this input. For DBs, a transition indicating the acknowledge signal from a DB is annotated by the *average case* delay of the DB. Transitions indicating acknowledge signals from a PB are annotated by the sum of the average case delay of logic block associated with the PB and the delay of the Select element used in generating the acknowledge transition.

Firing frequencies (F) can be assigned to any transition. In case of a place with two or more arcs leading out of it (indicating a choice), branching frequencies may be assigned based on data dependent branching information. We note that the equivalent circuit paradigm of such an occurrence in the Petri net is a PB, which is a Select element whose control signal behavior is data dependent. All other transitions are assumed to have a firing frequency of 1.

Background on GTPNA analysis

We present below a quick overview of the GTPNA analysis techniques from Holliday [10], for the sake of completeness. Some of the relevant terminology and theorems of Markov Chain theory are also presented here. A detailed discussion of the Discrete-Time Markov Chains and their relationship to stochastic Petri nets are described in [1, 15]. The strongly connected components of the state space (when viewed as a directed graph) are the *classes* of the Markov Chain. The condensed graph (one vertex for each class) is a directed acyclic graph with one root. In the case of a finite state space, the interior vertices of the condensed graph are called *transient classes*. The leaf vertices of the condensed graph are called *recurrent classes* $\mathcal{R}$. In a typical evolution of the system being modeled, the system starts in a state in the root of this condensed graph. It then filters through the transient classes until it is absorbed by one of the recurrent classes. Once absorbed it stays in that recurrent class permanently.

In a GTPN, we are interested in the long run (or "steady state") behavior. Once absorbed in a particular recurrent class, the *long run* probability distribution over the states in the recurrent class needs to be computed. This stationary probability distribution $\Pi_{\mathcal{R}}$ is easy to find since it is the unique solution to the set of equations $\Pi_{\mathcal{R}} = \Pi_{\mathcal{R}} P_{\mathcal{R}}$ and $\sum_{j \in \mathcal{R}} \Pi_j = 1$, where matrix $P_{\mathcal{R}} = \{P_{ij} | i, j \in \mathcal{R}\}$. Here, P_{ij} refers to the probability of visiting state j from state i where both states are within the recurrent class $\mathcal{R}$. The fact that $\sum_{j \in \mathcal{R}} \Pi_j = 1$ corresponds to the fact that the set of states $\mathcal{R}$ is a *recurrent* class, *i.e.*, once execution enters $\mathcal{R}$,

it does not leave $\mathcal{R}$. Both P_{ij} and P_i are obtained from the underlying state graph of the timed Petri net [9]. Equation $\Pi_{\mathcal{R}} = \Pi_{\mathcal{R}} P_{\mathcal{R}}$ corresponds to the fact that in the long run, instead of having a probability of going from state i to state j in the recurrent class $\mathcal{R}$ (as defined by the transition matrix), we now have a vector of probability values $\Pi_{\mathcal{R}}$ of visiting each state i in the recurrent class $\mathcal{R}$.

Let S be the set of states in recurrent class $\mathcal{R}$ and let s_1 be a state in S. Also, define $RelTime(s_1)$ to be the time spent in state s_1 relative to the long-run time for $\mathcal{R}$. The long-run time for any state s_1 in a recurrent class $\mathcal{R}$ defined as

$$LongRunTime(s_1) = \frac{\sum_{k \in S} TimeInState(k)\ \Pi_k}{\Pi_{s_1}} \quad (1)$$

where $TimeInState(k)$ is the time spent in state k which is determined from the user given timing annotations in the Petri net and Π_i is the probability of visiting state i in the long run. Note this notion of long-run time is analagous to the notion of cycle time for the same state if data dependent choice were not present. $RelTime(s_1)$ is given by

$$RelTime(s_1) = \frac{TimeInState(s_1)\ \Pi_{S_1}}{\sum_{k \in S} TimeInState(k)\ \Pi_k}. \quad (2)$$

We also can obtain for each recurrent class, the utilization of the resources while in that class. The *expectation* of Resource Usages for a recurrent class, E[Resusages], is given by

$$E[ResUsages] = \sum_{k \in S} ResUsages(k)\ RelTime(k) \quad (3)$$

where $ResUsages(k)$ is the amount of resources used while in state k of the recurrent class.

Performance Estimation

In estimating the performance of asynchronous circuits, we employ the long-run time, the RelTime in a state and resource usage as performance measures. Our usage of long-run time in this manner is related to Zuberek's [22] use of this notion on examples with a single recurrence class. It is important to note that *long run time* is useful in the comparison of different designs and serves as a numerical measure with which to compare the designs. For example, Zuberek has suggested that by using this approach two architectures can be compared against each other, although he does this only for systems that can be modeled as Petri Nets with a single recurrence class. The ratio of the performance of an optimized and unoptimized design gives us a fair idea of the amount of improvement we can see in the optimization process.

Table 1: Table of experimental results

Circuit Name	Num of states	Perf. (ns) Long Run Time	CPU (secs) (GTPN)
EXOPT	223	1620	0.25
EX-RF	223	1420	0.25
EX-MVPC	223	1560	0.25
WAVE1	428	1010	0.13
WAVE2	428	1490	0.13
WAVE3	428	2120	0.13
ARP	44	390	0.05
ARP-OPT	44	340	0.05

4 Results and Conclusions

We have been able to analyze reasonably large examples and obtained performance estimates in less than a second of CPU time in most cases as shown in Table 1. First, the execution unit of the NSR [4] processor was modeled, shown as EXOPT in Table 1. Then, optimizations were performed on this model, and the long run times for a particular state of interest were compared. In the second example, that of a crossbar arbiter [7], the probabilities of connection requests for the crossbar switches was varied and the performance was compared in terms of the long run times, as shown in WAVE1, WAVE2 and WAVE3. Finally, an Asynchronous Reordering Pipeline [12] was also analyzed. ARP is an unoptimized version of this circuit and ARP-OPT an optimized version based on data dependent probablities of the reordering requests. These examples are discussed in detail in [11].

Although the GTPN analysis tools are very efficient in generating the reachability graph and use good numerical techniques for analysis, the problem of state explosion in generating the reachability graph of the Petri net can be significant in some cases. This was noticed while trying to analyze another version of the crossbar arbiter called the crisscrossing arbiter [7] which is very decoupled in its execution. This analysis could not be completed even for a 4x4 crossbar switch. Therefore, more efficient techniques to generate the reachability graphs are necessary.

Statistical reasoning can, in general, introduce errors in the analysis, because of the fact that the performance of the underlying net is abstractly represented by its long-run time. For the NSR processor's execution unit, we observed that our analysis gave results in close agreement (in the sense of *tracking*) with the actual chip's measured performance. The exact magnitude of the error will be studied in our future work.

Another possible source of inaccuracy in our current approach is the following. In modeling the delay exhibited by data blocks whose computations are data dependent at a fine-grain level (*e.g.*, carry-completion addition), we currently take the *average delay* exhibited by those units

with respect to the distribution of the data being handled by those units. A more direct way to incorporate the data dependent delays of data blocks has to be investigated. Finally, macromodule based circuits lend themselves very well to Petri net based analysis. However, we would also like to consider other classes of asynchronous circuits as part of our future work.

References

[1] AJMONE MARSAN, G. B., AND CONTE, G. *Performance Models of Multiprocessor Systems*. The MIT Press, 1986.

[2] AKELLA, V., AND GOPALAKRISHNAN, G. SHILPA: A High-Level Synthesis System for Self-Timed Circuits. In *International Conference on Computer-aided Design, ICCAD 92* (Nov. 1992), pp. 587–591.

[3] BRUNVAND, E. *Translating Concurrent Communicating Programs into Asynchronous Circuits*. PhD thesis, Carnegie Mellon University, 1991.

[4] BRUNVAND, E. The NSR processor. In *Proceedings of the 26th Annual Hawaiian International Conference on System Sciences, Volume 1* (Jan. 1993), T. Mudge, V. Milutinovic, and L. Hunter, Eds.

[5] BURNS, S. Performance evaluation of asynchronous circuits. Tech. Rep. TR-91-1, Computer Science Dept., California Institute of Technology, 1991.

[6] DILL, D. L., NOWICK, S. M., AND SPROULL, R. F. Specification and automatic verification of self-timed queues. *Formal Methods in System Design 1*, 1 (July 1992), 29–62.

[7] GOPALAKRISHNAN, G. Micropipeline wavefront arbiters using lockable c-elements. *IEEE Design and Test of Computers* (1994). Special Issue on Asynchronous Systems, Winter 1994.

[8] HENRIK HULGAARD, STEVEN M BURNS, T. A., AND BORIELLO, G. An algorithm for exact bounds on the time separation of events in concurrent systems. Tech. Rep. UW-CSE-94-02-02, Department of Computer Science, University of Washington, 1994.

[9] HOLLIDAY, M., AND VERNON, M. The GTPN analyzer: Numerical methods and user interface. *IEEE Comp Soc. 1986 Fall Joint Computer Conference* (Nov. 1986).

[10] HOLLIDAY, M., AND VERNON, M. A generalized timed petri net model for performance analysis. *IEEE Transactions on Software Engineering 13* (Dec. 1987), 1297–1310.

[11] KUDVA, P., GOPALAKRISHNAN, G. C., AND BRUNVAND, E. L. Performance analysis and optimization for asynchronous circuits. Tech. rep., University of Utah, Department of Computer Science, 1994.

[12] LIEBCHEN, A., AND GOPALAKRISHNAN, G. Dynamic reordering of high latency transactions in time-warp simulation using a modified micropipeline. In *International Conference on Computer Design (ICCD)* (1992), pp. 336–340.

[13] MEAD, C. A., AND CONWAY, L. *An Introduction to VLSI Systems*. Addison Wesley, 1980. Chapter 7 entitled "System Timing".

[14] MISUNAS, D. Petri nets and speed independent design. *Communications of the ACM 16*, 8 (Aug. 1973), 474–481.

[15] MOLLOY, M. K. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers C-31* (Sept. 1982), 417–423.

[16] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* (1989).

[17] NIELSON, C. D., AND KISHINEVSKY, M. Performance analysis based on timing simulation. Tech. Rep. ID-TR:1993-125, Department of Computer Science, Technical University of Denmark, 1993.

[18] RAMAMOORTHY, C., AND HO, G. Performance evaluation of asynchronous concurrent systems using petri nets. *IEEE Transactions on Software Engineering SE-6*, 5 (Sept. 1980), 440–449.

[19] SPARSOE, J., AND STAUNSTRUP, J. Design and performance evaluation of delay insensitive multi-ring structures. In *Proceedings of the 26th Annual Hawaiian International Conference on System Sciences, Volume 1* (Jan. 1993), T. Mudge, V. Milutinovic, and L. Hunter, Eds.

[20] SUTHERLAND, I. Micropipelines. *Communications of the ACM* (June 1989). *The 1988 ACM Turing Award Lecture.*

[21] WILLIAMS, T. E. *Self-Timed Rings and Their Applications to Division*. PhD thesis, Department of Computer Science, Stanford University, May 1991.

[22] ZUBEREK, W. Timed petri nets and preliminary performance evaluation. In *7th Annual International Symposium on Computer Architecture* (1980), pp. 88–96.