Open access • Report • DOI:10.21236/ADA447734

Performance Analysis and Optimization of Asynchronous Circuits — Source link

Steven M. Burns

Institutions: California Institute of Technology

Published on: 01 Apr 1991 - Conference on Advanced Research in VLSI

Topics: Asynchronous circuit, Asynchronous system, Electronic circuit, Digital electronics and Asynchronous communication

Performance Analysis and Optimization of Asynchronous Circuits

Steven M. Burns
Alain J. Martin

Computer Science Department
California Institute of Technology

Caltech-CS-TR-90-18

Performance Analysis and Optimization of Asynchronous Circuits[1]

Steven M. Burns and Alain J. Martin
Computer Science Department
California Institute of Technology
Pasadena, CA 91125 USA
{steveb,alain}@vlsi.cs.caltech.edu

Abstract

We present a method for analyzing the time performance of asynchronous circuits, in particular, those derived by program transformation from concurrent programs using the synthesis approach developed by the second author. The analysis method produces a performance metric (related to the time needed to perform an operation) in terms of the primitive gate delays of the circuit. Such a metric provides a quantitative means by which to compare competing designs. Because the gate delays are functions of transistor sizes, the performance metric can be optimized with respect to these sizes. For a large class of asynchronous circuits—including those produced by using our synthesis method—these techniques produce the global optimum of the performance metric. A CAD tool has been implemented to perform this optimization.

1 Introduction

Performance analysis of a synchronous computer system is simplified by an external clock that partitions the events in the system into discrete segments. In asynchronous systems, no such quantization exists. Instead, the operation of the system proceeds at a rate determined by the speed of its individual components, and the sequencing of the operation of the components. Unlike the synchronous case, the time needed to perform an asynchronous computation cannot be determined by merely counting the number of clock cycles required and multiplying by the clock period. Instead, to determine the time required to perform the computation as a whole, the times of those individual components of the computation that must occur sequentially are summed.

The techniques required to analyze asynchronous systems resemble those used to determine the clock period of a synchronous system, that is, summing the delays along the longest path through the combinational logic connecting adjacent latches. In the clocked case, the critical path has a clear beginning and a clear end because all paths are broken by latches. No clear separation

[1]to appear in the *Advanced Research in VLSI Conference*, Santa Cruz, CA, March 1991

is available in asynchronous systems. Analysis procedures must deal directly with cyclic critical paths; thus, existing critical-path analysis tools such as CRYSTAL [11] cannot be easily applied to this problem.

This paper discusses a framework for determining the time needed to perform computations using asynchronous systems, and applies especially to repetitive computations. Early work in the scheduling of concurrent computing elements [14] is closely related to our approach. Previous work in the area of timed Petri nets [13, 6] applies to this problem as well. The results we describe here are based on event-rule systems, a different formalism that is more closely connected to the methods we use to synthesize the asynchronous systems. Furthermore, we use our formalism to model the performance of asynchronous circuits, and provide a method for optimizing such circuits for performance.

Martin ([9] and elsewhere) has developed a synthesis method whereby asynchronous circuits are produced from concurrent program descriptions. By applying a systematic series of semantics-preserving transformations, a high-level description (CSP program) is refined, using the intermediate forms of handshaking expansions and production rules, until a provably correct asynchronous CMOS circuit is constructed.

At each stage of the synthesis procedure, a variety of transformations can potentially be applied. In the automated compiler of [2], these choices are made so that the same subcircuit template can be used to implement each instance of the same CSP language construct. Instances of these small templates are composed together to form a correct circuit that implements the original CSP program. However, in order to produce high-performance circuits, these choices must be directed by performance concerns. We observed this potential benefit of performance-directed transformations during the design of the Caltech Asynchronous Microprocessor [8]. The decisions of what transformation to apply were based on performance goals and this accounts for its high performance.

Event-rule (*ER*) systems can be used at each stage of the synthesis procedure to analyze the potential performance of the current refinement. Given a trace of the execution of a complete, closed program (environment included), an *ER* system can be generated from any of the intermediate forms: CSP programs, handshaking expansions, production rules, or CMOS circuits. The trace of execution is used to unroll each process that contains guarded commands into a straight-line process. In the cases where the trace of execution repeats, a repetitive *ER* system can be generated. The cycle period (the time between repeated events) can be determined using the techniques explained in Section 2.

These techniques provide an expression for the cycle period in terms of maximums and sums of individual component delays. At the circuit level, the component delays are functions of transistor widths and, as such, the cycle period can be optimized with respect to these widths. Non-linear optimization methods (such as those used in TILOS [3] and COP [7]) can

be used to perform the optimization of this expression for the cycle period. Our approach differs from those used for synchronous systems because we optimize all critical paths simultaneously.

2 Event-Rule Systems

An *event-rule (ER) system*, is a pair $\langle E, R \rangle$, where:

E is a set of events, and

R is a set of rules defining timed constraints between the events. Each $r \in R$ is written $e \overset{\alpha}{\mapsto} f$, where

$e \in E$ is the *source* of r,

$f \in E$ is the *target* of r, and

$\alpha \in [0, +\infty)$ is the *delay* of r.

Neither E nor R need be finite. When R is infinite, we require that no event depend on an infinite number of other events. That is, the set of predecessors (immediate or otherwise) of an event must be finite. Sometimes it is convenient to view $\langle E, R \rangle$ as a directed graph (multiple arcs and self-loops allowed); this graph will be referred to as the *constraint graph* G. For a given $\langle E, R \rangle$, there is a (possibly empty) set of functions, T, that satisfies:

T is a subset of the functions from E to $[0, +\infty)$;

$t \in T$ if and only if

$$t(f) \geq t(e) + \alpha \text{ for every } e \overset{\alpha}{\mapsto} f \in R \,. \tag{1}$$

We call a function t in the set T a *timing function* of $\langle E, R \rangle$. Each t represents a possible or consistent timing specification for the events of the system. If the set T is empty, the constraints (1) cannot be satisfied by any such function t. In this case, the $\langle E, R \rangle$ is called *infeasible*; otherwise, it is called *feasible*.

Example 2.1 Consider the $\langle E, R \rangle$ with

$$E = \{a, b, c\} \qquad R = \{a \overset{\alpha_b}{\mapsto} b, b \overset{\alpha_a}{\mapsto} a, b \overset{\alpha_c}{\mapsto} c\}.$$

This ER system is feasible if and only if $\alpha_a = 0$ and $\alpha_b = 0$. □

The smallest timing function denotes the earliest time at which the events of E can execute, and thus corresponds to the observed execution times of real circuits. Any feasible ER system with a constraint graph containing cycles can be transformed into an equivalent acyclic system [1].

Lemma 2.1 If the constraint graph G of an event-rule system $\langle E, R\rangle$ is acyclic, then there exists a unique function $\hat{t} \in T$ such that for every $t \in T$,

$$\hat{t}(e) \leq t(e) \text{ for every } e \in E. \tag{2}$$

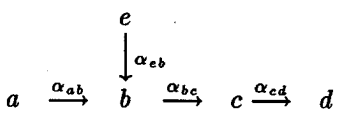
We call $\hat{t}$ the *timing simulation* of $\langle E, R\rangle$.

Proof: We propose the following recursive definition for $\hat{t}$:

$$\hat{t}(f) = \begin{cases} 0 & \text{if } sources(f) = \emptyset \\ \max\{\hat{t}(e) + \alpha \mid e \overset{\alpha}{\mapsto} f \in R\} & \text{otherwise.} \end{cases} \tag{3}$$

We can show, by contradiction, that $\hat{t}$ is the smallest timing function. (See [1] for a complete proof.) ∎

Example 2.2 The *ER* system defined by the constraint graph:



has the timing simulation:

$$\begin{array}{lcllcl} \hat{t}(a) & = & 0 & & & \\ \hat{t}(e) & = & 0 & \hat{t}(c) & = & \max(\alpha_{ab}, \alpha_{eb}) + \alpha_{bc} \\ \hat{t}(b) & = & \max(\alpha_{ab}, \alpha_{eb}) & \hat{t}(d) & = & \max(\alpha_{ab}, \alpha_{eb}) + \alpha_{bc} + \alpha_{cd} \end{array}$$

□

2.1 Repetitive Systems

ER systems of unbounded size corresponding to asynchronous circuits can be generated from bounded structures. Consider the event set E generated from a finite set E' by

$$E \;=\; E' \times \mathbb{N}\,.$$

The elements of E' are called *transitions*. An event $\langle u, i\rangle \in E$ is the indexed occurrence of the transition $u \in E'$. The non-negative integer i is called the occurrence index.

The rule set R is also generated from a finite set R'. The elements of R' are quadruples

$$r = \langle u, v, \alpha, \varepsilon\rangle \in R', \text{ where } R' \subseteq E' \times E' \times [0, +\infty) \times \mathbb{Z},$$

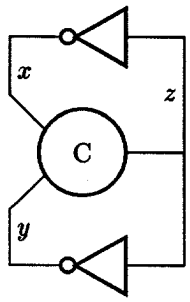
which we will write as

$$r \;=\; \langle u, i - \varepsilon\rangle \overset{\alpha}{\mapsto} \langle v, i\rangle.$$

The integer ε is called the *occurrence-index offset* of r. The dummy variable i is replaced by a non-negative integer no less than ε when r is instantiated

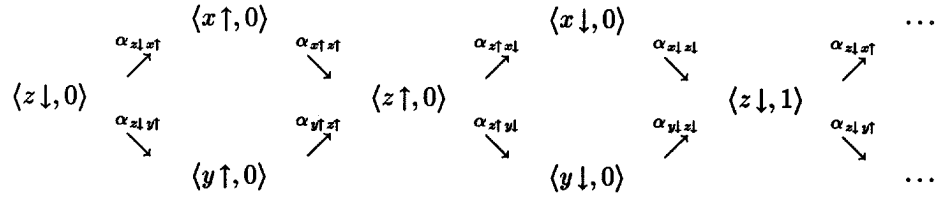
(an infinite number of times) to form the generated rule set R. We require $i \geq \max(0, \varepsilon)$ so that the occurrence indices of both the source and the target events of the instantiated rule are both non-negative and thus in E. We call $\langle E, R \rangle$ the (general) *ER* system generated from the *repetitive ER* system $\langle E', R' \rangle$.

Example 2.3 Consider the repetitive *ER* system constructed from a circuit containing a single Muller C-element:



$$
\begin{aligned}
E' &= \{x\uparrow, y\uparrow, z\uparrow, x\downarrow, y\downarrow, z\downarrow\} \\
R' &= \{\ \langle x\downarrow, i-1\rangle \overset{\alpha_{z\downarrow x\downarrow}}{\mapsto} \langle z\downarrow, i\rangle, \\
&\quad\ \langle y\downarrow, i-1\rangle \overset{\alpha_{y\downarrow z\downarrow}}{\mapsto} \langle z\downarrow, i\rangle, \\
&\quad\ \langle z\downarrow, i\rangle \overset{\alpha_{z\downarrow x\uparrow}}{\mapsto} \langle x\uparrow, i\rangle, \\
&\quad\ \langle z\downarrow, i\rangle \overset{\alpha_{z\downarrow y\uparrow}}{\mapsto} \langle y\uparrow, i\rangle, \\
&\quad\ \langle x\uparrow, i\rangle \overset{\alpha_{x\uparrow z\uparrow}}{\mapsto} \langle z\uparrow, i\rangle, \\
&\quad\ \langle y\uparrow, i\rangle \overset{\alpha_{y\uparrow z\uparrow}}{\mapsto} \langle z\uparrow, i\rangle, \\
&\quad\ \langle z\uparrow, i\rangle \overset{\alpha_{z\uparrow x\downarrow}}{\mapsto} \langle x\downarrow, i\rangle, \\
&\quad\ \langle z\uparrow, i\rangle \overset{\alpha_{z\uparrow y\downarrow}}{\mapsto} \langle y\downarrow, i\rangle\ \}
\end{aligned}
$$

Events are occurrences of transitions of circuit variables. The event $\langle x\uparrow, i\rangle$ represents the i^{th} occurrence of a transition from $x =$ **false** to $x =$ **true**. Similarly, $\langle x\downarrow, i\rangle$ represents the i^{th} occurrence of a transition from $x =$ **true** to $x =$ **false**. The repeated rules correspond to dependencies introduced by the inverters and the C-element that make up the circuit. Initially, x and y are **false** and z is **true**. We can represent the infinite sets E and R graphically:



Notice that event $\langle z\downarrow, 0\rangle$ has no predecessors. In the timing simulation, $\hat{t}(\langle z\downarrow, 0\rangle)$ is set to 0. [For ease of notation, $\hat{t}(\langle z\downarrow, 0\rangle)$ will sometimes be written as $\hat{t}(z\downarrow, 0)$.] The entire timing simulation $\hat{t}$, which can be constructed by inspection from the constraint graph, is:

$$
\begin{aligned}
&\hat{t}(z\downarrow, i) = p\,i && \hat{t}(z\uparrow, i) = \max(\alpha_{z\downarrow x\uparrow} + \alpha_{x\uparrow z\uparrow}, \alpha_{z\downarrow y\uparrow} + \alpha_{y\uparrow z\uparrow}) + p\,i \\
&\hat{t}(x\uparrow, i) = \alpha_{z\downarrow x\uparrow} + p\,i && \hat{t}(x\downarrow, i) = \max(\alpha_{z\downarrow x\uparrow} + \alpha_{x\uparrow z\uparrow}, \alpha_{z\downarrow y\uparrow} + \alpha_{y\uparrow z\uparrow}) + \alpha_{z\uparrow x\downarrow} + p\,i \\
&\hat{t}(y\uparrow, i) = \alpha_{z\downarrow y\uparrow} + p\,i && \hat{t}(y\downarrow, i) = \max(\alpha_{z\downarrow x\uparrow} + \alpha_{x\uparrow z\uparrow}, \alpha_{z\downarrow y\uparrow} + \alpha_{y\uparrow z\uparrow}) + \alpha_{z\uparrow y\downarrow} + p\,i
\end{aligned}
$$

where $p = \max(\alpha_{z\downarrow x\uparrow} + \alpha_{x\uparrow z\uparrow}, \alpha_{z\downarrow y\uparrow} + \alpha_{y\uparrow z\uparrow}) + \max(\alpha_{z\uparrow x\downarrow} + \alpha_{x\downarrow z\downarrow}, \alpha_{z\uparrow y\downarrow} + \alpha_{y\downarrow z\downarrow})$. □

2.2 Linear Timing Functions

In the previous example, we saw that the timing simulation of a repetitive *ER* system took on a simple form that is linear in the occurrence index i. This is not the case for all repetitive *ER* systems. However, as is shown later in Theorems 2.3 and 2.4, a *linear timing function* exists whenever the timing

simulation exists, and the "best" such function will be a good approximation of the timing simulation.

We call $\bar{t} \in T$ a *linear timing function* of $\langle E', R' \rangle$, if

$$\bar{t}(v,i) = x_v + p_v i \text{ for every } v \in E' \text{ and } i \in \mathbb{N}. \tag{4}$$

Each x_v and p_v are independent of i. For each $v \in E'$, x_v and p_v are called, respectively, the *offset* and the *cycle period* of the transition v.

Because of the linear form of $\bar{t}$, the timing function constraints, (1), reduce to linear inequalities in the offsets and the cycle periods of the transitions. All dependence on the occurrence index i can be eliminated. For each rule $r = \langle u, i-\varepsilon \rangle \overset{\alpha}{\mapsto} \langle v, i \rangle \in R'$, we have the infinite set of constraints:

$$\bar{t}(v,i) \geq \bar{t}(u, i-\varepsilon) + \alpha\,, \text{ for each } i \geq i_0 = \max(0, \varepsilon)\,.$$

Replacing $\bar{t}$ by its definition (4), we get

$$x_v + p_v i \;\; \geq \;\; x_u + p_u(i-\varepsilon) + \alpha\,.$$

Rearranging terms yields

$$x_v \;\; \geq \;\; x_u - p_u\varepsilon + \alpha + (p_u - p_v)\, i\,. \tag{5}$$

Equation (5) can never be satisfied for all i if $p_u > p_v$. Thus, the infinite set of constraints generated by r can be replaced by the two inequalities,

$$x_v \;\; \geq \;\; x_u - p_u\varepsilon + \alpha + (p_u - p_v)\, i_0\,, \text{ and} \tag{6}$$

$$p_v \;\; \geq \;\; p_u\,. \tag{7}$$

We define the *collapsed-constraint graph* G' of $\langle E', R' \rangle$ as the directed graph with nodes from E' and arcs from R'. From (7) we see that for a feasible solution to exist, a partial ordering between the p_v's must be satisfied. If two nodes, u and v, are in the same cycle of G', then p_u must equal p_v. Thus, all transitions in the same strongly-connected component of G' have the same cycle period. In the following, we consider only those repetitive ER systems in which G' is strongly connected, and we use p to denote the cycle period of every element in E'. Thus (6) simplifies to

$$x_v \;\; \geq \;\; x_u + \alpha - \varepsilon\, p\,. \tag{8}$$

Each arc of G' is labeled with $\alpha - \varepsilon\, p$ to signify constraint (8).

2.3 Minimum-Period Linear Timing Functions

Among the possible linear timing functions, there are those that minimize the cycle period p. The techniques of linear programming [4] can be used to find such a minimum-period linear timing function.

The constraints of a linear timing function, (8), are simple linear inequalities in the x_v's and p. By ordering the sets E' and R', we can construct a linear program in matrix form:

$$z = \min 0^T x + 1^T p,\ A'x + \varepsilon p \geq \alpha,\ x, p \geq 0\,. \tag{9}$$

The matrix A' is the arc-node incidence matrix of the collapsed-constraint graph G'. If row j of A' represents the constraint $r_j \in R'$, and column k of A' represents the transition $u_k \in E'$, then

$$a'_{jk} = \begin{cases} -1 & \text{if } u_k \text{ is the source transition of } r_j \\ 1 & \text{if } u_k \text{ is the target transition of } r_j \\ 0 & \text{otherwise .} \end{cases}$$

The jth elements of the (column) vectors ε and α are the occurrence-index offset and the delay of constraint r_j, respectively.

Example 2.4 Consider the system $\langle E', R' \rangle$ corresponding to a lazy-active/passive buffer (Figure 1) connected to the trivial environment:

$$\begin{aligned} E' &= \{lo\uparrow, li\uparrow, ro\uparrow, ri\uparrow, lo\downarrow, li\downarrow, ro\downarrow, ri\downarrow\} \\ R' &= \{\ \langle li\downarrow, i-1\rangle \overset{\alpha_{lo\uparrow}}{\mapsto} \langle lo\uparrow, i\rangle, \quad \langle ro\downarrow, i-1\rangle \overset{\alpha_{li\uparrow}}{\mapsto} \langle lo\uparrow, i\rangle, \\ &\quad\ \langle li\uparrow, i\rangle \overset{\alpha_{lo\downarrow}}{\mapsto} \langle lo\downarrow, i\rangle, \quad \langle ri\uparrow, i\rangle \overset{\alpha_{ro\uparrow}}{\mapsto} \langle ro\uparrow, i\rangle, \\ &\quad\ \langle lo\downarrow, i\rangle \overset{\alpha_{ro\uparrow}}{\mapsto} \langle ro\uparrow, i\rangle, \quad \langle ri\downarrow, i\rangle \overset{\alpha_{ro\downarrow}}{\mapsto} \langle ro\downarrow, i\rangle, \\ &\quad\ \langle lo\uparrow, i\rangle \overset{\alpha_{li\uparrow}}{\mapsto} \langle li\uparrow, i\rangle, \quad \langle lo\downarrow, i\rangle \overset{\alpha_{li\downarrow}}{\mapsto} \langle li\downarrow, i\rangle, \\ &\quad\ \langle ro\downarrow, i-1\rangle \overset{\alpha_{ri\uparrow}}{\mapsto} \langle ri\uparrow, i\rangle, \quad \langle ro\uparrow, i\rangle \overset{\alpha_{ri\downarrow}}{\mapsto} \langle ri\downarrow, i\rangle\ \} \end{aligned}$$

For this example, $A'x + \varepsilon p \geq \alpha$ is:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{lo\uparrow} \\ x_{li\uparrow} \\ x_{ro\uparrow} \\ x_{ri\uparrow} \\ x_{lo\downarrow} \\ x_{li\downarrow} \\ x_{ro\downarrow} \\ x_{ri\downarrow} \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} p \geq \begin{pmatrix} \alpha_{lo\uparrow} \\ \alpha_{lo\uparrow} \\ \alpha_{lo\downarrow} \\ \alpha_{ro\uparrow} \\ \alpha_{ro\uparrow} \\ \alpha_{ro\downarrow} \\ \alpha_{li\uparrow} \\ \alpha_{li\downarrow} \\ \alpha_{ri\uparrow} \\ \alpha_{ri\downarrow} \end{pmatrix}$$

□

The duality theorem of linear programming relates the primal program (9) to the dual program:

$$w = \max y^T \alpha,\ y^T A' \leq 0^T,\ y^T \varepsilon \leq 1,\ y \geq 0\,. \tag{10}$$

If both the primal and the dual programs have optimal solutions, then the optimal value z of the primal equals the optimal value w of the dual. Since A' is an arc-node incidence matrix, and thus $A'1 = 0$, any y feasible for (10) also satisfies $y^T A' = 0^T$. Thus, to determine the optimal value w, we need only solve the simplified dual program:

$$w = \max y^T \alpha,\ y^T A' = 0^T,\ y^T \varepsilon \leq 1,\ y \geq 0\,. \tag{11}$$

2.4 Cycle Vectors of a Graph

A *cycle* c of length ℓ in a directed graph, $\mathcal{G} = \langle \mathcal{N}, \mathcal{A} \rangle$, is an ordered subset $(a_0, a_1, \ldots, a_{\ell-1})$ of the arcs $\mathcal{A}$ such that $target(a_{k-1}) = source(a_k)$ for all $0 < k < \ell$ and $target(a_{\ell-1}) = source(a_0)$. The cycle c can be represented by a *cycle vector* u, a $\{0,1\}$-vector of length $|\mathcal{A}|$, where $u_j = 1$ if and only if the j^{th} arc of $\mathcal{A}$ is in the set c. For each cycle vector u, $u^T A' = 0^T$, where A' is the arc-node incidence matrix of the graph $\mathcal{G}$. A cycle is *simple* if each node in the cycle has one incoming and one outgoing cycle arc. The following lemma relates the simple-cycle vectors to an arbitrary vector y satisfying $y^T A' = 0^T$.

Lemma 2.2 Let U_i, $0 \leq i < q$ denote the simple-cycle vectors of a graph with arc-node incidence matrix A'. Then, if $y \geq 0$ is such that $y^T A' = 0^T$, there exist scalars $\theta_i \geq 0$, $0 \leq i < q$ such that

$$y = \theta_0 U_0 + \theta_1 U_1 + \ldots + \theta_{q-1} U_{q-1} \quad . \tag{12}$$

Proof: See [1] for complete proof. The proof uses induction on the number of simple cycles in the graph of A'. ∎

This lemma provides a straightforward means of determining the minimum cycle period p. By enumerating every simple cycle in the collapsed-constraint graph, and computing the sum of the delays and the sum of the occurrence-index offsets around each cycle, we can find p.

Theorem 2.3 The minimum cycle period, or equivalently, the optimal value of the dual program (11), is

$$\max \left\{ \left. \frac{U_k^T \alpha}{U_k^T \varepsilon} \right| \text{for all simple-cycle vectors } U_k \right\} \quad . \tag{13}$$

Proof: Let U be the simple-cycle matrix constructed by concatenating the (column) simple-cycle vectors $U_0, U_1, \ldots, U_{q-1}$. By construction, $U^T A' = 0$. By Lemma 2.2, any $y \geq 0$ with $y^T A' = 0^T$ can be represented as the product $U\Theta$, where the vector Θ has non-negative elements. The dual program (11) reduces to:

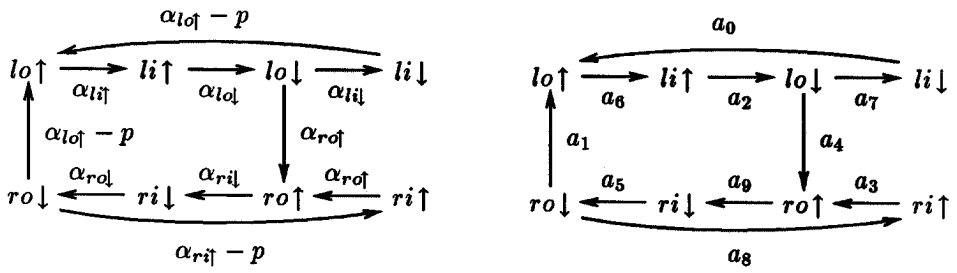
$$w = \max \Theta^T (U^T \alpha) \; , \Theta^T (U^T \varepsilon) \leq 1 \; , \Theta \geq 0 \, .$$

The dual of the reduced dual is easily solved:

$$z = \min \lambda \; , (U^T \varepsilon)\lambda \geq (U^T \alpha) \; , \lambda \geq 0 \, . \tag{14}$$

The smallest scalar λ that satisfies the vector inequality in (14) yields the desired minimum cycle period. ∎

Example 2.5 The minimum cycle period of the previous example can be determined by a cycle-period analysis. Two views of the collapsed-constraint graph are:



The view on the left uses the standard labeling; on the right, the labels denote the numbered arcs. The three cycles through the graph can be represented by the simple-cycle matrix:

$$U^T = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The vector inequality $(U^T\varepsilon)\,\lambda \geq (U^T\alpha)$ becomes:

$$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \lambda \geq \begin{pmatrix} \alpha_{lo\uparrow} + \alpha_{lo\downarrow} + \alpha_{ro\uparrow} + \alpha_{ro\downarrow} + \alpha_{li\uparrow} + \alpha_{ri\downarrow} \\ \alpha_{lo\uparrow} + \alpha_{lo\downarrow} + \alpha_{li\uparrow} + \alpha_{li\downarrow} \\ \alpha_{ro\uparrow} + \alpha_{ro\downarrow} + \alpha_{ri\uparrow} + \alpha_{ri\downarrow} \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix}$$

Thus, the minimum cycle period is $\max(\alpha_0, \alpha_1, \alpha_2)$. □

2.5 Efficient Computation of the Minimum Cycle Period

Enumeration of all simple cycles of a graph does not lead to an efficient procedure to compute the cycle period because an arbitrary graph may have exponentially more simple cycles than nodes or arcs.

Lawler in [5] provides an $O(|\mathcal{N}|\,|\mathcal{A}|\log B)$ solution to the *minimal cost-to-time ratio cycle problem*, which is equivalent to (13). Given a candidate cycle period that is too short and one that is too long, the algorithm uses binary search and a procedure to test—for a particular candidate cycle period—whether there is a negative cycle in the graph in order to determine the minimum cycle period in $\log B$ steps. (B is related to desired precision of the result.)

In [1], we provide an $O(|\mathcal{N}|\,|\mathcal{A}|)$ algorithm to determine the minimum cycle period when the sum of the ε values around each simple cycle is bounded. This algorithm is based on a direct solution to the linear program (11) and its corresponding dual and uses a customization of the general primal–dual algorithm [12].

2.6 Case Study: Comparison of Two FIFOs

We now apply the performance analysis techniques of Section 2 to compare the performance of two implementations of a first-in/first-out (FIFO) queue. While it is possible to extend repetitive *ER* systems to allow the analysis of

an unbounded linear array of identical processes [1], we instead perform the analysis directly on small arrays. We leave it as an exercise to the reader to show that, in these two cases, instantiating additional processes in the array does not increase the cycle period.

Three stages of a four-phase lazy-active/passive (*lap*) FIFO (Example 2.4) with the datapath between the stages are shown in Figure 1. For a circuit level implementation of a *lap* stage, see [1] or [10]. The three critical cycles through the transitions of the middle process are represented by bold arcs in the collapsed-constraint graphs (Figure 2). Assuming all delays in the circuit are small compared to the datapath delay, we get that $p = \alpha_{D\uparrow} + \max(\alpha_{D\uparrow}, \alpha_{D\downarrow})$.

Two stages of Ivan Sutherland's two-phase FIFO ([16], Figure 16) can be described by the circuit and collapsed-constraint graph shown in Figure 3. Since this circuit is symmetric in up and down transitions, we write, for example, li for both $li\uparrow$ and $li\downarrow$. The bold arcs in the graph represent the critical cycle. Assuming all delays in the circuit are small compared to the datapath delay, we get that $p = 2\alpha_D$.

A more complete analysis is provided in [1] which compares several other designs for FIFOs. This example shows that the best existing two-phase and four-phase implementations of a FIFO have comparable cycle periods. This result validates our long-standing beliefs that four-phase implementations are as fast as two-phase implementations, and that because of their simplicity and generality, they offer a better discipline in which to design asynchronous circuits.

2.7 Approximating the Timing Simulation

We now show that a minimum-period linear timing function provides an accurate approximation to the timing simulation.

Theorem 2.4 Let $\bar{t}$ and $\hat{t}$ be a minimum-period linear timing function and the timing simulation, respectively, of the connected repetitive system $\langle E', R' \rangle$. There exists a finite B such that for all $u \in E'$ and all $i \geq 0$

$$s_{u,i} = \bar{t}(u,i) - \hat{t}(u,i) \leq B \quad .$$

Proof: By definition for each u and i

$$\begin{aligned} \bar{t}(u,i) &= x_u + pi \\ \hat{t}(u,i) &= x_u + pi - s_{u,i} \quad . \end{aligned}$$

Each $s_{u,i}$ is non-negative because $\hat{t}$ is the smallest timing function. For the constraints generated from $r = \langle u, i-\varepsilon \rangle \overset{\alpha}{\mapsto} \langle v, i \rangle \in R'$, we define the non-negative slack variables, $\hat{z}_{r,i}$ and $\bar{z}_r$, thus transforming inequalities into equalities:

$$x_u - p\varepsilon + \alpha + \bar{z}_r = x_v \tag{15}$$

$$x_u - p\varepsilon - s_{u,i-\varepsilon} + \alpha + \hat{z}_{r,i} = x_v - s_{v,i} \tag{16}$$

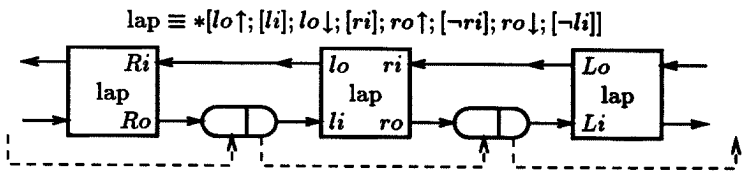


Figure 1: Three stages of a four-phase *lap* FIFO. The cigar-shaped objects represent the datapaths. The dashed lines denote the flow of data in the FIFO.

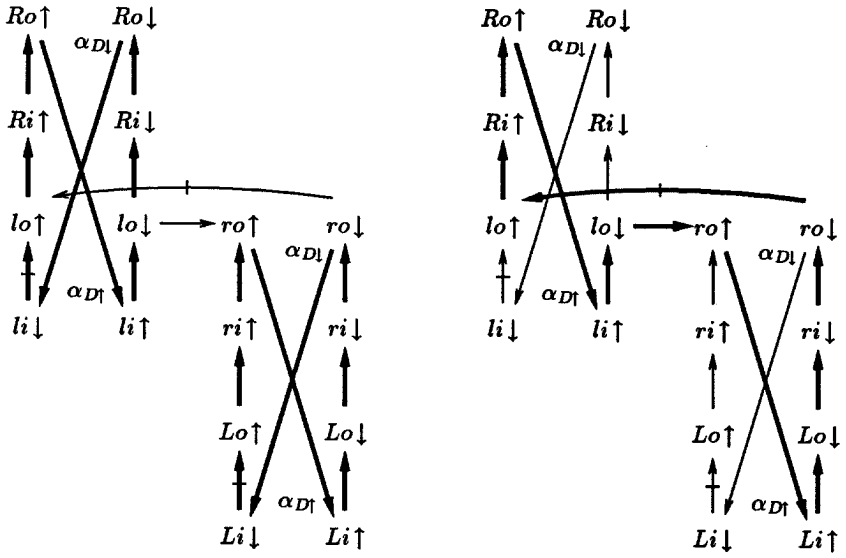


Figure 2: Critical cycles through the graph of a three-stage *lap* FIFO. The two graphs are identical; the sets of connected, bold arcs represent the critical cycles. An arc with a tick mark has $\epsilon = 1$. All other arcs have $\epsilon = 0$.

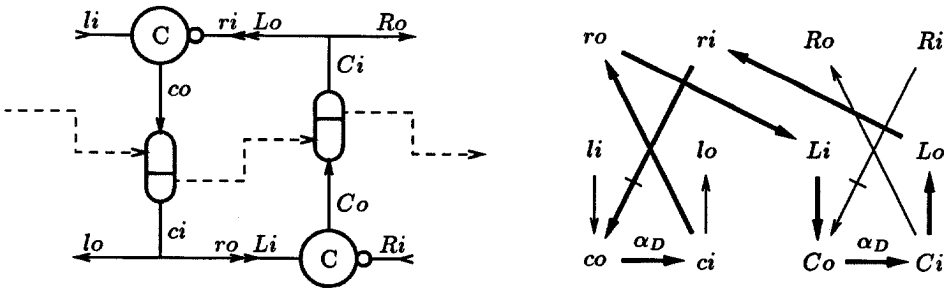


Figure 3: Circuit and graph of Sutherland's two-phase FIFO. The operators marked *C* are Muller C-elements.

By subtracting these equations and simplifying, we get

$$\bar{z}_r - \hat{z}_{r,i} = s_{v,i} - s_{u,i-\varepsilon} \quad . \tag{17}$$

From Theorem 2.3, $p\sum_{r\in c}\varepsilon_r = \sum_{r\in c}\alpha_r$ for at least one cycle c. Adding the constraints on $\bar{t}$, (15), for each $r \in c$, we see that

$$\sum_{r\in c} x_{u_r} - p\sum_{r\in c}\varepsilon_r + \sum_{r\in c}\alpha_r + \sum_{r\in c}\bar{z}_r = \sum_{r\in c} x_{v_r} \quad .$$

Since along any cycle $\sum_{r\in c} x_{u_r} = \sum_{r\in c} x_{v_r}$, we have for all $r \in c$,

$$\bar{z}_r = 0 \quad .$$

By (17), $s_{u,i-\varepsilon} \geq s_{v,i}$ for all $i \geq \max(0,\varepsilon)$ and all u, v on cycle c. By summing along the cycle c, we see that for each $u \in c$ and $i' \geq 0$

$$s_{u,i'} \geq s_{u,i} \text{ where } i = i' + \sum_{r\in c}\varepsilon_r \quad .$$

Therefore, we can bound $s_{u,i}$, for every $u \in c$, by

$$B' = \max\left\{ s_{u,i'} \,\middle|\, u \in c \wedge i' < \sum_{r\in c}\varepsilon_r \right\} \quad .$$

For any transition v not on cycle c, we find a path P_v to transition v from a transition u on c. Because G' is strongly connected, such a path must exist and be independent of i. Then, by summing (17) along that path, we get for all $i, i' \geq 0$

$$s_{u,i'} + \sum_{r\in P_v}\bar{z}_r \geq s_{v,i} \quad ,$$

where $i = i' + \sum_{r\in P_v}\varepsilon_r$. But $\sum_{r\in P_v}\bar{z}_r$ is independent of i; thus, $s_{v,i}$ is bounded by a quantity that does not increase with successive occurrences. Thus, every $s_{v,i}$ with $v \notin c$ is bounded by B where

$$B \geq \max\left\{ s_{v,i} \,\middle|\, v \notin c \wedge i < \sum_{r\in P_v}\varepsilon_r \right\} \text{, and}$$

$$B \geq B' + \max\left\{ \sum_{r\in P_v}\bar{z}_r \,\middle|\, v \notin c \right\} \quad . \blacksquare$$

3 Performance Optimization

Using the above analysis method, a performance metric (the minimum cycle period p) can be expressed in terms of the primitive delays of an *ER* system. These delays can be estimated from the sizes of the transistors that make up the operators of the circuit and from the way these operators are interconnected, by using a simple resistance-capacitance (RC) timing model. Composing the performance metric in terms of component delays with the delay approximation of the operators in terms of transistor sizes, we get an expression for the performance of the system in terms of transistor sizes. This expression is minimized, producing optimal sizes for the transistors.

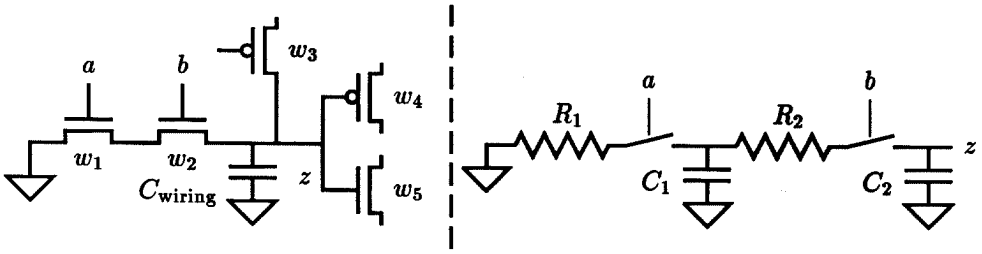


Figure 4: Linear approximation of a CMOS pulldown.

3.1 Tau Model

A simple RC switch model is used to relate each individual delay α to the various widths (w's) of the circuit's transistors. Each transistor is modeled as a switch with a resistance inversely proportional to its width. The gate of each transistor has a capacitance to ground proportional to its width. Source and drain capacitances are also proportional to transistor widths. Thus, the delays between $a\uparrow$ and $z\downarrow$, and $b\uparrow$ and $z\downarrow$, of the circuit shown in Figure 4 are modeled as:

$$\begin{aligned} \alpha_{a\uparrow z\downarrow} &= R_1C_1 + (R_1+R_2)C_2, \quad \alpha_{b\uparrow z\downarrow} = (R_1+R_2)C_2, \\ R_1 &= \mu/w_1, \quad R_2 = \mu/w_2, \\ C_1 &= K_{\text{int}}(w_1+w_2), \quad C_2 = K_{\text{ext}}(w_2+w_3) + C_{\text{wiring}} + K_{\text{g}}(w_4+w_5)\,, \end{aligned}$$

where μ is a constant that describes the differing per-unit-width strengths of the n- and p-channel transistors, K_{int} is the per-unit-width capacitance contributed by internal (to the series chain) drain and source terminals, K_{ext} is the per-unit-width capacitance contributed by external (the output node) drain terminals, K_{g} is the per-unit-width gate capacitance, and C_{wiring} is the capacitance contributed by wiring. All capacitances are expressed in terms of transistor width; thus $K_{\text{g}} = 1$. Each delay α is expressed in units of τ, the time needed for a unit-width n-channel transistor to switch a unit-width load. (Thus, $\mu_{\text{n}} = 1$, $\mu_{\text{p}} > 1$.) The values of K_{int}, K_{ext} and C_{wiring} are not constant, but depend on the final circuit layout that depends weakly on the transistor widths. This dependence is normally small and is ignored in the optimization problem.

Example 3.1 As an example, we will construct the optimization equations for the C-element circuit of Example 2.3. For purposes of this example, the constants of the tau model take on these values:

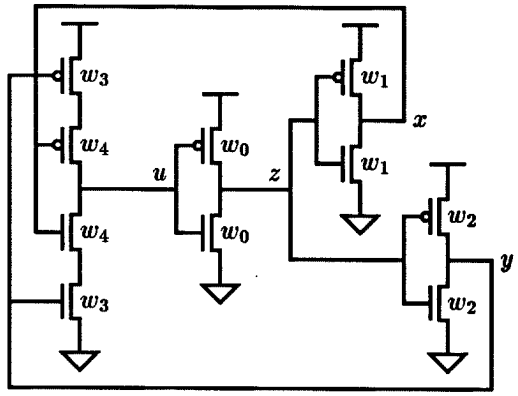
$$K_{\text{ext}} = 1, \quad K_{\text{int}} = 0.5, \quad K_{\text{g}} = 1, \quad \mu_p = \mu_n = 1, \quad C_{\text{wiring}} = 0\,.$$

Since the mobilities of the pull-up and pull-down devices are assumed to be identical, by symmetry the widths of the pull-up and pull-down devices are identical as are the pull-up and pull-down delays. By removing the transition-direction reference from the delay names (for example, $\alpha_{z\uparrow x\downarrow}$ is named α_{zx}), and expanding the delays

between transitions on x and y, and transitions on z (by introducing the new variable u), the expression for the cycle period becomes

$$p = 2\left(\alpha_{uz} + \max(\alpha_{zx} + \alpha_{xu}, \alpha_{zy} + \alpha_{yu})\right).$$

The circuit and corresponding α values are shown below:



$$\alpha_{uz} = \frac{1}{w_0}(2w_0 + 2w_1 + 2w_2)$$
$$\alpha_{zx} = \frac{1}{w_1}(2w_1 + 2w_4)$$
$$\alpha_{zy} = \frac{1}{w_2}(2w_2 + 2w_3)$$
$$\alpha_{xu} = \left(\frac{1}{w_3} + \frac{1}{w_4}\right)(2w_4 + 2w_0)$$
$$\alpha_{yu} = \frac{1}{2w_3}(w_3 + w_4) + \alpha_{xu}$$

□

3.2 Convex Objective Function

Every α derived using this simple model (and also other more accurate ones) is a posynomial function (polynomial with positive coefficients and positive variables) of the transistor widths w's, and thus a convex function of the $\log w$'s [4]. Because both the sum and the maximum of two convex functions are convex functions, the resulting expression for p is a convex function of the $\log w$'s; and, thus, each minimum of p is global. The addition of convex constraints, for example, to limit power consumption or to bound transistor sizes, does not alter the unique minimum property.

Example 3.2 The optimal value for the cycle period of Example 3.1 occurs when the width values are a positive scalar multiple of

$$(w_0, w_1, w_2, w_3, w_4) = (1.0, 0.4782, 1.1632, 1.8002, 0.9209).$$

(See [1] for an explanation of how to achieve a unique minimum point by constraining the power consumption and/or the largest and smallest transistor width.) □

We have implemented a CAD tool for solving the resulting non-linear, non-differentiable, convex optimization problems based on the *subgradient* techniques described by Shor [15]. Table 1 lists the results of this program when applied to a variety of circuits. The column n_{trans} denotes the number of transistors in the circuit, and thus the number of free variables in the optimization problem. The columns p_{unsized} and p_{sized} show the cycle period in units of τ of the circuit before and after optimization. In the unsized case, all n-channel transistors have equal sizes, and each p-channel transistor is $\sqrt{\mu_p}$ times (optimal for an inverter ring) wider than the n-channel transistors. The *% imp* column shows the percent improvement in p provided by

	n_{trans}	p_{unsized}	p_{sized}	% imp	CPU
Three inverters	6	42	42	0	1.4
C-element and two inverters	10	66	61	8	2.8
One stage *lap*	21	143	114	25	9.3
Three stage *lap*	59	189	143	32	49
Three stage *lap**	59	189	143	32	25
Ten stage *lap*	192	189	151	25	279
Ten stage *lap**	192	189	151	25	189
Simple microprocessor control*	285	646	430	50	369

Table 1: Performance of CAD tool.

the optimization. The CPU column denotes the number of user seconds needed to compute the optimum value on a SUN/Sparcstation 1. A direct implementation of the optimization algorithm requires $O(n_{\text{trans}}^2 + n_{\text{cycles}}\ell_{\max})$ arithmetic operations per iteration, where n_{cycles} is the number of cycles used to form the cycle-period function and $\ell_{\max}$ is the maximum number of arcs per cycle. A more sophisticated implementation (results denoted by *) that uses the primal–dual algorithm of Section 2.5 to determine the cycle period at each iteration requires only $O(n_{\text{trans}}^2)$ arithmetic operations per iteration.

4 Summary

We have presented a method for determining the performance of circuits described by event-rule systems. Furthermore, we have shown how to optimally size transistors in such circuits. What we have not shown, due to lack of space, is how to transform the specifications of asynchronous circuits that we use for synthesis into *ER* systems. With the addition of these techniques, we have a complete method combining synthesis and performance analysis. The performance analysis can be done early and at each level of the synthesis procedure and can be used to guide the synthesis of efficient circuits. A complete description is given in [1].

Acknowledgments

We wish to thank Nan Boden, Pieter Hazewindus, Marcel Van der Goot, Dražen Borković, Tony Lee, José Tierno, Mass Sivilotti, and Dian De Sha for their comments on this manuscript. The research described in this paper is sponsored by an IBM Graduate Fellowship; and by the Defense Advanced Research Projects Agency, DARPA Order number 6202, monitored by the Office of Naval Research under contract number N00014–87–K–0745.

References

[1] Steven M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* Ph.D. thesis, California Institute of Technology, 1991. CS-TR-91-1.

[2] Steven M. Burns and Alain J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In J. Allen and F. Leighton, editors, *Advanced Research in VLSI, Proceedings of the Fifth MIT Conference*, pages 35–50. MIT Press, Cambridge, MA, 1988.

[3] J.P. Fishburn and A.E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *IEEE ICCAD*, pages 326–328, November 1985.

[4] Joel Franklin. *Methods of Mathematical Economics.* Springer-Verlag, Berlin, 1980.

[5] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[6] Jan Magott. Performance evaluation of concurrent systems using Petri nets. *Information Processing Letters*, 18:7–13, 1984.

[7] David P. Marple and Abbas El Gamal. Optimal selection of transistor sizes in digital VLSI circuits. In Paul Losleben, editor, *Advanced Research in VLSI, Proceedings of the 1987 Stanford Conference*, pages 151–172. MIT Press, Cambridge, MA, 1987.

[8] A.J. Martin, S.M. Burns, T.K. Lee, D. Borković, and P.J. Hazewindus. The design of an asynchronous microprocessor. In C.L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI*, pages 351–373, Cambridge, MA, 1989. MIT Press.

[9] Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C.A.R. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming.* Addison-Wesley, Reading, MA, 1990.

[10] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Staunstrup, editor, *Formal Methods for VLSI Design.* North-Holland, Amsterdam, 1990.

[11] J.K. Ousterhout. A switch-level timing verifier for digital MOS VLSI. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 4(3), July 1985.

[12] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[13] C.V. Ramamoorthy and Gary S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, September 1980.

[14] Raymond Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, October 1968.

[15] N.Z. Shor. *Minimization Methods for Non-Differentiable Functions.* Springer-Verlag, Berlin, 1985. Translated from Russian.

[16] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.