Pipeline Optimization for Asynchronous Circuits: Complexity Analysis and an Efficient Optimal Algorithm

Sangyun Kim, *Member, IEEE*, and Peter A. Beerel, *Member, IEEE*

***Abstract*—This paper addresses the problem of identifying the minimum pipelining needed in an asynchronous circuit (e.g., number/size of pipeline stages/latches required) to satisfy a given performance constraint, thereby implicitly minimizing area and power for a given performance. The paper first shows that the basic pipeline optimization problem for asynchronous circuits is NP-complete. Then, it presents an efficient branch and bound algorithm that finds the optimal pipeline configuration. The experimental results on a few scalable system models demonstrate that this algorithm is computationally feasible for moderately sized models.**

***Index Terms*—Asynchronous circuits, complexity analysis, pipeline optimization.**

I. Introduction

MOST designs use a global clock to synchronize data flow. Recently, however, asynchronous designs have demonstrated potential benefits in low power, high average performance, composability, design reuse, and improved noise immunity and electromagnetic compatibility. Many tools and techniques have been developed to address hazard freedom and area minimization. Estimation and optimization of their performance, however, remain somewhat of a stumbling block. The basic problem is that the complex interaction of various handshaking protocols makes direct optimization for performance very difficult.

There are two basic approaches to the performance optimization of asynchronous circuits. The first approach involves using performance analysis techniques to guide manual or semi-automated design changes (e.g., [25]). The alternative approach is to develop synthesis techniques that directly optimize for performance. Successful efforts in this area have addressed transistor sizing [5], technology mapping [6], and allocation and scheduling (e.g., [1]–[3]) in high-level synthesis.

This paper formalizes a new performance optimization area for asynchronous circuits called pipeline optimization. In particular, most previous researches are either at a much lower level than pipelining (e.g., logic synthesis) or assuming that the pipelining is fixed (e.g., in high-level synthesis). More specifically, to the best of the authors' knowledge, no automated tool exists to indicate the degree of pipelining (e.g., number of pipeline stages) needed to achieve a given performance. In other words, while it is well known that good pipelining design styles in asynchronous circuits are critical to reduce the asynchronous control circuit overhead (e.g., [25]–[27]), it is more difficult to determine the best means of breaking up a large combinational block into pipeline stages to achieve a given performance. In fact, recent experiences suggest that this optimization problem is getting more difficult. Namely, Caltech researchers propose partitioning asynchronous data paths into bit slices and pipelining between bit slices to achieve higher throughput [8], [18]. When combined with standard pipelining between functional component boundaries, this creates a complex two-dimensional (2-D) pipeline. As a general rule in asynchronous design, the number of pipeline stages increases the power and area of the design due to extra completion sensing and control logic. Thus, one reasonable objective for pipeline optimization is to identify the minimal pipelining needed to satisfy a given performance constraint, thereby implicitly minimizing area and power for a given performance.

This paper first proposes an abstract performance model of the circuit on which the basic pipeline optimization problem can be defined. This abstract performance model is sufficient to characterize a variety of pipelining schemes, including those from Williams, Caltech, and the University of Manchester [10], [18], [26], [27]. However, it is currently restricted to deterministic pipelines (no choice) and only considers fixed delays. It is also limited to designs that have the important property of slack elasticity, which guarantees that their input/output behavior does not change if the degree of pipelining within the design changes [17]. The authors first explore the complexity of the optimization problem. One contribution of this paper is a proof that the defined asynchronous pipeline optimization problem is NP-complete. In addition, the authors present an efficient branch and bound algorithm that demonstrates the feasibility of the optimization problem for moderately sized models. The experimental results on a few scalable models of asynchronous systems demonstrate that the branch and bound solver can successfully find the optimal solution among over 2^{30} pipeline configurations. It may be worth pointing out similarities with a somewhat analogous problem of retiming [22] in the domain of synchronous circuits. In particular, like the problem here, a basic version of retiming is to

Manuscript received April 29, 2003; revised January 28, 2004 and July 11, 2004. This paper is funded in part by the National Science Foundation (NSF) Grant CCR-9812164 and NSF Information Technology Research (ITR) Award No. CCR-00-86036. This paper was recommended by Associate Editor S. Hassoun.

S. Kim is with Synopsys, Inc, Hillsboro, OR 97124 USA.

P. A. Beerel is with the University of Southern California, Los Angeles, CA 90089 USA.

Digital Object Identifier 10.1109/TCAD.2005.853689

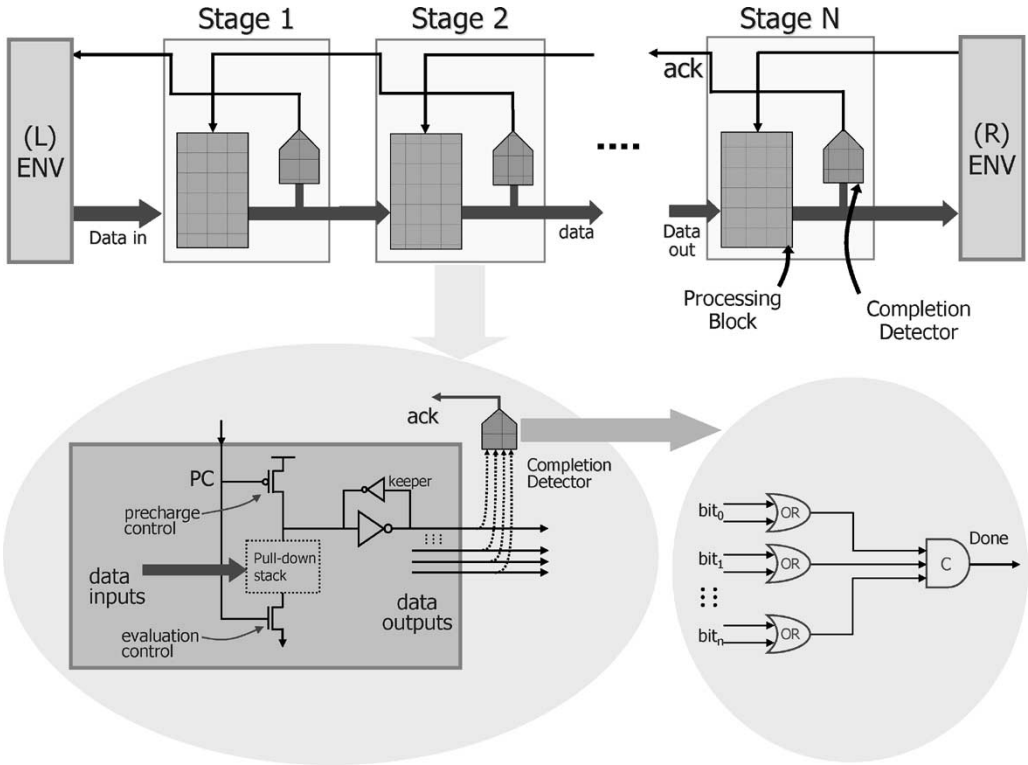


Fig. 1. Williams PS0 pipeline scheme.

achieve a desired cycle time with the fewest number of latches. In addition, like retiming, the authors do not significantly change the structure of the circuit. That is, the authors currently do not consider resynthesizing the circuit jointly with pipeline optimization. The key difference between the two problems, however, is that in the synchronous domain an initial assignment of latches must be given and the number of latches along any cycle must not be changed. In contrast, for the problem here, the initial latch assignment is not necessary and the correctness requirements on the number of latches along a cycle are different. Interestingly, the basic version of retiming can be solved in polynomial time [15].

The rest of this paper is organized as follows. Section II presents a pipeline analysis background. Section III describes the circuit model on which the authors formulate the optimization problem. Section IV then proves the NP-completeness of the problem while Section V describes a relatively efficient exact solution based on a branch and bound approach. Sections VI and VII present experimental results, conclusions, and potential directions for future work.

II. Background: Asynchronous Pipelines and Their Performance

An asynchronous pipeline is typically partitioned into sets of stages that communicate via point-to-point asynchronous channels. Each channel includes a set of wires for data in the forward direction and handshaking control signal going in the reverse direction. A deterministic pipeline is a pipeline system that does not contain choice pipeline stages that exhibit choice behavior such as arbitration, splits, and merges. This structure is formally captured in a labeled directed graph called a circuit model, in which each node is a stage, each edge represents a channel, and each node is labeled with the above set of delays. The ith stage is associated with a function evaluation and reset delay $\tau(F_i^e)$ and $\tau(F_i^r)$ that map to the latency through the stage during evaluation and reset, that is, evaluation and precharge delay of a processing block in Fig. 1. In addition, each stage is associated with completion sensing delays for evaluation and reset $\tau(D_i^e)$ and $\tau(D_i^r)$, and/or control overhead delays for evaluation and reset $\tau(C_i^e)$ and $\tau(C_i^r)$. An implementation example is depicted in Fig. 1.

Marked graphs are typically used to analyze the performance of an asynchronous circuit in terms of the above quantities [13], [21], [26], [27], [29]. A marked graph contains nodes that represent transitions of signals, edges that represent dependencies between signal transitions, and an initial marking of tokens on edges that represents the initial state of the system. For performance analysis, the signal transitions in the marked graph are associated with the corresponding delays in the circuit model. In particular, each cycle in the graph has a cycle metric that is the sum of the delays of all associated transitions divided by the number of tokens that can reside in the cycle. The cycle time of a deterministic pipeline is defined as the largest cycle metric in its marked graph representation [5], [21]. Note that the cycle time is the inverse of the throughput (measured as the average data items processed per unit time).

For efficiency and clarity, it is useful to realize that the cycles in the marked graph can be partitioned into three classes. The first corresponds to local pipeline constraints and to the interaction of neighboring pipeline stages in the circuit model.

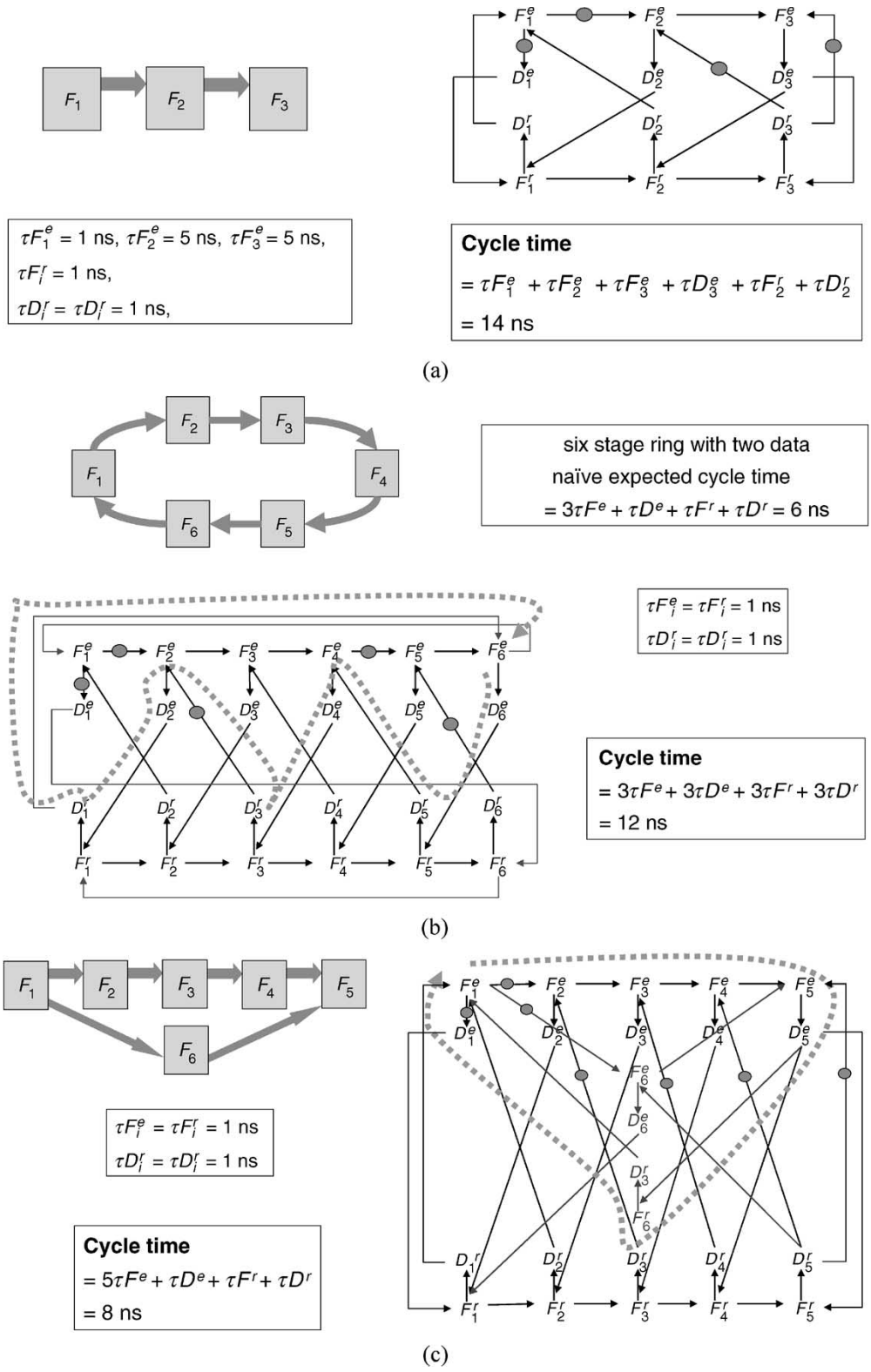


Fig. 2. Cycle time of asynchronous pipelines. (a) Local cycle time. (b) Algorithmic cycle time. (c) Fork–join cycle time.

For example, consider the simple three-stage pipeline and corresponding marked graph illustrated in Fig. 2(a).[1] This marked graph models pipelines using Williams PS0 template style. For this marked graph, there exists three one-token cycles, containing only one token, as

$$\tau(F_1^e) + \tau(F_2^e) + \tau(F_3^e) + \tau(D_3^e) + \tau(F_2^r) + \tau(D_2^r)$$
$$\tau(F_1^r) + \tau(F_2^r) + \tau(F_3^r) + \tau(D_3^r) + \tau(F_2^e) + \tau(D_2^e)$$
$$\tau(F_1^e) + \tau(F_2^e) + \tau(D_2^e) + \tau(F_1^r) + \tau(F_2^r) + \tau(D_2^r).$$

[1]Note that the places in the marked graphs are omitted for brevity.

The intuition behind the first of the three cycles is as follows. After stage 1 evaluates, stage 2 can evaluate, followed by stage 3. Once stage 3 evaluates, the results from stage 2 are no longer needed and it can precharge. Once stage 2 precharges, stage 1 can reevaluate, completing the cycle. The intuition of the other cycles is similar. The local cycle time is the maximum of these delays and, because there are no loops in the undirected version of the optimization circuit graph, this is also the overall cycle time of the circuit. More formally, the local cycle time is the maximum delay of the cycles in the marked graph that do not have corresponding cycles in the undirected version of the circuit model. For the delay parameters given in Fig. 2(a), the cycle time evaluates to 14 ns. Note that for general PS0 pipelines that contain fork-and-join, the above equations must be modified to include delays associated with additional control overhead. In fact, the authors know of no pipeline style in which the local cycle time constraint is determined by more than three pipeline stages.

The second and third classes map to loops in the undirected version of the circuit model. The second corresponding to algorithmic loop dependencies involves sequences of signal transitions that map to directed loops in the circuit model, and maps to an algorithmic cycle time. For example, in asynchronous pipeline rings that implement iterative algorithms, e.g., Williams asynchronous divider [26], [28], the cycle time may be dictated by how long it takes for a data or bubble (i.e., a single token) to travel around the ring. This concept is abstractly illustrated in the ring of six pipeline stages depicted in Fig. 2(b).

The third class, fork-join dependencies, corresponds to sequences of signal transitions that map to other undirected loops in the circuit model corresponding to alternate paths between pairs of circuit stages, i.e., fork-and-join paths. In synchronous pipelines, the number of pipeline stages along different paths between two stages must be the same to ensure correctness. In contrast, this is guaranteed for the correct operation of an asynchronous pipeline. However, widely unbalanced numbers of stages in fork-and-join paths may impact performance because one path may prevent data or bubble injection into the other path. These cycles define a fork-join cycle time that may also limit circuit performance. The optimization techniques developed in this paper focus on the general class of pipelines consisting of local, algorithmic, and fork-and-join cycles, where only sequences of up to three stages contribute local pipeline constraints. This covers general deterministic circuits using most pipeline strategies of current interest. In fact, the authors know no pipeline strategies in which the local pipeline constraint is delimited by more than three stages. In addition, the authors assert that extensions to pipeline strategies in which fewer than three [19], [20], [23] or more than three stages yield local pipeline cycles are straight forward.

III. Pipeline Optimization Model

The performance marked graph models used for analyzing pipelines assume a fixed pipeline structure and thus cannot be directly used as a model to optimize the pipeline structure itself. More specifically, a pipeline optimization model must characterize the set of possible pipeline structures. This section describes the proposed pipeline optimization model. The authors first describe a model for the most general pipeline optimization problem and then restrict the model targeted to an important subproblem called pipeline buffer optimization.

The pipeline optimization model is a labeled directed graph (S, U, M, F, L, κ) with nodes S, edges $U \subseteq S^2$, binary labels on edges $M : U \rightarrow \mathcal{B}$, and two sets of positive integer labels on nodes $F : S \rightarrow \kappa$ and $L : S \rightarrow \kappa$. The edges U represent unpartitionable combinational blocks called units. The unit u_i has function evaluation and reset delays $\tau(f_i^e)$ and $\tau(f_i^r)$, completion sensing delays for function evaluation and reset $\tau(d_i^e)$ and $\tau(d_i^r)$, as well as control overhead delays for function evaluation and reset $\tau(c_i^e)$ and $\tau(c_i^r)$.

The nodes S represent candidate boundaries between pipeline stages called slots. The labels F denote slots that have preassigned latches that delineate pipeline stage boundaries. For $F(s) \geq 1$, the first latch may be an abstract latch that does not represent any explicit storage element. In particular, many of the Williams PS0 and Caltech's style pipelines [16], [26], [27] need not have explicit latches to separate pipeline stages. Rather, each stage has an internal storage for this purpose. Consequently, the first latch in these styles simply delineates a pipeline stage boundary. In contrast, any second or additional latches must be associated with explicit storage elements. As an example, PS1 is an optimized form of PS0 where each stage is followed by an explicit storage element. In this paper, these explicit storage elements are called pipeline buffers. For convenience, the authors let κ denote the maximum number of latches assigned to any slot.

The labels M denote the edges u_i for which independent data can initially reside. The authors require that every loop in the pipeline optimization model contains at least one edge that is labeled with a data. However, loops may have multiple such labeled edges, reflecting the algorithmic intention to have multiple independent data flowing simultaneously through the circuit. Thus, more generally, the authors require that every loop in the pipeline optimization model be assigned enough abstract latches to support the number of edges u_i labeled with independent data. For example, for both Williams PS0 and PC0 schemes, the minimum number of latches to support d independent data is $2d + 1$ [25]–[27]. Also, the authors must consider terminal slots that have either no incoming or no outgoing edges. To ensure that the cycle time can be computed, the authors require that terminal slots be preassigned abstract latches. Otherwise, it is unclear how to account for the delay of units attached to terminal slots when computing the cycle time. These two conditions together ensure that the cycle time is well defined.

The function evaluation delay of stage_i is defined as $\tau(F_i^e) = \sum_{u_j \in \mathrm{stage}_i} \tau(f_j^e)$. The reset delay of stage_i is usually defined as $\tau(F_i^r) = \max_{u_j \in \mathrm{stage}_i} \tau(f_j^r)$ based on the assumption that all units within a stage reset (e.g., precharge) simultaneously. The completion sensing delays of stage_i are set to the last unit's completion sensing delay for both function evaluation and reset. The intuition here is that the completion

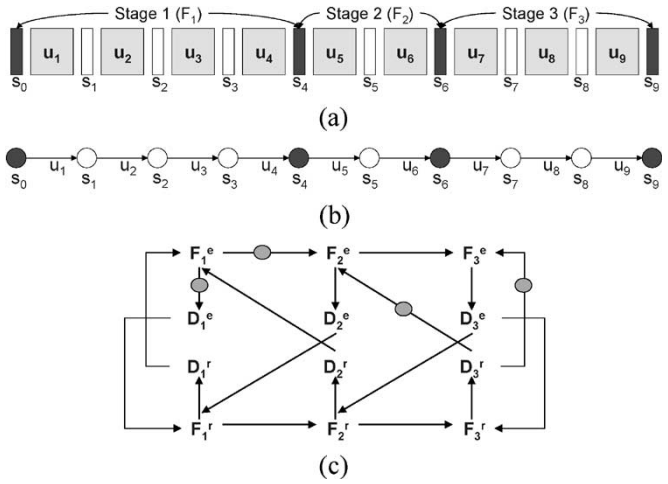


Fig. 3. Before optimization. (a) Asynchronous linear pipeline structure. (b) Pipeline optimization model. (c) Performance marked graph model.

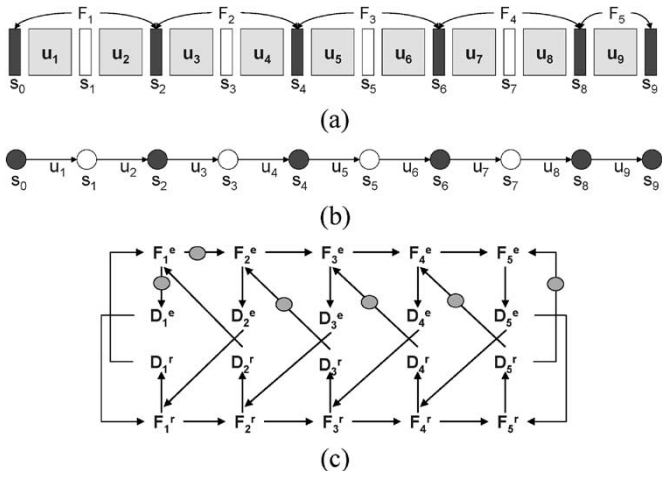
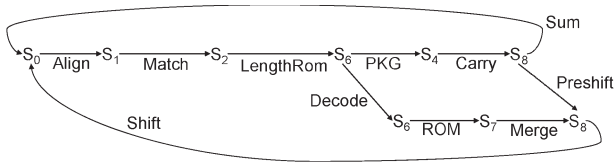


Fig. 4. After optimization. (a) Asynchronous linear pipeline structure. (b) Pipeline optimization model. (c) Performance marked graph model.

sensing units for the other units are not needed and can be discarded. Similarly, the control overhead delays of stage_i (for both function evaluation and reset) are defined as the number of input control signals to F_i and R_i.

The labels L denote number of abstract latches to be assigned to each slot, also referred to as latch assignment. The min-latch pipeline optimization problem is to find the minimum cardinality latch assignment that yields a cycle time that is well defined and less than or equal to a given constraint δ. An example of the optimization for an asynchronous linear pipeline along with changes in the associated pipeline optimization model and performance marked graph is illustrated in Figs. 3 and 4. Note that the choice of adding pipeline buffers or splitting stages is based on the design flow at hand. For example, in some high-performance design flows, the pipeline stages used are essentially at the gate level and are indivisible. Thus, the only choice is to add pipeline buffers [18]. In medium-grained performance systems, however, choosing the degree of pipelining is an important design factor [4].

Example: To make this model more concrete, consider the pipeline optimization model for a Huffman decoder [4]



	Align	Match	Length Rom	PKG	Carry	Sum	Decode	ROM	Merge	Pre-shift	Shift
$\tau(f^e)$	3	10	5	3	3	3	10	15	5	3	3
$\tau(f^r)$	2	3	0	2	2	2	3	0	2	1	1
$\tau(d^e)$	2	6	2	2	2	2	2	2	2	2	2
$\tau(d^r)$	2	6	2	2	2	2	2	2	2	2	2
$\tau(c^e)$	0	0	2	0	2	0	2	0	0	2	0
$\tau(c^r)$	0	0	2	0	2	0	2	0	0	2	0

Fig. 5. Asynchronous Huffman decoder model and its detailed delay information.

depicted in Fig. 5 using the PS0 pipeline scheme. The model decomposes the Huffman circuit into 11 units separated by nine slots and includes the estimated delays for each unit. There are three loops in this optimization model, each representing an algorithmic loop dependency. The maximum sum of the unit evaluation delays (reset evaluation delays) along any such loop represents a lower bound on the cycle time. In this case, the evaluation delays of the top loop dominate, yielding a lower bound of 46.[2]

The authors also identify an important subproblem of the pipeline optimization problem in which adding latches only to those slots already preassigned with at least one latch was considered. In other words, the authors assume that the degree of functional pipelining has already been fixed and consider only the problem of adding pipeline buffers to improve performance. In particular, the authors define pipeline buffer optimization as finding the minimum number of additional pipeline buffers required to yield a cycle time that is well defined and less than or equal to a given constraint δ.

Figs. 6 and 7 illustrate how the various forms of performance bottlenecks described in Fig. 2 can all be resolved by either splitting existing pipeline stages or adding new pipeline buffers, respectively.

IV. Complexity Analysis

Given that the basic synchronous retiming problem can be optimally solved in polynomial time [15], it seems prudent to determine the complexity of the problems here before exploring efficient algorithms. This section proves that the problems are NP-complete for the simplified pipelining performance model depicted in Fig. 8. This graph is equivalent to the more complicated marked graph in Fig. 2(a) for the special case of $\tau(f_i^r) = \tau(f^r), \tau(d_i^e) = \tau(d_i^r) = \tau(c_i^e) = \tau(c_i^r) = 0$, for all i units. The proof of NP-completeness for a variety of more complex marked graphs, including the graph depicted in Fig. 2(a), then follows directly by restriction [11]. The intuition behind

[2]Thus, our optimization problem is to find the minimum abstract latch assignment that yields a cycle time of no larger than 46.

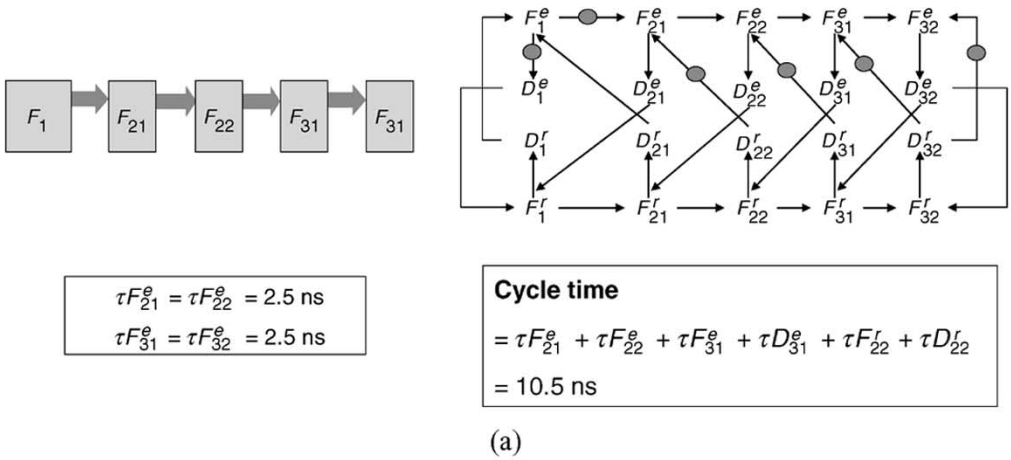


Fig. 6. Asynchronous pipeline optimization by inserting abstract latches. (a) Improving the throughput of the linear pipeline depicted in Fig. 2(a) by pipelining slow stages. (b) Improving the throughput of the small ring depicted in Fig. 2(b) by pipelining two stages, increasing the number of bubbles in the ring. (c) Improving the throughput of the fork-and-join structure in Fig. 2(c) by pipelining the short branch.

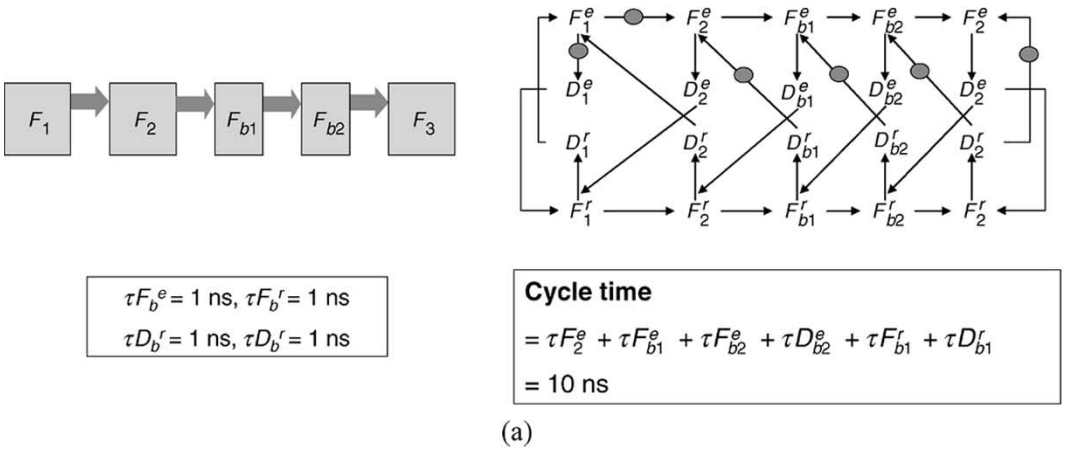


Fig. 7. Asynchronous pipeline optimization by adding pipeline buffers. (a) Improving the throughput of the linear pipeline depicted in Fig. 2(a) by adding pipeline buffers. (b) Improving the throughput of the small ring depicted in Fig. 2(b) by adding two pipeline buffers, increasing the number of bubbles in the ring. (c) Improving the throughput of the fork-and-join structure in Fig. 2(c) by adding two pipeline buffers to the short branch.

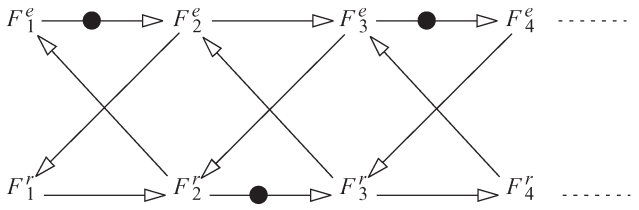


Fig. 8. Marked graph of an abstract performance model.

these results is that, in general, the number of potentially optimal pipeline configurations in an asynchronous circuit is much larger than considered by synchronous retiming for a similar-sized problem.

A. Complexity Analysis of Asynchronous Pipeline Optimization Problem

The authors define the asynchronous pipeline decision (APD) problem as the task of determining whether there exists a pipelining strategy using K or less abstract latches for which the pipeline cycle time is well defined and less than or equal to δ. This problem is proved to be NP-complete by reduction from 3-satisfiability (3SAT) problem in two steps.

First, let Z be a set of variables z_i and X be a collection of sum-of-product clauses over positive and negative literals of Z such that each clause $x_i \in X$ has $|x_i| = 3$. The 3SAT problem is a well-known problem whose task is to determine whether there exists a satisfying truth assignment for X. The complexity of the 3SAT problem has been well established.

Theorem 1—Complexity of the 3SAT [7]: 3SAT problem is NP-complete.

Consider a simplified pipeline optimization model $G = (S, U)$ without fork-and-join paths, where S is a set of slots and U is a set of units. The authors define a 3U1L assignment as the task of determining whether there exists a set of slots $S' \subset S$ with cardinality less than or equal to K, for which every terminal slot is in S' and every three consecutive unit sequence should span at least one slot in S'. The first step of the proof involves showing that the 3U1L problem is NP-complete.

To do this, the authors follow the same reduction strategy to the vertex cover problem from 3SAT [11]. It was observed that ensuring every three-unit sequence is spanned by at least one slot in S' is equivalent to ensuring that every middle unit is touched by at least one slot in S'. Mapping units to edges and slots to vertices, this is equivalent to ensuring that all middle edges must be covered by selected vertices, which is the key point behind the following proof.

Lemma 1—Complexity of 3U1L Assignment (3U1L): The 3U1L problem is NP-complete.

Proof (Sketch): First, the 3U1L problem is in NP because a modified depth-first-search algorithm can verify that every terminal slot is in S', every three-unit sequence contains a slot in S', and that S' is the appropriate size in polynomial time. To prove 3U1L is NP-hard, the authors show that the 3SAT problem can be reduced to the problem here.

The authors first construct a graph $G = (S, U)$ and a positive integer $K \leq |S|$ such that G has a 3U1L assignment with K or less latch assignment if and only if X is satisfiable. The graph consists of three different subgraphs. First, for each variable $z_i \in Z$, the authors create a truth-setting subgraph $T_i = (S_i, U_i)$ with $S_i = \{t_i, z_i, \bar{z}_i, \bar{t}_i\}$ and $U_i = \{\{t_i, z_i\}, \{z_i, \bar{z}_i\}, \{\bar{z}_i, \bar{t}_i\}\}$. For each clause $x_j \in X$, there is a satisfaction-testing subgraph $A_j = (S'_i, U'_i)$ consisting of three slots and three units joining them to form a cycle with three slots, i.e.,

$$S'_j = \{a_1[j], a_2[j], a_3[j]\}$$
$$U'_j = \{\{a_1[j], a_2[j]\}, \{a_2[j], a_3[j]\}, \{a_3[j], a_1[j]\}\}.$$

The third and last subgraph consists of only communication units and is the only subgraph that depends on which literals occur in the clauses of the 3SAT problem. For each clause $x_j \in X$, let the three literals in x_j be denoted by p_i, q_i, and r_i. Then, let the communication units of A_j be given by

$$U''_j = \{\{p_j, a_1[j]\}, \{q_j, a_2[j]\}, \{r_j, a_3[j]\}\}.$$

The instance of 3U1L is composed by setting $K = 3|Z| + 2|X|$ and $G = (S, U)$, where S is the union of all S_i and S'_j and U is the union of U_i, U'_j, and U''_j. The first subgraph is a simple di-graph that has no fork-and-join paths, the second subgraph is a ring structure, and the third subgraph generates only forks with no joins. Thus, G by construction has no fork-and-join paths. Note also that this construction clearly has polynomial time complexity.

Now, the authors show that the original 3SAT problem is satisfiable if and only if the constructed 3U1L problem is satisfiable. First, suppose that $S' \subseteq S$ is a valid solution of 3U1L for G with $|S'| \leq K$. S' must contain at least three slots from each T_i and at least two slots from each A_j. Since $K = 3|Z| + 2|X|$, however, the authors can further conclude that S' must contain exactly three slots from each T_i, two of which are terminal slots, and exactly two slots from each A_j. Note that the third (nonterminal) slot chosen in each T_i defines which variable, z_i or $\bar{z}_i$, is set to one in the solution to the 3SAT problem. To see how this truth assignment satisfies each of the clauses $x_j \in X$, consider the three units in U''_j. Exactly one of these three units must not be attached to a slot in $S' \cap A_j$ because only two of the three slots in A_j can be in S'. This slot thus must be connected to a slot z_i ($\bar{z}_i$) that is in S', which implies that the clause x_j is satisfied. For the other direction, suppose a truth assignment satisfies X. The corresponding 3U1L solution S' contains three slots from each T_i, two of which are the terminal slots and one defined by the truth assignment, and two slots from each A_j, corresponding to the slots not connected to the third (nonterminal) slot of T_i. This set of selected slots ensures that every three consecutive unit sequence has at least one selected slot. ■

Fig. 9 shows an example of the proposed constructed graph for the 3SAT problem $Z = \{z_1, z_2, z_3, z_4\}$ and $X = \{\{z_1, \bar{z}_3, \bar{z}_4\}, \{\bar{z}_1, z_2, \bar{z}_4\}\}$. For example, the 3U1L solution $S' = \{z_1, z_2, \bar{z}_3, \bar{z}_4, a_2[1], a_3[1], a_1[2], a_3[2]\}$ identifies the satisfying solution to the 3SAT problem $z_1 = 1, z_2 = 1$, $z_3 = 0$, and $z_4 = 0$.

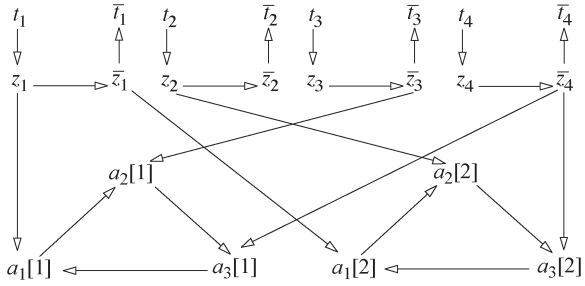


Fig. 9. 3U1L instance resulting from a 3SAT instance.

The second step of the proof requires the following useful definitions. The authors define a sequence of units to be decomposed into k stages by slot assignment if the units are part of k distinct stages (as defined by the slot assignment). A sequence of units is said to be a violating unit sequence (VUS) if the sequence must be decomposed into at least four stages in order to satisfy the cycle time constraint δ, i.e., there does not exist any slot assignment that yields a well-defined cycle time less than or equal to δ that decomposes the sequence of units into three or fewer stages. A sequence of slots is said to be a violating slot sequence (VSS) if it is spanned by a VUS, i.e., there exists a VUS that connects the sequence of slots.

Theorem 2: The local cycle time is less than or equal to δ if and only if every VUS spans parts of at least four stages, i.e., contains units in four distinct stages. In other words, the corresponding VSS must contain at least three abstract latches.

Proof (Sketch): $\Leftarrow$: The authors first prove that if every VUS spans at least four stages, the local cycle time does not violate δ. To prove this, the authors prove the equivalent statement that if the cycle time constraint δ is not satisfied by the local cycle time, there must exist a VUS that constitutes at most three stages. To see this, note that to violate δ for the local cycle time there must exist at least three consecutive stages whose cycle time is larger than δ. The sequence of units that corresponds to this sequence of stages is a VUS, thereby completing this part of the proof.

$\Rightarrow$: If local cycle time is less than or equal to δ, every VUS constitutes parts of at least four stages. This follows directly from the definition of a VUS. ■

Finally, the NP-completeness of the APD problem is proven by restricting the APD problem to $\tau(f_i^e) = 0.2$, $\delta = 0.99$, and circuit graphs that have no fork-and-join paths and a sufficient number of data tokens in every ring structure such that the algorithmic cycle time is smaller than δ.

Theorem 3: The APD problem is NP-complete.

Proof (Sketch): The authors first show that the APD problem $\in$ NP. To verify that a given solution π to the APD problem is valid, it must be verified that it has less than or equal to K slots and that it yields a circuit whose cycle time satisfies the given cycle time constraint δ. The first part involves counting the number of slots in π and the second part of the problem involves finding the longest sequence of three stage delays that can be solved using a trivially modified version of depth first search. Thus, both of these steps take polynomial time.

Next, to prove that the APD problem is NP-hard, a polynomial time algorithm that maps any instance of the 3U1L problem to an instance of the restricted APD problem is provided. First, the authors construct an APD problem instance G' from an instance of 3U1L problem. Every unit u_i in G is divided into two units $u_{i,0}$ and $u_{i,1}$ in G'. Moreover, for each new slot created, the authors create two additional slots and add units in-between the three slots to make a directed ring of size 3. Thus, G' consists of $5|U|$ units and $|S| + 3|U|$ slots. The transformation from G to G' can be done easily in polynomial time and does not introduce fork-and-join paths.

Next, the authors prove that there exists a subset of slots with cardinality less than or equal to K latches that satisfies any instance of the 3U1L problem if and only if there exists a latch assignment using $K' = K + 3|U|$ that satisfies the constructed instance of the APD problem.

Consider both directions of the if and only if condition. First, suppose there exists a latch assignment with K latches that satisfies the 3U1L problem. It was observed that a property of the construction is that every five-unit sequence in G' has a corresponding three-unit sequence in G. In particular, every five-unit sequence in the constructed graph G' consists of two newly added slots and two slots that were consecutive in G, one of which must be in the solution to the 3U1L problem. Consider the slot assignment in which, in addition to the selected latches in the 3U1L solution, every newly added slot is assigned a latch. First, notice that this assignment requires less than or equal to $K + 3|U|$ latches. Second, notice that the solution guarantees that every five-unit sequence in the constructed graph spans three latches. Thus, Theorem 2 guarantees that the local cycle time is less than or equal to δ. Since the circuit model is restricted to graphs with no fork-and-join paths, fork–join dependencies do not exist in the resulting graph. Moreover, loop dependencies in the resulting graph are satisfied by construction. Thus, the latch assignment satisfies the constructed instance of the APD problem.

Conversely, suppose there exists a satisfying latch assignment using less than or equal to $K' = K + 3|U|$ latches for an instance of the APD problem. Another property of the construction is that every three-unit sequence in G has two corresponding five-unit sequences in G'. Each corresponding five-unit sequence spans two slots that were consecutive in G and two newly created slots. Any solution to the APD problem must assign a latch to one of the consecutive slots in G. Consider the solution to the 3U1L problem created by selecting these slots in G. Each three-unit sequence in G spans a selected slot and the number of selected slots must be less than or equal to K, thereby completing the proof. ■

Fig. 10 shows an example of mapping a 3U1L problem where $K = 1$ to an APD problem where $K' = K + 3|U| = 10$. Unit u_1 in G is divided into two units $u_{1,0}$ and $u_{1,1}$ in G'. Units u_2 and u_3 are also divided in the same manner. Moreover, for each new slot created, the authors create two additional slots and add units in-between the three slots to make a directed ring of size 3. Any APD solution must have every slot in the intermediate ring assigned a latch because each intermediate ring spans five units; however, these assignments have no relation to the slot assignments of the 3U1L problem. In particular, two APD solutions exist in which, in addition to the above assignment of latches to slots, either S1 or S2 is assigned a latch (ensuring

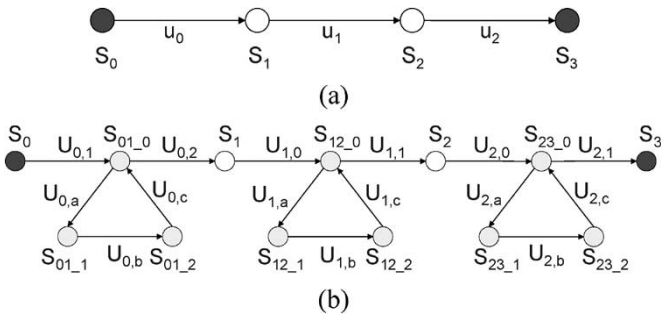


Fig. 10. Example of mapping a 3U1L problem instance to an APD problem instance.

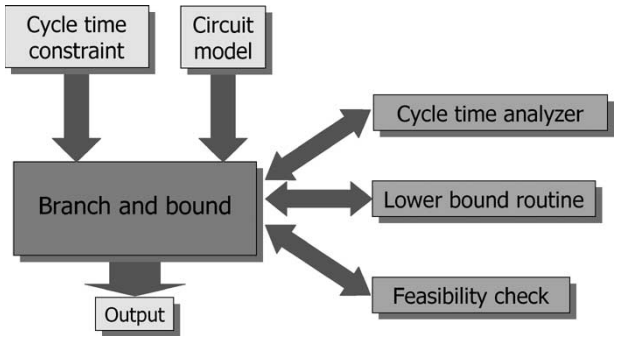


Fig. 11. Overview of APO.

the two five-unit sequences $S_0, S_{01\_0}, S_1, S_{12\_0}, S_2, S_{23\_0}$ and $S_{01\_0}, S_1, S_{12\_0}, S_2, S_{23\_0}, S_3$ both have three latches assigned to them). This suggests that the 3U1L problem also has two solutions, with either slot S1 or S2 assigned a latch. Indeed, by directly analyzing the 3U1L problem illustrated in Fig. 10(b), the authors can confirm that both assignments are valid 3U1L solutions and that there is no other solution.

B. Complexity Analysis of Pipeline Buffer Optimization Problem

First, it was assumed that $\tau(f_i^e) = \tau(f_i^r) = \tau(d_i^e) = \tau(d_i^r) = \tau(c_i^e) = \tau(c_i^r) = 0$ for all pipeline buffers. The authors define the pipeline buffer decision (PBD) problem as the task of determining whether there exists a pipeline buffer optimization strategy using K or less pipeline buffers for which the pipeline cycle time is less than or equal to δ. The authors prove that the complexity of the PBD problem is NP-complete by restriction.

Finally, the authors prove the NP-completeness of the PBD problem by restriction such that $\tau(F_i^e) = 0.2$ (which means function evaluation delay of every stage before the pipeline buffer optimization is 0.2) and $\delta = 0.59$.

Theorem 4: The PBD Problem is NP-Complete.

Proof (Sketch): By restriction. Allow a maximum of one pipeline buffer for each stage. Now the PBD problem itself is exactly a 3U1L problem since every local cycle time is 0.6 and each local cycle requires a pipeline buffer to meet δ. ■

V. Asynchronous Pipeline Optimizer (APO)

There exists a variety of techniques that may be used to solve the minimization problem. The most general technique is to cast the problem as an integer programming problem and use generic IP solvers. Alternatively, one could define a binary decision diagram (BDD) [12] describing the possible solutions for each VSS and take the product of all such BDDs. Any path through the BDD that leads to 1 represents the candidate of a valid solution, and the path with the minimal number of "1" branch represents a candidate for the minimal solution [24]. Both of these solution strategies, however, do not take advantage of the structure of the solution space and thus may be inefficient. In contrast, this section proposes an efficient branch and bound algorithm that incorporates a new lower-bound technique tailored to the problem. Moreover, the authors assert that the branch and bound algorithm is more robust than the possible BDD-based techniques because it may be terminated early to obtain a nonoptimal solution, whereas BDD-based approaches may catastrophically fail if the BDD size blows up.

The APO consists of a branch and bound framework that interacts with a cycle time analyzer to check for algorithmic constraints as well as both a lower-bound routine and a feasibility checker to prune branches off the decision tree. The overview of APO is depicted in Fig. 11. The theoretical foundation of a satisfying node is based on the notion of properly decomposed (PD) VUSs as explained in the subsequent section. Subsequent sections describe the constitute algorithms in detail.

A. Proper Decomposition of VUS

A VUS is PD by a slot assignment if the following conditions are satisfied.

Condition 1—Covering Condition: VUS is decomposed into at least four stages by the slot assignment.

Condition 2—Satisfying Condition: VUS does not contain any (complete) sequence of stages that violates δ.

Let M be a set of VUS such that every sequence of units is either a subset of a VUS in M or a superset of a VUS in M, that is, no sequence can just partially intersect or be disjoint with all VUS in M.

Lemma 2: The local cycle time is met if and only if all VUS $\in M$ are PD.

Proof (Sketch): $\Leftarrow$: Consider a three-stage sequence of units that violates the cycle time. It is either a superset or a subset of at least a VUS in M. If it is the superset of a VUS in M, it cannot be a three-stage or less sequence. (Contradiction of Condition 1.) If it is the subset of a VUS in M, it should be PD. (Contradiction of Condition 2.)

$\Rightarrow$: Proof by the definition of VUS. ■

The key theorem that identifies the optimization approach follows directly from the above lemma.

Theorem 5: If and only if all VUS $\in M$ are PD with the minimum slot assignment, then the local cycle time is met with the minimum abstract latches.

B. Branch and Bound Algorithm

The nodes in the branch and bound tree represent slots. Each node has up to two children, one representing the partial

solution in which the slot is assigned an abstract latch, referred to as a slot-assigned-child, and the other representing the partial solution in which the slot is not assigned an abstract latch, referred to as a slot-excluded-child. Each node is associated with the set of VSSs that contains that slot. Each time a new abstract latch is added to a partial solution, the authors compute the subset of associated VSSs that is PD and modify the marked graph representation. When all VSSs are PD, the marked graph is analyzed to verify that the cycle metrics associated with other dependencies are less than the cycle time constraints δ. The authors do not search the subtree rooted at a slot-assigned-child when 1) the number of abstract latches assigned up to that child node plus the derived lower bound for that subtree is larger than the current best solution; or 2) the child node represents a solution better than the current best, in which case the current best solution is updated; or 3) the cycle metrics associated with any loop dependency involving only functional evaluation delays exceed δ.[3] The authors do not search the subtree rooted at a slot-excluded-child when it is determined that there exists no feasible solution for a VSS associated with the slot.

Nodes associated with slots assigned with the least number of abstract latches are searched first and nodes associated with slots that have been excluded in a parent are never searched. The authors break ties between nodes that represent slots that have equal number of assigned abstract latches by prioritizing the slot that is associated with the most uncovered VSS computed once at the beginning of the search. The authors prioritize assigning abstract latches over pipeline buffers, thereby greedily avoiding increasing the latency of the design.

In traditional branch and bound approaches to covering problems, the MIS_QUICK independent-set-based lower-bound algorithm [12] is widely used because it is simple and fast. This algorithm is generalized to the optimization problem as follows. For each node in the branch and bound tree, the authors create a lower-bound graph consisting of a vertex for each VSS and an edge between every two VSSs that share at least one slot. Each vertex is labeled with the number of additional abstract latches needed to be assigned for the VSS to be satisfied (which, recall, is only one of two conditions to be PD). Each edge is labeled with the number of slots shared between the two VSSs. The weight of a vertex is defined as the sum of connected edge labels divided by the vertex label. The authors identify the vertex with the minimum weight and decrease all connected vertices by the minimum of the identified vertex's label and the connecting edge label. Then, they remove the identified vertex along with all connected edges and iterate. It can be easily verified that the sum of the identified vertices' labels is a lower bound of the problem. Fig. 12 shows an example of one iteration of the lower-bound heuristic.

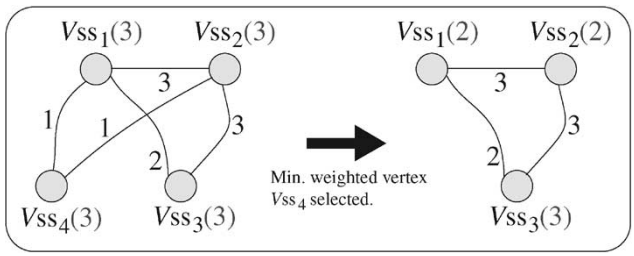


Fig. 12. Example of the lower-bound heuristic.

```
Construct di-graph{
  /* Construct di-graph for the cycle time analysis */
  for each place p_i
    create a vertex v_i
  for each place p_i
    for each place p_j
      if p_i and p_j are connected through a transition t_k
        Create edge e_ij and w_ij = d(t_k)
}
Reduce di-graph{
  /* Reduce the number of vertices in the di-graph */
  for each vertex v_i which is corresponding to unmarked edge p_i
    for each incoming vertex v_a to v_i
      for each outgoing vertex v_b from v_i
    create an edge e_ab and w_ab = w_ai + w_ib
  remove vertex v_i along with its adjacent edges
}
```

Fig. 13. Conversion procedure to di-graph for the cycle time analysis.

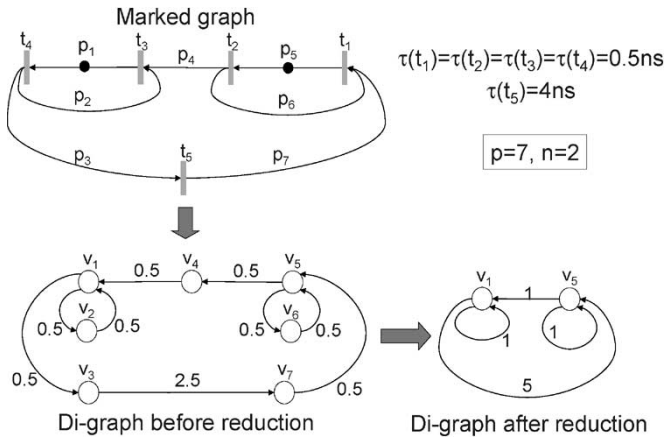


Fig. 14. Example of di-graph construction and reduction.

C. Cycle Time Analysis

This section describes an efficient method based on Karp's algorithm to find the largest cycle metric of potential solutions. The time complexity of the traditional approach proposed by [21] is $O(P^3)$ where P is the set places in the marked graph. The naive application of Karp's algorithm [9], [14] to the problem is to create a vertex for each marked place and to create edges between two vertices if there exists a path between their corresponding places. The weight of an edge is the largest sum delay of such paths. Karp's algorithm will then find the maximum mean cycle that is equivalent to the largest cycle metric. To improve the time complexity of this procedure, a new conversion procedure described in Fig. 13 is proposed.

The key idea of the procedure is to iteratively remove any vertex associated with an unmarked place by bypassing the vertex with edges from incoming vertices to outgoing vertices with a corresponding weight. Interestingly, the resulting

[3]This last condition is because additional abstract latches cannot decrease cycle metrics associated with data-limited loop dependencies.

TABLE I
EXPERIMENTAL RESULTS FOR ASYNCHRONOUS PIPELINES, RINGS, AND HUFFMAN DECODER. QUANTITIES WITH A SUBSCRIPT 1 REFER TO EXPERIMENTS WITH THE LOWER-BOUND DISABLED WHILE QUANTITIES WITH A SUBSCRIPT 2 REFER TO EXPERIMENTS WITH THE LOWER-BOUND ENABLED

Total Number of Slots	Total Number of Units	Number of PA Latches	Optimal Number of Latches	Cycle Time	Run Time$_1$ (second) / Number of Visited Branches$_1$	Run Time$_2$ (second) / Number of Visited Branches$_2$
Asynchronous Linear Pipeline (δ = 150)						
15	14	2	8	123.64	0.65 / 2281	0.55 / 1797
20	19	2	11	132.19	19.98 / 47691	18.33 / 37980
25	24	2	13	141.46	211.65 / 529141	172.46 / 317467
30	29	2	15	143.31	2532.11 / 5829341	1325.17 / 2138475
30	29	7	16	146.28	178.28 / 293358	148.77 / 216345
Asynchronous Ring (δ = 150)						
15	15	4(2)	8	142.60	0.18 / 571	0.16 / 466
20	20	6(3)	11	146.45	1.52 / 3346	1.27 / 2664
25	25	8(4)	14	144.99	16.87 / 25018	12.93 / 19346
30	30	10(5)	17	145.80	120.33 / 123979	98.7 / 97374
Asynchronous Huffman Decoder (δ = 60)						
9	11	4	5	54	0.03 / 13	0.01 / 6
Asynchronous LFSR (δ = 27, κ = 5)						
15	15	15	19	26	132.7 / 82356	87.9 / 52050

reduced di-graph may have more simple cycles[4] than the original, which at first glance may suggest that the maximum mean cycle is not preserved. However, all additional simple cycles originate from nonsimple cycles and consequently are guaranteed to have metrics no larger than the maximum mean cycle. Thus, as proven in Theorem 6, the graph reduction preserves the largest cycle metric.

Theorem 6: Any cycle C'' in the di-graph that does not correspond to a simple cycle in the marked graph is not a maximum mean cycle of the di-graph.

Proof (Sketch): Let the sum of the delay along the ith simple cycle C_i be $D(C_i)$ and the number of tokens along C_i be $T(C_i)$. Let the cycle corresponding to the largest cycle metric in the marked graph be denoted C_k.

The cycle C'' exists if and only if there exists a corresponding nonsimple cycle in the marked graph. This nonsimple cycle can be divided into multiple simple cycles $C_m, \ldots, C_n$. Then, the mean cycle of $C'' = ((D(C_m) + \cdots + D(C_n))/(T(C_m) + \cdots + T(C_n))$. Because $D(C_k)/T(C_k) > D(C_l)/T(C_l)$ $(l = m, \ldots, n)$, the mean cycle of C'' is no larger than $D(C_k)/T(C_k)$. ■

The time complexity of the proposed graph transformation is $O(P^2)$. The subsequent Karp algorithm takes $O(V \cdot E) = O(P'^3)$, where V is the set of marked places in the detailed marked graph and E is the set of edges between marked places. A simple example of the construction and reduction of di-graph is illustrated in Fig. 14. In this example, the number of vertices is two while the number of places in the original marked graph is seven.

VI. EXPERIMENTAL RESULTS

The authors have implemented the algorithm in C. To demonstrate its feasibility and limitations, it was applied to the asynchronous Huffman decoder model depicted in Fig. 5 as well as three scalable asynchronous circuit structures: a linear pipeline, a pipeline ring, and a pipelined ring-of-ring structure. The authors tested linear pipelines and pipeline rings with 15, 20, 25, and 30 slots. The last structure [linear feedback shift register (LFSR)] was tested with complicated interacting rings containing five data tokens. For all examples, the Williams PS0 pipeline scheme was chosen. For all scalable examples, the function evaluation delay, the function reset delay, the completion sensing delay for evaluation, and the completion sensing delay for reset are randomly generated between 10.0 and 30.0, 5.0 and 15.0, 1.0 and 20.0, and 1.0 and 10.0, respectively.

Tables I and II show the experimental results of the algorithm with and without the lower-bound algorithm (presented in Section V-B) enabled. When the lower-bound algorithm is enabled, the run time is cut by half. The results demonstrate that using the lower-bound algorithm, the optimal pipeline configuration for moderately sized problem is feasible. It is also important to note that for large systems, the run time can be reduced by either removing slots from consideration or preassigning slots with abstract latches. For instance, the authors ran additional experiments where for each structure they preassigned several selected slots with abstract latches. As shown in Tables I and II, the run times are significantly reduced.

Table III shows the interesting results of asynchronous pipeline buffer matching. To achieve the local cycle time (20), adding only a few fast (with $\tau(f^e) = \tau(f^r) = 2$) pipeline buffers is required. In particular, an initial fork-and-join path with 17 and three stages in their respective branches was optimally balanced by adding only eight fast pipeline buffers to the short branch.

VII. CONCLUSION

This paper formalizes a new asynchronous pipeline optimization problem common to a variety of pipelining styles and proves that it is NP-complete. It then proposes an efficient branch and bound algorithm for the exact solution. The experimental results suggest that the algorithm is feasible for

[4]For a graph $G = (V, E)$, a simple cycle is a sequence $\langle v_1, v_2, , v_k \rangle$ of distinct vertices from V such that $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq k$ and such that $\{v_k, v_1\} \in E$.

TABLE II
EXPERIMENTAL RESULTS FOR ASYNCHRONOUS PIPELINES, RINGS, AND HUFFMAN DECODER. QUANTITIES WITH A SUBSCRIPT 1 REFER TO EXPERIMENTS WITH THE LOWER-BOUND DISABLED WHILE QUANTITIES WITH A SUBSCRIPT 2 REFER TO EXPERIMENTS WITH THE LOWER-BOUND ENABLED

Total Number of Slots	Total Number of Units	Number of PA Latches	Optimal Number of Latches	Cycle Time	Run Time$_1$ (second) / Number of Visited Branches$_1$	Run Time$_2$ (second) / Number of Visited Branches$_2$
Asynchronous Linear Pipeline ($\delta = 120$)						
15	14	2	10	117.10	0.73 / 1696	0.76 / 1589
20	19	2	13	119.99	6.75 / 16748	5.95 / 11155
25	24	2	16	119.98	63.85 / 119240	57.81 / 80350
30	29	2	19	119.98	604.02 / 1000494	551.37 / 534454
30	29	7	20	118.29	105.90 / 149800	99.46 / 87308
Asynchronous Ring ($\delta = 120$)						
15	15	4(2)	-	-	-	-
20	20	6(3)	-	-	-	-
25	25	8(4)	17	118.14	16.23 / 18821	15.15 / 16182
30	30	10(5)	21	117.08	79.95 / 76839	78.95 / 60704
Asynchronous Huffman Decoder ($\delta = 46$)						
9	11	4	6	46	0.05 / 18	0.01 / 8
Asynchronous LFSR ($\delta = 22$, $\kappa = 5$)						
15	15	15	20	21.6	1124.71 / 524417	857.13 / 374152

TABLE III
EXPERIMENTAL RESULTS FOR ASYNCHRONOUS FORK-AND-JOIN PIPELINES TO DEMONSTRATE PIPELINE BUFFER OPTIMIZATION

Initial Number of Stages$_A$	Initial Number of Stages$_B$	Optimal Number of Stages$_B$	Cycle Time	Run Time (second)	Optimal Number of Stages$_B$	Cycle Time	Run Time (second)
		$\delta = 27$			$\delta = 25$		
8	2	3	26	0.07	4	22.5	0.45
13	2	4	26.5	0.55	5	22.8	3.84
17	3	6	26.4	37.78	7	24	132.85
		$\delta = 23$			$\delta = 20$		
8	2	4	22.5	0.42	6	20	4.4
13	2	5	22.8	6.06	7	20	20.5
17	3	9	21.71	715.37	11	20	2125.48

moderately sized systems. Moreover, complexity reduction methods for its application to larger systems are also presented and evaluated. However, the approach is not practical for very large systems and heuristics practical for large systems or ones that enable hierarchical analysis of large systems are an important area of future research.

Although the algorithm as described is restricted to models that do not exhibit choice, the approach can also heuristically be applied to systems with choice modeled by, e.g., free-choice Petri nets. The idea is to sequentially apply the algorithm to distinct choice-free behaviors (e.g., marked graph components) from those with the highest probability to those with the lowest probability. Specifically, the abstract latches assigned in one iteration would be assumed preassigned for the remainder of the optimization process. Other more effective strategies may also be possible and are an interesting area of future research. In addition, extensions that allow stochastic delays may also be possible and useful.

Slack optimization is the pipeline buffer optimization subproblem in which the design is guaranteed to meet the local cycle time constraints. This problem is particularly important for fine-grain pipeline design styles for which manual decomposition can readily guarantee local and algorithmic cycles times, but identifying the optimal locations for additional pipeline buffers to satisfy fork and join constraints is more challenging. Thus, important areas of future research include determining if this problem is also NP-complete and developing targeted algorithms that tradeoff complexity for optimality.

ACKNOWLEDGMENT

The authors would like to thank A. Kondratyev at Cadence Berkeley Labs for his invaluable comments on an earlier version of this manuscript.

REFERENCES

[1] V. Akella and G. Gopalakrishnan, "SHILPA: A high-level synthesis system for self-timed circuits," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*. Santa Clara, CA: IEEE Comput. Society Press, Nov. 1992, pp. 587–591.

[2] B. Bachman, H. Zheng, and C. J. Myers, "Architectural synthesis of timed asynchronous systems," in *Proc. Int. Conf. Computer Design (ICCD)*. Austin, TX: IEEE Comput. Society Press, Oct. 1999, pp. 354–363.

[3] R. M. Badia and J. Cortadella, "High-level synthesis of asynchronous systems: Scheduling and process synchronization," in *Proc. European Conf. Design Automation (EDAC)*. Paris, France: IEEE Comput. Society Press, Feb. 1993, pp. 70–74.

[4] M. Benes, S. M. Nowick, and A. Wolfe, "A fast asynchronous Huffman decoder for compressed-code embedded processors," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits and Systems*, San Diego, CA, 1998, pp. 43–56.

[5] S. M. Burns, "Performance analysis and optimization of asynchronous Circuits," Ph.D. dissertation, Comput. Sci. Dept., California Inst. Technol., Pasadena, 1991.

[6] W.-C. Chou, P. A. Beerel, and K. Y. Yun, "Average-case technology mapping of asynchronous burst-mode circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 10, pp. 1418–1434, Oct. 1999.

[7] S. A. Cook, "The complexity of theorem-proving procedures," in *Proc. 3rd Annu. ACM Symp. Theory Computing*. Shaker Heights, OH: ACM Press, 1971, pp. 151–158.

[8] U. Cummings, A. Lines, and A. Martin, "An asynchronous pipelined lattice structure filter," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits and Systems*, Salt Lake City, UT, Nov. 1994, pp. 126–133.

[9] A. Dasdan and R. K. Gupta, "Faster maximum and minimum mean cycle algorithms for system performance analysis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 10, pp. 889–899, Oct. 1998.

[10] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Trans. Very Large Scale (VLSI) Integr. Syst.*, vol. 4, no. 2, pp. 247–253, Jun. 1996.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[12] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Boston, MA: Kluwer, 1996.

[13] H. Hulgaard, "Timing analysis and verification of timed asynchronous circuits," Ph.D. dissertation, Dept. Comput. Sci., Univ. Washington, Seattle, 1995.

[14] R. M. Karp, "A characterization of the minimum cycle mean in a diagraph," *Discrete Math.*, vol. 23, no. 3, pp. 309–311, 1978.

[15] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI Comput. Syst.*, vol. 1, no. 1, pp. 41–67, 1983.

[16] A. M. Lines, "Pipelined asynchronous circuits," Master's thesis, Comput. Sci. Dept., California Inst. Tech., Pasadena, 1996.

[17] R. Manohar and A. J. Martin, "Slack elasticity in concurrent computing," in *Proc. 4th Int. Conf. Mathematics Program Construction*, J. Jeuring, Ed., Marstrand, Sweden, 1998, vol. 1422, pp.272–285.

[18] A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, "The design of an asynchronous MIPS R3000 microprocessor," in *Proc. Conf. Advanced Research VLSI*, Ann Arbor, MI, Sep. 1997, pp. 164–181.

[19] R. O. Ozdag and P. A. Beerel, "High-speed qdi asynchronous pipelines," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits and Systems*. Manchester, U.K.: IEEE Comput. Soc. Press, Apr. 2002, pp. 13–22.

[20] R. O. Ozdag, M. Singh, P. A. Beerel, and S. M. Nowick, "High-speed non-linear asynchronous pipelines," in *Proc. Design, Automation and Test Europe (DATE)*, Paris, France, Mar. 2002, pp. 1000–1007.

[21] C. V. Ramamoorthy and G. S. Ho, "Performance evaluation of asynchronous concurrent systems using Petri nets," *IEEE Trans. Softw. Eng.*, vol. 6, no. 5, pp. 440–449, Sep. 1980.

[22] N. Shenoy, "Retiming: Theory and practice," *VLSI J. Integr.*, vol. 22, no. 1–2, pp. 1–21, 1997.

[23] M. Singh and S. M. Nowick, "High-throughput asynchronous pipelines for fine-grain dynamic datapaths," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits and Systems*. Eilat, Israel: IEEE Comput. Soc. Press, Apr. 2000, pp. 198–209.

[24] F. Somenzi, private communications, 1999, F. Somenzi is a professor of computer science at the University of Colorado.

[25] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures," *VLSI J. Integr.*, vol. 15, no. 3, pp. 313–340, Oct. 1993.

[26] T. E. Williams, "Self-timed rings and their application to division," Ph.D. thesis, Dept. Elect. Eng. Comput. Sci., Stanford Univ., CA, Jun. 1991.

[27] ——, "Analyzing and improving the latency and throughput performance of self-timed pipelines and rings," in *Proc. Int. Symp. Circuits and Systems*, San Diego, CA, May 1992, pp. 665–667.

[28] T. E. Williams and M. A. Horowitz, "A zero-overhead self-timed 160 ns 54 b CMOS divider," *IEEE J. Solid-State Circuits*, vol. 26, no. 11, pp. 1651–1661, Nov. 1991.

[29] A. Xie, S. Kim, and P. A. Beerel, "Bounding average time separations of events in stochastic timed Petri nets with choice," in *Proc. Int. Symp. Advanced Research Asynchronous Circuits and Systems*, Barcelona, Spain, Apr. 1999, pp. 94–107.

Sangyun Kim (S'97–M'04) received the B.S. degree in electronic engineering from Yonsei University, Seoul, Korea, in 1994, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California, Los Angeles, CA, in 1997 and 2003, respectively.

In 2003, he joined Synopsys, Inc, Hillsboro, OR, as a Senior R&D Engineer. His research interests include physical synthesis, performance analysis, and asynchronous circuits design.

Peter A. Beerel (S'88–M'95) received the B.S.E. degree in electrical engineering from Princeton University, Princeton, NJ, in 1989, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1991 and 1994, respectively.

He has consulted for Yuni Networks, San Diego, CA, and Applied Micro Circuits Corporation (AMCC), San Diego, CA, in the areas of networking chip design; Intel Corporation, Haifa, Israel, in the areas of asynchronous design and CAD; and TrellisWare Technologies, San Diego, CA, in the area of communication chip design. In addition, he was the Vice-President of Asynchronous CAD and Verification at Fulcrum Microsystems, Inc., Calabasas, CA, a startup company developing and commercializing asynchronous designs while on leave of absence from USC. He is currently an Associate Professor in the Department of Electrical Engineering Systems at the University of Southern California (USC), Los Angeles, CA. He was the coauthor of eight patents in the area of very large scale integration (VLSI) and computer-aided design (CAD). His research interests include a variety of topics in CAD and VLSI.

Dr. Beerel has been a member of the Technical Program Committee for the International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC) since 1997, was Program Co-Chair for ASYNC'98, and is currently on its steering committee. He has also served on the Technical Program Committee for ICCAD'00 and ICCAD'01. He was the recipient of an Outstanding Teaching Award in 1997 and the Junior Research Award in 1998, both from USC's School of Engineering. He received a National Science Foundation (NSF) Career Award and a 1995 Zumberge Fellowship. He was also co-winner of the Charles E. Molnar Award for two papers published in ASYNC'97 that best bridged theory and practice of asynchronous system design and was a co-recipient of the best paper award in ASYNC'99.