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

Sangyun Kim   Peter A. Beerel

Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-2562
{sangyunk, pabeerel}@eiger.usc.edu

Abstract

This paper addresses the problem of identifying the minimal 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. In contrast to the somewhat analogous problem of *retiming* in the synchronous domain, we first show that the basic pipeline optimization problem for asynchronous circuits is NP-complete. This paper then presents an efficient branch and bound algorithm that can find the optimal pipeline configuration for moderately-sized problems. Our experimental results on a few scalable system models demonstrate that our novel branch and bound solver can find the optimal pipeline configuration for models that have up to 2^{35} possible pipeline configurations.

1 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, and improved noise immunity and electromagnetic compatability. Many tools and techniques have been developed to address hazard-freedom and area minimization. Estimation and optimization of their performance, however, remains 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 performance optimization of asynchronous circuits. The first approach involves using performance analysis techniques to guide manual or semi-automated design changes (e.g., [16]). 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., [3, 2, 1]) in high-level synthesis.

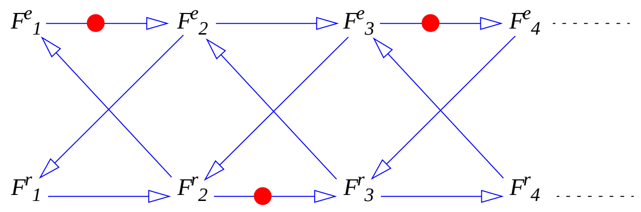
This paper formalizes a new performance optimization area for asynchronous circuits called *pipeline optimization*. In particular, previous research is either at a much lower level than pipelining (e.g., logic synthesis) or assumes that the pipelining is fixed (e.g., in high-level synthesis). *More specifically, to the best of our 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., [17, 16]), 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 this optimization problem is getting more difficult. Namely, Caltech researchers et al. propose partitioning asynchronous data-paths into bit-slices and pipelining between bit-slices to achieve higher throughput [12, 7]. When combined with standard pipelining between functional component boundaries, this creates a complex 2-dimensional 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.

It may be worth pointing out similarities with a somewhat analogous problem of *retiming* [14] in the domain of synchronous circuits. In particular, like our problem, one basic version of retiming is to achieve a desired cycle time with the fewest number of latches. In addition, like retiming, we do not significantly change the structure of the circuit. That is, we currently *do not* consider re-synthesizing 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 our problem, the initial latch assignment is not necessary and the correctness requirements on the number of latches along a cycle are different.

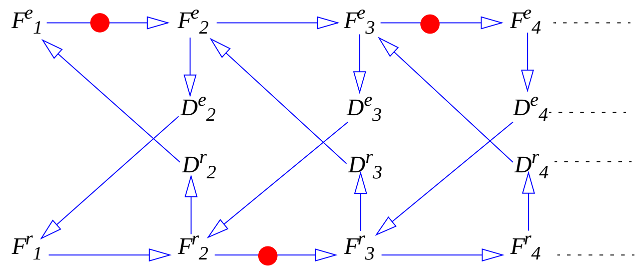
This paper first proposes an abstract model of the circuit on which the basic pipeline optimization problem can be defined. This abstract model is sufficient to characterize a variety of pipelining schemes, including those from Williams and Caltech [17, 12]. However, it is currently restricted to deterministic pipelines (no-choice) and only considers fixed delays. Given that the basic synchronous retiming problem can be optimally solved in polynomial time [11], we first explored the complexity of our optimization problem. One contribution of this paper is a proof that the defined asynchronous pipeline optimization problem is NP-complete. In addition, we present an efficient branch and bound algorithm which demonstrates the feasibility of the optimization problem for moderately-sized models. Our experimental results on a few scalable models of asynchronous systems that our branch and bound solver can successfully find the optimal solution among over 2^{35} pipeline configuration.

The organization of the remainder of this paper is as follows. Section 2 presents pipeline analysis background and Section 3 describes the model on which we formulate the optimization problem. Section 4 then proves NP-completeness of our problem while Section 5 describes an relatively efficient exact solution based on a branch and bound approach. Sections 6 and 7 present experimental results, conclusions, and potential directions for future work.

[†]This work is funded in part by NSF Grant CCR-9812164.



(a) Abstract model



(b) PS0 model

Figure 1: Marked graph models of asynchronous pipelines.

2 Background: Asynchronous Pipelines

Previous work related to performance of asynchronous pipelines have focused on assuming a given structure of an asynchronous pipeline and analyzing its performance. For example, a deterministic pipeline is generally partitioned into a set of *stages* each controlled by a different control signal. The i^{th} stage is associated with a *function evaluation delay* $\tau(F_i^e)$, a *function reset delay* $\tau(F_i^r)$, a *completion sensing delay for evaluation* $\tau(D_i^e)$, a *completion sensing delay for reset* $\tau(D_i^r)$, a *control overhead delay for evaluation* $\tau(C_i^e)$, and a *control overhead delay for reset* $\tau(C_i^r)$. Marked graphs are typically used to analyze the interaction of neighboring stages in terms of the above quantities [17, 10, 18, 13].[1] 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, 13].

The largest cycle metric in the marked graph either arises from pipelining constraints or from *algorithmic loop dependences.* For example, in asynchronous pipeline rings which implement iterative algorithms, e.g., Williams' asynchronous divider [17], 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.

We first consider the marked graph illustrated in Figure 1(a).[2] This marked graph abstractly models pipelines using both Williams style PC0 and PS0 scheme [17] as well as some of Caltech's precharge-logic pipelining schemes [12]. For this marked graph, there exists three *one-token* cycles, containing only one-token, for every sequence of three pipeline stages as follows:

$$\max\Big(\tau(F_i^e)+\tau(F_{i+1}^e)+\tau(F_{i+2}^e)+\tau(F_{i+1}^r),\\ \tau(F_i^r)+\tau(F_{i+1}^r)+\tau(F_{i+2}^r)+\tau(F_{i+1}^e),\\ \tau(F_i^e)+\tau(F_{i+1}^e)+\tau(F_i^r)+\tau(F_{i+1}^r)\Big)$$

[1]Due to space limitations, we refer the interested reader to [13] for a formal introduction to marked graphs and their application to performance analysis.

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

As an example, the intuition behind the first of the three cycles is as follows. After stage i evaluates, stage $i+1$ can evaluate, followed by stage $i+2$. Once stage $i+2$ evaluates, the results from stage $i+1$ are no longer needed and it can precharge. Once stage $i+1$ pre-charges, stage i can re-evaluate, completing the cycle. The *cycle time* is lower bounded by the maximum of the above quantity for each three-pipeline-stage sequence. More specifically, the cycle time is the maximum of this lower bound and the cycle metrics associated with all loop dependencies.

Note that the marked graph in Figure 1(a) is general but ignores the control and completion sensing overheads. In contrast, the marked graph in Figure 1(b) illustrates a more detailed model of a specific pipeline style, namely Williams' PS0 pipeline scheme. For this marked graph, a sequence of three stages yields the following three one-token cycles:

$$\max\Big(\tau(F_i^e)+\tau(F_{i+1}^e)+\tau(F_{i+2}^e)+\tau(D_{i+2}^e)+\tau(F_{i+1}^r)+\tau(D_{i+1}^r),\\ \tau(F_i^r)+\tau(F_{i+1}^r)+\tau(F_{i+2}^r)+\tau(D_{i+2}^r)+\tau(F_{i+1}^e)+\tau(D_{i+1}^e),\\ \tau(F_i^e)+\tau(F_{i+1}^e)+\tau(D_{i+1}^e)+\tau(F_i^r)+\tau(F_{i+1}^r)+\tau(D_{i+1}^r)\Big)$$

For general PSO pipelines that contain forks and joins the above equations must be modified to include control circuit overhead.

The optimization techniques developed in this paper focus on the general class of pipelines in which each sequence of 3 stages contributes some number of one-token cycles which covers most pipeline strategies of current interest. We assert, however, that extensions to pipeline strategies in which fewer than 3 or more than 3 stages yield one-token cycles are straight-forward.

3 Pipeline Optimization Model

The abstract circuit 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 our proposed model.

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

The nodes S represent candidate boundaries between pipeline stages called *slots*. The labels F denote slots which have preassigned abstract latches that delineates pipeline stage boundaries.

The labels L denote which slots are *to be* assigned *abstract latches*. Note that the presence of the latch changes the implied control structure of the circuit but does not necessarily represent a physical logic entity. In particular, note that many of the Williams style pipeline [17] need not have explicit latches. In particular, the set of combinational logic blocks in between two slots that are assigned abstract latches is one *stage*. An example of asynchronous linear pipeline is in Figure 2. Note that introducing more than one slot in between stages (by adding a fictitious functional units with zero delay) facilitates the introduction of explicit latches to further increase throughput (such as in PC1 and PS1 [17]).

The labels M denote the edges u_i for which independent data can initially reside. We require that every loop in the pipeline optimization model contain at least one edge that is labeled with a data.

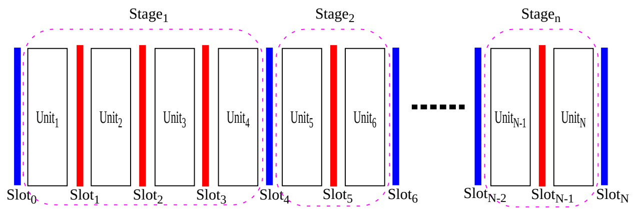


Figure 2: Our optimization model of an asynchronous linear pipeline.

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, we 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 abstract latches to support d independent data is $2d+1$ [17, 16]. Also, we must consider *terminal* slots that have either no incoming or no outgoing edges. To ensure the cycle time can be computed, we require that terminal slots be pre-assigned 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 the cycle time is *well-defined*.

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

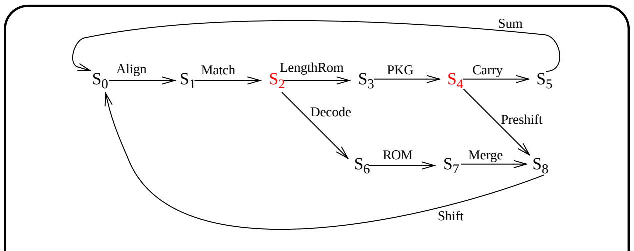
The output of the optimization problem is a subset of slots to be assigned abstract latches and is referred to as an *abstract latch assignment*. Thus, the *min-abstract-latch pipeline optimization problem* is to find a minimum cardinality abstract latch assignment that yields a cycle time that is well-defined and less than or equal to a given constraint δ.

Example To make this model more concrete, consider the pipeline optimization model for a Huffman decoder [4] depicted in Figure 3 using the PS0 pipeline scheme. The model decomposes the Huffman circuit into 11 units separated by 9 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 dominates, yielding a lower bound of 46.[3] ■

4 Complexity Analysis

Given that the basic synchronous retiming problem can be optimally solved in polynomial time [11], it seems prudent to determine the complexity of our problem before exploring efficient algorithms. This section proves that our problem is NP-complete for the simplified pipelining performance model depicted in Figure 1(a). This graph is equivalent to the more complicated marked

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



	Align	Match	Length Rom	PKG	Carry	Sum	Decode	ROM	Merge	PreShift	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
$\tau(c^f)$	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

Figure 3: An asynchronous Huffman decoder model and its detailed delay information.

graph in Figure1(b) 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. Lastly, we assume that the given cycle time constraint δ is larger than cycle metrics associated with loop dependencies, which for this simplified dependency graph model, is independent of the degree of pipelining. The proof of NP-completeness for a variety of more complex marked graphs, including the graph depicted in Figure 1(b), then follows directly by *restriction* [8]. The intuition behind 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.

We 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. We prove this problem is NP-complete by reduction to 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 $|x_i| = 3$ [8]. The 3-Satisfiability (3SAT) problem is a well known problem whose task is determine whether there exists a satisfying truth assignment for X. The complexity of the 3SAT problem has been well established:

Theorem 1 Complexity of 3-Satisfiability (3SAT) *[8]*
3SAT problem is NP-complete.

Consider a simplified pipeline optimization model $G = (S, U)$, where S is a set of slots, U is a set of units, and no cycle consists of less than three slots. We define a *3U1L assignment* as the task of determining whether there exist 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 our proof involves showing that the 3U1L problem is NP-complete.

To do this, we follow the same reduction strategy to 3SAT from the vertex cover problem [8]. We 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 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, we show that our problem can be reduced to the 3SAT problem which is known to be NP-complete.

We 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 $z_i \in Z$, we create a *truth-setting* subgraph $T_i = (S_i,U_i)$ with $S_i = \{t_i, z_i, \overline{z_i}, \overline{t_i}\}$ and $U_i = \{\{t_i,z_i\},\{z_i,\overline{z_i}\},\{\overline{z_i},\overline{t_i}\}\}$. 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.

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

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 construction of our instance of 3U1L is composed by setting $K = 3|Z| + 2|X|$ and $G = (S,U)$ where S is an union of all S_i and S'_j and U is an union of U_i, U'_j and U''_j. Note, that this construction clearly has polynomial time complexity.

Now, we 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 slot from each T_i and *at least* two slots from each A_j. Since $K = 3|Z| + 2|X|$, however, we 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 (non-terminal) slot chosen in each T_i defines which variable, z_i or $\overline{z_i}$, 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 ($\overline{z_i}$) 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 (non-terminal) slot of T_i. This set of selected slots ensures that every three consecutive unit sequence has at least one selected slot. ■

Figure 4 shows an example of the proposed constructed graph for the 3SAT problem $Z = \{z_1,z_2,z_3,z_4\}$ and $X = \{\{z_1,\overline{z_3},\overline{z_4}\}, \{\overline{z_1},z_2,\overline{z_4}\}\}$. For example, the 3U1L solution $S' = \{z_1,z_2,\overline{z_3},\overline{z_4}, a_2[1],a_3[1],a_1[2],a_3[2]\}$ identifies the satisfying solution to the 3SAT problem $z_1 = 1, z_2 = 1, z_3 = 0$, and $z_4 = 0$.

The second step of the proof requires the following useful definitions. We define a sequence of units to be *decomposed* into k stages by a slot assignment if the units are part of k distinct stages (as defined by the slot assignment). We say a sequence of units is a *violating unit sequence (VUS)* if the sequence must be decomposed into

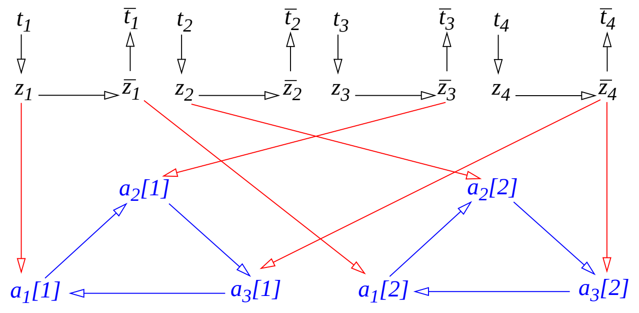


Figure 4: A 3U1L instance resulting from a 3SAT instance.

at least 4 stages in order to satisfy the cycle time constraint, δ, i.e., there doesn't exist any slot assignment that yields a well-defined cycle time less than or equal to δ that decomposes the sequence of units into 3 or fewer stages. We say a sequence of slots is a *violating slot sequence (VSS)* if it is spanned by a VUS, i.e., their exists a VUS that connects the sequence of slots.

Theorem 2 *The pipeline cycle time is less than or equal to δ if and only if every VUS spans parts of at least 4 stages, i.e., contains units in 4 distinct stages. In other words, the corresponding VSS must contain at least 3 abstract latches.*

Proof (*Sketch*) $\Leftarrow$: We first prove that if every VUS spans at least 4 stages, the cycle time constraint δ is satisfied. To prove this, we prove the equivalent statement that if the cycle time constraint δ is not satisfied, there must exist a VUS which constitutes at most 3 stages. To see this, note that to violate δ, there must exist at least three consecutive stages whose cycle time is larger than δ. The sequence of units that correspond to this sequence of stages is a VUS, thereby completing this part of the proof.
$\Rightarrow$: If cycle time constraint δ is satisfied, every VUS constitutes parts of at least 4 stages. We prove the above statement by contradiction. Assume that that cycle time constraints δ is satisfied but that there exists a VUS with three or less stages. By the definition of pipeline cycle time, this VUS however implies that δ is violated, a contradiction. ■

Finally, we prove NP-completeness of APD problem by restricting the APD problem such that $\tau(f_i^e) = 0.2$ and $\delta = 0.99$ and showing a reduction to the 3U1L problem.

Theorem 3 *The APD problem is NP-complete.*

Proof (*Sketch*) We first show that APD problem $\in NP$. To verify that a given solution π to the APD problem is valid, we must verify 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 which can be solved using a trivially modified version of depth first search. Thus, both of these steps take polynomial time.

Next, to prove the APD problem is NP-hard, we provide a polynomial-time algorithm that maps any instance of the 3U1L problem to an instance of the APD problem. First, we construct an APD problem instance G' from an instance of 3U1L problem. Every unit u_i in G is divided into two unit $u_{i,1}$ and $u_{i,2}$ in G'. Moreover, for each new slot created, we 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.

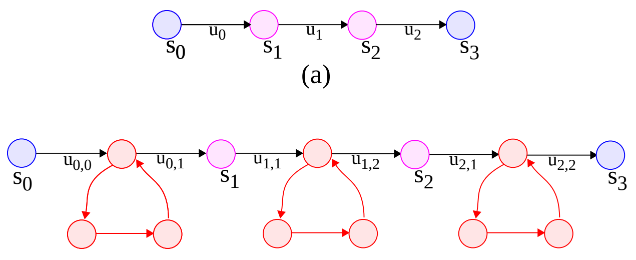


Figure 5: An example of mapping a 3U1L problem instance to an APD problem instance.

Next, we 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.

Lets 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. We observe that a property of our construction is that every five unit sequence in G' has a corresponding 3 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 solution guarantees that every five unit sequence in the constructed graph spans three latches, that the cycle time is well-defined and is less than or equal to δ.

Conversely, suppose there exists a satisfying latch assignment using less than or equal to $K' = K + 3|U|$ latches for an instance of APD problem. Another property of our construction is that every three unit sequence in G has two corresponding five unit sequences in G'. Each corresponding five unit sequences 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. ■

An example of mapping a 3U1L problem to an APD problem is depicted in Figure 5.

5 Proper Decomposition of Violating Unit Sequence

To solve the general optimization problem, we first introduce the following definitions. A VUS is *Properly Decomposed (PD)* by a slot assignment if the following conditions are satisfied:

Condition 1 Covering condition*: The VUS is decomposed into at least* 4 *stages by the slot assignment.*

Condition 2 Satisfying condition*: The VUS does not contain any (complete) sequence of stages which violate* 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 disjoint with all VUS in M.

Lemma 2 *The cycle time is met if and only if all VUS* $\in M$ *are properly decomposed.*

Proof (*Sketch*) $\Leftarrow$: Consider a 3-stage sequence of units, which 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 can't be a 3-stage or less sequence. (Contradiction of the condition 1). If it is the subset of a VUS in M, it should be properly decomposed. (Contradiction of the condition 2).
$\Rightarrow$: Proof by the definition of VUS. ■

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

Theorem 4 *If and only if all VUS* $\in M$ *are properly decomposed with the minimum slot assignment, then the cycle time is met with the minimum abstract latches.*

6 Branch and Bound Algorithm

There exist a variety of techniques that may be used to solve our 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 BDD describing the possible solutions for each VSS and take the product of all such BDDs. Any path through the BDD that leads to one represents a valid solution and the path with the minimal number of "1" branches, represents a minimal solution [15]. 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 our problem. Moreover, we assert that our branch and bound algorithm is more robust than possible BDD-based techniques because it may be terminated early to obtain a non-optimal solution whereas BDD-based approaches may catastrophically fail if the BDD-size blows up.

The nodes in our 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 contain that slot. Each time a new abstract latch is added to a partial solution we compute the subset of associated VSSs that are properly decomposed. We do not search the subtree routed 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 or equal to 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 exceeds δ.[4] We do not search the subtree routed at a slot-excluded-04 when we determine there exist no feasible solution for a VSS associated with the slot.

In the traditional branch and bound approaches to covering problems, the MIS_QUICK independent-set-based lower bound algorithm [9] is widely used because its simple and fast. We generalize this algorithm to our optimization problem as follows. For each node in the branch and bound tree, we 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 properly decomposed). Each edge is labeled with the number of

[4]This last condition is because additional abstract latches cannot decrease cycle metrics associated with loop dependencies.

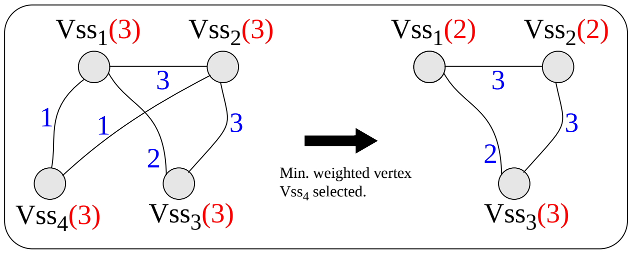


Figure 6: An example of the lower bound heuristic.

slots shared between the two VSSs. We define the *weight* of a vertex as the sum of connected edge labels divided by the vertex label. We 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. We then remove the identified vertex along with all connected edges and iterate. It can be easily verified that the sum of the identified verticies' labels is a lower bound of our problem. Figure 6 shows an example of one iteration of our lower bound heuristic.

7 Experimental results

We have implemented a prototype of our algorithm in C. To demonstrate its feasibility and limitations, we applied it to the asynchronous Huffman decoder model depicted in Figure 3 as well as three scalable asynchronous circuit structures, a linear pipeline, a pipeline ring, and a pipelined ring-of-ring structure. We tested linear pipelines and pipeline rings with 20, 25, 30 and 35 slots. The last structure (ring-of-rings) we tested with 5 rings, each ring containing 10 slots, with 2 slots shared by crossing rings. Thus each ring communicates with 2 adjacent rings, as illustrated in Figure 7. For all examples, we choose Williams' PS0 pipeline scheme. 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.

Table 1 shows the experimental results of our algorithm with and without the lower bound algorithm (presented in Section 6) enabled. When the lower bound algorithm is enabled, the run time is cut by a factor of up to two orders of magnitudes. The results demonstrate that using our 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 pre-assigning slots with abstract latches. For instance, we ran additional experiments where for each structure, we pre-assigned several selected slots with abstract latches. As shown in Table 1, the run-times are significantly reduced.

8 Conclusions

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 moderately sized systems. Moreover, complexity reduction methods for its application to larger systems are also presented and evaluated.

There are many interesting future directions for this research. For example, although the algorithm as described is restricted to models

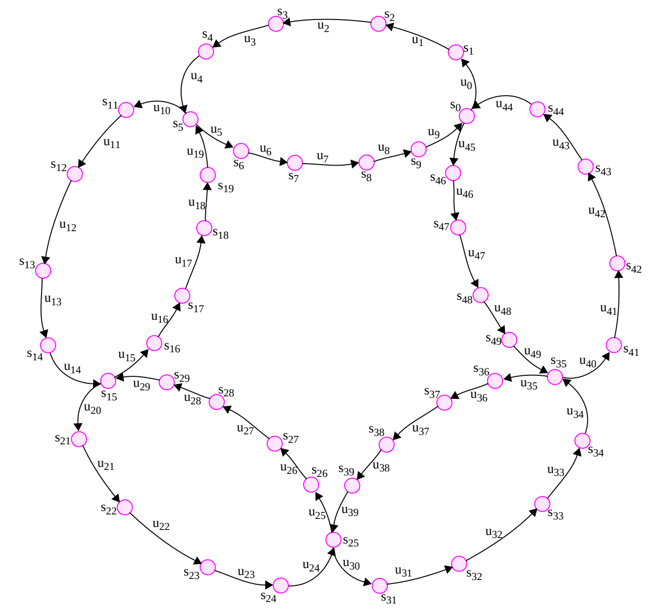


Figure 7: An asynchronous ring-of-rings model.

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 highest probability to those with lowest probability. Specifically, the abstract latches assigned in one iteration would be assumed pre-assigned 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.

Acknowledgment

The authors would like to thank Alex Kondratyev of Theseus Logic, for his invaluable comments on an earlier version of this work.

References

[1] V. Akella and G. Gopalakrishnan. SHILPA: A high-level synthesis system for self-timed circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 587–591. IEEE Computer Society Press, Nov. 1992.

[2] B. Bachman, H. Zheng, and C. J. Myers. Architectural synthesis of timed asynchronous systems. In *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, Oct. 1999.

[3] R. M. Badia and J. Cortadella. High-level synthesis of asynchronous systems: Scheduling and process synchronization. In *Proc. European Conference on Design Automation (EDAC)*, pages 70–74. IEEE Computer Society Press, Feb. 1993.

[4] M. Benes, S. M. Nowick, and A. Wolfe. A fast asynchronous Huffman decoder for compressed-code embedded processors. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 43–56, 1998.

Total # Slots	Total # Units	# Pre-assigned A.L.s	Opt. # A.L.s	Run Time$_1$ (sec) / # visited branches$_1$	Run Time$_2$ (sec) / # visited branches$_2$
Asynchronous Linear Pipeline ($\delta = 150$)					
20	19	2	11	0.37 / 10510	0.03 / 224
25	24	2	13	8.12 / 174868	0.63 / 4283
30	29	2	15	98.73 / 2302461	2.87 / 17979
35	34	2	17	2020.47 / 40418915	46.27 / 246168
35	34	7	19	91.83 / 1573362	1.83 / 9785
Asynchronous Ring ($\delta = 150$)					
20	20	0	9	1.97 / 31648	0.32 / 1518
25	25	0	12	16.45 / 306567	1.33 / 6844
30	30	0	14	339.03 / 5482036	14.17 / 63252
35	35	0	17	5482.67 / 97512110	74.68 / 334881
35	35	5	18	445.05 / 7877279	6.22 / 32055
Asynchronous Ring of Rings ($\delta = 150$)					
46	50	0	23	$\geq$ 3h (time out)	3091438 / 7738.97
46	50	5	23	$\geq$ 3h (time out)	1540014 / 3719.43
Asynchronous Huffman Decoder ($\delta = 50$)					
9	11	4	6	0.01 / 7	0.01 / 3

Table 1: Experimental results for asynchronous pipelines and rings. Quantities with a subscript 1 (2) refer to experiments with the lower bound algorithm disabled (enabled).

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

[6] W.-C. Chou, P. A. Beerel, and K. Y. Yun. Average-case technology mapping of asynchronous burst-mode circuits. *IEEE Transactions on Computer-Aided Design*, 18(10):1418–1434, Oct. 1999.

[7] U. Cummings, A. Lines, and A. Martin. An asynchronous pipelined lattice structure filter. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 126–133, Nov. 1994.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[9] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[10] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, Department of Computer Science, University of Washington, 1995.

[11] C. E. Leiserson and J. B. Saxe. Optimizing synchronous systems. *Journal of VLSI Computer System*, 1:41–67, 1983.

[12] 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 *Advanced Research in VLSI*, pages 164–181, Sept. 1997.

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

[14] N. Shenoy. Retiming: Theory and practice. *Integration, the VLSI journal*, 22:1–21, 1997.

[15] F. Somenzi. *Private Communications*, 1999. F. Somenzi is a professor of computer science at the University of Colorado.

[16] J. Sparsø and J. Staunstrup. Delay-insensitive multi-ring structures. *Integration, the VLSI journal*, 15(3):313–340, Oct. 1993.

[17] T. E. Williams. *Self-Timed Rings and their Application to Division*. PhD thesis, Stanford University, June 1991.

[18] A. Xie, S. Kim, and P. A. Beerel. Bounding average time separations of events in stochastic timed Petri nets with choice. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 94–107, Apr. 1999.