Optimization of Latency Insensitive Systems through Back Pressure Minimization

Bin Xue, Student Member, IEEE, Sandeep K. Shukla, Senior Member, IEEE, and S. S. Ravi, Senior Member, IEEE

Abstract—Latency insensitive protocols (LIP) were originally invented to circumvent the problem of shrinking clock period, and concomitant violation of the synchrony hypothesis in System on Chip (SoC) designs. The multi-cycle delays on long interconnects between synchronously clocked IP blocks are accommodated in latency insensitive system (LIS) designs through extra valid/stall handshakes between components and additional logic blocks called relay stations. The use of handshaking interconnects and relay stations leads to area and latency penalties, that must be minimized for cost effective SoC designs. Interconnected IP blocks with certain graph topology have periodic behaviors that can be exploited to remove the need for handshake interconnects. Unfortunately, the periodic schedule may not exist for any LIS designs consist of two or more strongly connected components. Some of these systems are not bounded without back pressure. In the past, back pressure between SCCs has always been implemented as stall signals in the backward direction, and they are required to prevent overflow. In this paper, we propose an LIS design optimization algorithm which computes a minimum set of back pressure arcs required between SCCs. We model an LIS by a partial back pressure graph (PBPG) and show that the boundedness of a PBPG can be verified by checking the reachability in its strongly connected component graph (SCCG). Based on this, we formulate the problem of finding a minimum set of back pressure arcs (MBPA) and show that this problem can be reduced to the Minimum Cost Arborescence (MCA) problem for directed graphs. This allows us to obtain a polynomial time algorithm for synthesizing a minimum cost LIS implementation starting from a synchronous model of the original system. After adding back pressure arcs, we develop a localized Mixed Integer Linear Programming (LMLP) approach to optimize the throughput of the resulting LIS. This approach scales better than existing MILP-based throughput optimization techniques. In addition, we also provide an implementation of the LIS which refines its PBPG model. To the best of our knowledge, this is the first effort that considers the optimization of back pressure and throughput together in the design of latency insensitive systems.

Index Terms—Latency insensitive system, back pressure, throughput, optimization, buffer sizes

1 INTRODUCTION

1.1 Latency insensitive design

In modern ‘system on chip’ (SOC) designs, interconnect latency is no longer negligible compared to gate delays due to the progress in IC technology scaling and high clock speed. Multi-cycle wiring delay might exist, making intellectual property (IP) composition more challenging than before. Latency Insensitive Protocol (LIP) [1] has been proposed as a synchronous approach to solve the problem by converting a synchronous design into a latency insensitive system (LIS) without compromising the clock speed. Suppose multi-cycle latencies are detected on the connections between (IP1, IP0) and (IP3, IP0) for a design shown in Fig. 1(a), where IP0, IP1, IP2, IP3 are four IP modules. According to theory of LIS [1], a special register named relay station (RS) is inserted on problematic connections (IP1, IP0) and (IP3, IP0) to pipeline the inter-IP communication. Also each IP module is packed within a wrapper as shown in Fig. 1(b), in order to allow the IP module to participate in the LI protocol. In the original proposal, the data flow control between wrappers and relay stations was implemented using handshakes (valid/stall). Adding relay stations introduces bubbles in such communication pipelines. Therefore, it has been proposed to send a valid signal along the data path, indicating whether the current pipeline holds a real data or a bubble brought on by relay stations. At the same time, a stall signal is transferred backward which asserts when a buffer at a given stage is full. Stall signal embodies “back pressure”. Valid and stall signals are also used to clock gate each IP, making it only respond to valid incoming traffic when its output buffer is not full.

Various implementations of this handshake mechanism have been widely proposed in the literature [2]–[5]. However, in all these proposals, handshake has been used along each interconnect without attempts to optimize away some unnecessary ones. Handshaking has its obvious disadvantages such as:

1) Handling back pressure renders the design relatively complicated with additional logic which is error prone and is not area efficient.
2) Unnecessary stalls might reduce the system’s
throughput if buffer size of each pipeline stage is not properly chosen as observed in [6], [7].

3) Techniques such as [8], [9] are proposed to improve the throughput with buffer resizing or buffer addition. However, they are based on solving a mixed integer linear programming (MILP) formulation which is NP-hard. The number of integer variables of this MILP formulation equals the number of back pressures. Solving MILP will not scale if the system is large.

Above drawbacks opens an opportunity to optimize the traditional LIS implementation so that the back pressures are used sparingly with sufficiency conditions. Fig. 2 demonstrates a design where back pressure can be avoided. IP1, IP2, packed within Wrapper1 and Wrapper2, are connected through an RS. Only ‘valid’ signals v1, v2, v3 are transferred and stored in flip-flops to clock gate each IP. The clock gating logic is not shown to keep the figure simple. This design does not overflow even without “stall” (back pressure). This is a special case of the observation also reported in [10] and [11]. The observation is that some LIS have implicit periodic behavior. Without back pressure, static schedule still exists for each node of such LIS graph by computing its longest distance from a reference node in the corresponding graph with arc weight re-annotated [10], [12]. According to [12], this reference node should be chosen from the critical cycle (see definition in Sec.2.3), otherwise it’s not a critical operation with respect to a given throughput. Relative schedule to such nodes might fail to meet the timing constraints. However, LIS implementation proposals based on this observation did not achieve the objectives listed below which are necessary for LIS optimization by minimizing back pressure.

1) Implementations must preserve boundedness for any LIS graph topology. For LIS with multiple strongly connected components (SCC), it might be the case that not all nodes are reachable from the reference node which is on the critical cycle as required by [10], [12]. We will elaborate this in Section 2 and Section 3. Overflow could happen on some inter-SCC arcs if back pressures is necessary for that arc. Further analysis in Section 2.4 reveals that back pressure is not necessary for every inter-SCC arc. Our design tools must explore the minimal necessary set of back pressures arcs to bound an LIS. [13] classified the situations where back pressure is required but did not cover the optimization problem. [6] and [7] also analyzed systems which consist of several SCCs but their focus is on throughput enhancement through back pressure at the cost of adding more than necessary number of buffers.

2) Throughput should not be compromised when back pressure is inevitably required. Also, the buffer size used to improve throughput should be minimized since there is a trade off between the two. Our design tools must find the minimal buffer size while the throughput does not decrease after it has been maximized. Throughput improvement through buffer resizing has been discussed in [8], [9] yet applied to LIS where back pressure is partially used.

3) Implementation should be area efficient. When an LIS is bounded, each IP block will be enabled periodically as if it was clock gated by a periodic sequence (see the discussion in Section 6). In [10] and [11], shift registers are proposed to store the periodic sequences (schedulings) and clock gate each IP block. That is why they are sometimes referred as scheduling based protocol. However, it may cause large area overhead if the sequence’s period is long. Fig. 2 shows an implementation of clock gating sequences without using shift registers. Suppose v1 = 1, v2 = 0 and v3 = 1 initially, thus RS and IP2 are enabled while IP1 is not (see the left part of Fig. 2). In the next cycle, the valid bits are right shifted such that v1 = 1, v2 = 1 and v3=0, which enables RS and IP1 but disables IP2. (see the right part of Fig. 2). Over time, RS, IP1 and IP2 will have periodic schedulings (110), (011) and (101). This implementation should be extended for more general design.

1.2 Main Contribution

In this paper, we propose a novel front to end design flow for LIS optimization (see Fig. 3) based on minimiz-
Fig. 3. Proposed design flow for LIS optimization

1) Model LIS using partial back pressure graph (PBPG). We propose PBPG to model LIS’s entire evolution from no initial back pressure to the minimal necessary back pressure added for boundedness. Throughput and buffer sizes can be computed based on PBPG.

2) Check boundedness of LIS based on strongly connected component graph (SCCG). We propose a graph model SCCG and based on which a theorem is derived to determine whether such an LIS is bounded.

3) Make LIS bound by adding minimal back pressure arc (MBPA). If the LIS is not bounded, we provide an algorithm to search for the minimal back pressures (MBPA) required to prevent overflow. We also show that MBPA can be solved as the well known Minimum Cost Arborescences problem, which has polynomial solution.

4) Optimizing throughput through buffer resizing by solving localized mixed integer linear programming (LMLIP) formulation. If throughput is reduced after MBPA, LMLIP is proposed to improve the throughput with minimal buffer addition. LMLIP scales better than existing methods for the resulting system of MBPA.

5) Implementation of LIS by mapping PBPG to circuits. With both buffer and handshake information, an LIS implementation precepts are proposed by mapping PBPG to circuits.

Related Work

Optimization of an LIS’s performance has been studied in a variety of past papers. For throughput optimization, apart from those mentioned earlier in [8], [9] based on buffer resizing, recycling is also proposed by [14], [15]. This method adds bubbles into the fast branch to balance the throughput by buffer insertion. [16], [17] optimizes LIS implementation by exploiting dependency information inside an IP block through a functional independence condition (FIC) module. Early evaluation is proposed by [18] which prevents stalling of a block caused by non-required inputs to improve throughput. Antitoken is used to annihilate any non-required inputs if they arrived late after a computation is complete. Instead of antitoken, [19] uses an adaptive wrapper to monitor the misalignment between processed and non-processed inputs. This paper tries to extend implicit schedule to more general LIS optimization by adding minimal back pressures.

To make use of the implicit schedule, we assume the LIS have known and constant sending/consuming rates of its inputs/outputs. We also assume each IP block should have known pipe stage in this paper. This scenario exists in many DSP systems. IP with cache, branch prediction or other unknown delays are beyond the scope of this paper. The paper is organized as follows. Section 2 discusses PBPG and its modeling of LIS. Section 3 proposes SCCG, and presents a necessary and sufficient condition to check boundedness of a PBPG by SCCG. In Section 4, MBPA is proposed and shown solvable as the well known minimum cost arborescence MCA problem. Section 5 talks about LMLIP formulation for throughput optimization via buffer resizing. Section 6 demonstrates an implementation protocol which refines the PBPG model. Section 7 demonstrates experimental results which validate our methods. Conclusion is drawn with future work in Section 8.

2 MODEL LIS USING PARTIALLY BACK PRESSURE GRAPH

2.1 Model of LIS

LISs have been successfully modeled by marked graphs [18]. Notations used in this paper are from [20].

Definition 1 (Marked Graph): A marked graph is a Petri net \((P,T,W,M)\), where \(P, T\) are the sets of places and transitions, \(W\) is the set of arcs such that \(W \subseteq (P \times T) \cup (T \times P)\), \(M\) represents the markings \((M : P \rightarrow \mathbb{N})\), and every place \(P\) has exactly one incoming arc and one outgoing arc. \(M_0\) represents the initial markings.

Notation \(*\) \((\ast)\) represents the set of input (output) places of \(t\), i.e. \(\ast = \{ p \mid (p,t) \in W \}\) \((\ast = \{ p \mid (t,p) \in W \}\}\); Similarly, \(\ast p = \{ t \mid (t,p) \in W \}\) \((\ast = \{ t \mid (p,t) \in W \}\}\) is the set of input (output) transitions of \(p\). A transition \(t \in T\) is enabled, if \(M(p) > 0\) for every \(p \in \ast t\).

Marked graph is conflict free [20] that once a transition is enabled it will not be disabled because of firing other transitions. Marked graph is originally used to model asynchronous system, if we require each transition fire as soon as it enables, then marked graph can be used to model synchronous system. This way of firing transitions is called (as soon as possible) (ASAP) scheduling. However, the firing of \(t\) does not depend on \(M(p)\) where
p ∈ M(t*) which implicitly assumes that p can take infinite number of tokens. For marked graph, |p*| = |p| = 1. We can use place p to represent arcs (t, p) and (p, t'), that is, the link connecting t and t', where t ∈ *p and t' ∈ p*. In this situation, we use the term "arc p" for the link between t and t' for the sake of simplicity. Accordingly, we use (P, T, M) rather than (P, T, W, M) to address an marked graph. Associating places and transitions with time delays leads to Time Marked Graph [20].

Elastic Marked Graph (EMG) is presented in [21] to model LIS with back pressure for every interconnects. In an EMG, any arc p ∈ P has a complementary arc p′ ∈ P such that *p = p* and *p′ = p*. The complementary arc p′ makes p bounded [20] with ASAP scheduling and represents back pressure. A labeling function L maps all arcs of an EMG as forward or backward (L : P → {F, B}) such that L(p) = F if L(p′) = B. Forward arc p is bounded by 2 such that M(p) + M(p′) = 2. This is consistent with [2], [4], [5] where the capacity of each buffer is 2.

Since our approach for LIS optimization relies on reducing unnecessary back pressure arcs, we need to model an LIS where only parts of the interconnects have back pressure. That is why partial back pressure graph (PBPG) is proposed.

Definition 2 (PBPG): A partial back pressure graph (PBPG) is a 4-tuple (T, A, M, B), such that:

1. (T, A, M) is a marked graph;
2. A consists of three subsets A_U, A_F and A_B and is associated with a labeling function L : A → {F, B}:
   - A_U: set of forward arcs without back pressure arcs, ∀a ∈ A_U, L(a) = F;
   - A_F: set of forward arcs with back pressure arcs, ∀a ∈ A_F, L(a) = F;
   - A_B: set of back pressure arcs, ∀a ∈ A_B, L(a) = B;
3. A bijection exists between A_F and A_B that for each arc a(t_i, t_j) ∈ A_F, ∃a′(t_i, t_j) ∈ A_B, a ∈ A has unit time delay.
4. B : A_U ∪ A_F → N is the upper bound on the capacity of forward arcs. ∀a ∈ A_U, B(a) = ∞; ∀a ∈ A_F, B(a) ≥ 2;

2.2 Model LIS with or without back pressure using PBPG

In our discussion, we assume that each RS or IP block has single pipeline stage such that it can be regarded as a composition of combinational logics and one stage flip-flops. IPs with multiple pipeline stages can be divided into several sub-blocks in serial. We model an LIS with partial back pressure according to the following rules:

1) The combinational logics of a computation block B (either IP modules or RSs) is modeled by a transition t. The storage of B is modeled by arc a ∈ T ∩ (A_U ∪ A_F).
2) For arc a ∈ A_U ∪ A_F, M_0(a) = 1 if a models the storage of an IP module. Otherwise, M_0(a) = 0.
3) Arc a ∈ A has unit latency which models one clock cycle delay introduced by a single pipeline stage.
4) A PBPG model is fired with ASAP scheduling. "One clock cycle transaction of a PBPG" means one time ASAP scheduling of the PBPG.
5) M(a) (a ∈ A_U ∪ A_F) equals the number of valid data stored in the buffer modeled by a. Arcs in A_B are drawn by dash lines. a′ ∈ A_B has the initial token M_0(a′) = B(a) − M_0(a), where a is its corresponding forward arc p ∈ A_F. M(a′) represents the number of available buffers for a. M(a′) = 0 means a is full and the IP block modeled by *a should be stalled.

An example of PBPG is shown in Fig. 4(a) and 4(b) which have four blocks Bi(i = 0, 1, 2, 3). Their combinational logics are modeled as transitions T_{B_i} in Fig. 4(b). Bi’s output buffer is Qi. vqi indicates whether Qi has valid data. The output buffers are modeled as arcs ai. M(ai) are the number of valid data in Qi. sq3 is the “stall” from B3 to B2 and which is modeled by the dashed arc a3’. a3’ is the back pressure arc of a3. M(a3′) = B(a3) − M(a3) represents the number of available buffers on edge a3. M(a3′) = 0 means a3 is full. a1 and a2 have no back pressure, so a1, a2 ∈ A_U. a3 has back pressure a3’, so a3 ∈ A_F and a3’ ∈ A_B.

B2 is enabled when Q2 has valid data and Q3 is not full. While B1 is enabled only when Q1 has valid data which implicitly assumes that Q2 is big enough to hold data produced by B1. Even though B1 and B2 are activated differently, their activation can be uniformly modeled in its PBPG that a transition is enabled when all of its incoming arcs are marked. For example, T_{B2} can fire when both M(a2) > 0 and M(a3′) > 0 while T_{B1} can fire when M(a1) > 0. Note that overflow happens on Q2 in this example.

In the discussion of this paper, we start from an LIS with AF = AB = φ (no back pressure). Gradually adding back pressure arc to a ∈ AU moves a from A_U to A_F. Eventually, only finite number of tokens will reside on arc a ∈ A_U under ASAP scheduling.
2.3 Performance analysis based on PBPG

A directed cycle \( C \) in a PBPG is formed by arcs in \( A \). The cycle-mean of \( C \) is \( \delta(C) = \frac{\sum_{a \in C} M_0(a)}{|C|} \), where \(|C|\) is the number of arcs on \( C \) and is also the latency of \( C \). The throughput of an SCC \( scc_k \) is defined by \( \delta(scc_k) = \min_{C \in scc_k} \delta(C) \). For a system \( G \) consisting of multiple SCCs, the maximum achievable throughput is \( \delta_U = \min_{scc \in G} \delta(scc) \). The cycle whose cycle-mean equals \( \delta_U \) is denoted as critical cycle \( C_0 \).

2.4 Interconnects between SCCs

Now, we describe the boundedness issue by considering the example in Fig. 5(a) and Fig. 5(b). \( scc_1 \) has four transitions \( \{T0, T1, T2, T3\} \) and \( \delta(scc_1) = 3/4 \). \( scc_2 \) has three transitions \( \{T4, T5, T6\} \) and \( \delta(scc_2) = 2/3 \). \( scc_1 \) and \( scc_2 \) are connected through \( a0 \). If the data flow is from \( scc_1 \) to \( scc_2 \) (Fig. 5(a)), \( T1 \) produces three tokens into \( a0 \) for every four clock cycles and \( T6 \) consumes two tokens from \( a0 \) for every three clock cycles. This leads to overflow on \( a0 \) over time. On the other hand, if the data flow is from \( scc_2 \) to \( scc_1 \) (Fig. 5(b)), \( T1 \) can’t fire until \( M(a0) > 0 \), so \( a0 \) is bounded. Even though three tokens will accumulate at \( a1 \) because of buffer synchronization, we can increase the capacity of \( a1 \) with bounded value for the accumulated tokens and no overflow happens in this situation. This example also shows a situation where the method in \([10], [12]\) can’t find schedule. The critical cycle here is \( scc_2 \). In Fig. 5(a), since \( \{T0, T1, T2, T3\} \) are not reachable from \( scc_2 \), their absolute distances from the reference nodes on \( scc_2 \) are infinite, no finite schedule for these transitions can be computed by \([10], [12]\). While in Fig. 5(b), every transition is reachable from \( T4 \), static schedule can be computed accordingly. This observation leads to Theorem 1. Limited by space, one can refer \([22]\) for the detailed proof of Theorem 1.

![Fig. 5. Communication between two SCCs with different throughputs: scc1(3/4), scc2(2/3)](image)

**Theorem 1:** Let \( scc_1 \) and \( scc_2 \) be two SCCs in a PBPG with \( \delta(scc_1) > \delta(scc_2) \). Let there be unidirectional links between \( scc_1 \) and \( scc_2 \) in any direction. The subsystem consisting of \( scc_1 \) and \( scc_2 \) is bounded with ASAP scheduling if and only if the unidirectional links are from \( scc_2 \) to \( scc_1 \).

In case of overflow, back pressure arcs are required. For example, \( a0' \) is added for \( a0 \) which stalls \( T1 \) if \( a0 \) is full. Note that this also makes every transition reachable from \( T4 \), so \( T4 \) could be a reference transition required by \([10], [12]\). Adding \( a0' \) merges \( scc_1 \) and \( scc_2 \) into a new SCC \( scc_{12} \) whose throughput is \( 2/3 \). It equals the throughput of \( scc_2 \), the SCC with smaller throughput. However, in general we only have \( \delta(scc_{12}) \leq \min\{\delta(scc_1), \delta(scc_2)\} \). We prefer the equality to preserve the performance. Whether the equality hold depends on the buffer size of the inter-SCC arcs. In the following discussion of this paper we assume:

**Assumption 1:** when several SCCs: \( scc_1, scc_2, \ldots, scc_k \) are merged into one \( scc_m \) due to added back pressure arcs, we always have enough buffer sizes for the inter-SCC arcs to make \( \delta(scc_m) = \min\{\delta(scc_i), i = 1, 2, \ldots, k\} \).

In Section 5, we will discuss about how to optimize the required buffer sizes for this assumption.

3 Boundedness analysis with strongly connected component graph

Example in Section 2 indicates that the boundedness of an LIS depends on the throughput relation between any pair of connected SCCs. It’s natural to find all SCCs from a PBPG and compute their individual throughputs, which constructs a strongly connected component graph (SCCG). We only focus on connected systems; thus, we assume that the graphs under consideration have a single connected component when the directions on the edges are removed.

**Definition 3 (Strongly Connected Component Graph):**

SCCG(\( V_{scc}, A_{scc}, \delta, \delta', MinV \)) is a directed acyclic multi-graph constructed from a given PBPG model \( G \):

- Each \( v \in V_{scc} \) is an SCC of \( G \).
- Each arc \((v1, v2) \in A_{scc}\) connects two SCC \( v1 \) and \( v2 \).
- \( \delta : V_{scc} \rightarrow Q_{+} \), \( \delta(v) \) is the throughput of \( v \) when \( v \) is disconnected from other SCCs.
- \( \delta' : V_{scc} \rightarrow Q_{+} \), \( \delta'(v) \) is the throughput of \( v \) when connected to the entire system. (We require that \( \delta'(v) = \delta(v) \) initially, then \( \delta'(v) \) changes when connecting to other SCCs due to dependencies.)
- For trivial SCC \( v \in V_{scc} \), which has a single transition, we assume \( \delta(v) = 1 \).
- \( MinV \subseteq V_{scc} \) is the set of SCCs which have the minimum \( \delta \). The maximum throughput that \( G \) can achieve is \( \delta_U = \delta(MinV) \). \( \delta(MinV) \) represents \( \delta(v) \) where \( v \in MinV \).

Two SCCs may be connected by multiple arcs in an SCCG but these arcs should have the same direction; otherwise, the two SCCs is actually one SCC. The difference between \( \delta(v) \) and \( \delta'(v) \) is the throughput when \( v \) is disconnected from other components; therefore, \( \delta(v) \) is determined only by \( v \)'s topology. On the other hand, \( \delta'(v) \) reflects the throughput change when \( v \) is connected to other SCCs due to data dependencies. The SCCG of the PBPG in Fig. 6(a) is shown in Fig. 6(b). By Definition 3, \( V_{scc} = \{v1, v2, v3, v4\}, v1, v2, v4 \) are three trivial SCCs with singular transitions \( T0, T1, \) and \( T5 \) respectively. \( v3 \) consists of \( \{T2, T3, T4\} \) and their interconnects. \( A_{scc} = \{a0, a1, a3, a5\} \). \( a1 \) and \( a3 \)
have the same direction from \(v_2\) to \(v_3\). \(\delta(v_1) = \delta(v_2) = \delta(v_4) = 1\) and \(\delta(v_3) = 1/3\). Thus, \(\text{Min} V = \{v_3\}\) and \(\delta(\text{Min} V) = 1/3\). It is easy to show that the SCCG is weakly connected (i.e., when all of its directed arcs are replaced by undirected arcs, the resulting undirected graph is connected).

At the end of Step (c), every node is reachable from \(\text{Min} V = \{v_3\}\). In the following discussion, we will show that it is a necessary and sufficient condition for a bounded SCCG under Assumption 1. Reachability is formally defined as:

Definition 4 (Reachability): Let \(v\) be a node in \(V\), and let \(V_1\) be a subset of \(V\). Node \(v\) is reachable from \(V_1\) iff either \(v \in V_1\), or \(\exists a_0, a_1, \ldots, a_k\) such that \(\forall v \in V_1\) and \(a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_k \rightarrow v\) (i.e., there is a directed path from \(u\) to \(v\)). Here, \(a_i (i = 0, 1, \ldots, k)\) is either a forward or an added back pressure arc.

With stabilization, a different \(\text{SCCG}_{f}(V_{\text{scce}}, A_{\text{scce}}, \delta_f, \delta_f', \text{Min} V_f)\) is formed finally compared to the original \(\text{SCCG}(V_{\text{scce}}, A_{\text{scce}}, \delta, \delta', \text{Min} V)\) by merging SCCs and recomputing their throughputs (Fig 7(d)). All inter-SCC arcs in \(\text{SCCG}_{f}\) are forward arcs.

Lemma 1: In \(\text{SCCG}_{f}(V_{\text{scce}}, A_{\text{scce}}, \delta_f, \delta_f', \text{Min} V_f)\), \(\forall a \in A_{\text{scce}}\), \(a\) is a forward arc, i.e., \(L(a) = F\).

Proof: Suppose \(\exists v, v_1 \in A_{\text{scce}}\) and \(L(a) = B\). \(v_1, v_2\) are two different SCCs in \(V_{\text{scce}}\). Since back pressure arc must have a complementary forward arc, \(\exists a' \in A_{\text{scce}}\) such that \(L(a') = F\). Then, \(v_1\) and \(v_2\) should be one SCC.

When \(\text{SCCG}_{f}\) is bounded after stabilization, nodes in \(V_{\text{scce}}\) are also reachable from \(\text{Min} V_f\) as shown in Fig. 7(d). It can be proven as a sufficient and necessary condition for boundedness.

Theorem 3: \(\text{SCCG}_{f}(V_{\text{scce}}, A_{\text{scce}}, \delta_f, \delta_f', \text{Min} V_f)\) is bounded iff \(\forall v \in V_{\text{scce}}, v\) is reachable from \(\text{Min} V_f\).

Proof: "\(\Rightarrow\)" Suppose for \(\forall v \in V_{\text{scce}}, v\) is reachable from \(\text{Min} V_f\). Due to dependency, \(\delta'(v) \leq \delta(\text{Min} V_f)\) while \(\delta'(v) \geq \delta(\text{Min} V_f)\) by the definition of \(\delta(\text{Min} V_f)\). So \(\delta'(v) = \delta(\text{Min} V_f)\). For \(\forall a(v_i, v_j) \in A_{\text{scce}}\), since \(v_i\) and \(v_j\) are reachable from \(\text{Min} V_f\), so \(\delta'(v_i) = \delta'(v_j) = \delta(\text{Min} V_f)\). Therefore, no overflow happens on arc \(a\) and \(\text{SCCG}_{f}\) is bounded.

"\(\Leftarrow\)" For any \(V \subseteq V_{\text{scce}}\), let \(\text{Rec}(V)\) be the set of nodes in \(V_{\text{scce}}\) which are reachable from \(V\). Let \(V_s\) be the subset of \(V_{\text{scce}}\) where nodes have no predecessors, then \(V_{\text{scce}} = \text{Rec}(V_s)\). Let \(V_s = V_s \setminus \text{Min} V_f\) and \(V_s = V_s \setminus \text{Min} V_f\).

1) If \(V_s = \emptyset\), then \(V_{\text{scce}} = \text{Rec}(V_s) \subseteq \text{Rec}(\text{Min} V_f)\), given \(V_s \subseteq \text{Min} V_f\). The proof completes;
2) If \(V_s = \emptyset\), then all nodes in \(\text{Min} V_f\) have predecessors. We can always find a node \(u \in \text{Min} V_f\) such that \(u\) has a predecessor \(w\) via arc \(a(u, w)\) and \(w \notin \text{Rec}(\text{Min} V_f)\), since \(\text{SCCG}_{f}\) is DAG. Given \(\text{SCCG}_{f}\) is bounded, no overflow happens on arc \((u, w)\), so \(\delta'(ux) = \delta'(u) = \delta(\text{Min} V_f)\). We trace \(ux\)'s predecessor until it reaches a node \(v_x \in V_s\) and \(\delta'(vx) = \delta(\text{Min} V_f)\) (nodes along this path must have the same \(\delta\)). However, \(vx\) has no predecessor and \(vx \notin \text{Min} V_f\), so \(\delta'(vx) = \delta(vx) > \delta(\text{Min} V_f)\), which contradicts \(\delta'(vx) = \delta(\text{Min} V_f)\).

3) Otherwise, \(\text{Rec}(V_s) \cap \text{Rec}(V_s) = \emptyset\) since \(\text{SCCG}_{f}\) is connected. We can always find a node...
Since we trace $\delta'(u) = \delta(\text{MinV}_f)$, given $u \in \text{Rec}(V_{s2})$. Since $u$ is bounded, $\delta'(ux) = \delta'(u) = \delta(\text{MinV}_f)$. We trace $ux$'s predecessor to a node $vx \in V_{sl}$, so $\delta'(vx) = \delta'(u) = \delta(\text{MinV}_f)$. However, $vx$ has no predecessor and $vx \notin \text{MinV}_f$, so $\delta'(vx) = \delta(\text{MinV}_f)$, which contradicts $\delta'(vx) = \delta(\text{MinV}_f)$.

Theorem 4 allows us to check boundedness of SCCG by analyzing reachability with respect to its original pressure arcs. Let $v$ be an arbitrary node in $\text{MinV}_f$, $u \notin \text{MinV}_f$, and $\text{SCCG}_f = \text{SCCG}(V_{sccf}, \delta_{sccf}, \delta')$. Theorem 5 allows one to analyze a PBPG's boundedness by checking reachability with respect to its original $\text{MinV}$ which is unchanged during stabilization. Note that Theorem 5 can also check whether method [10], [12] can find schedule in a general graph by checking if every node is reachable from the critical cycle. Limited by space, we only briefly the explanation. Since $\text{MinV}$ is the set of SCCs with the minimum cycle mean. Critical cycle must be on one of the SCCs in $\text{MinV}$, so is the reference node. If every SCC is reachable from $\text{MinV}$ in SCCG, every node in the original PBPG is reachable from one node inside $\text{MinV}$ which could be used as the reference node to compute schedule. On the contrary, if an SCC is not reachable from $\text{MinV}$, nodes on this SCC are not reachable from the critical cycle. Their relative schedule to the reference node can't be found by [10], [12].

4 ALGORITHM FOR MINIMIZING BACK PRESSURE ARCS (MBPA)

Most SCCGs do not satisfy the requirements of Theorem 5. Not all nodes in $V_{scc}$ are reachable from $\text{MinV}$. For example in Fig. 9(a), only $v_8$ is reachable from $\text{MinV} = \{v_4, v_6\}$ and this SCCG is not bounded. Back pressure arcs should be added to make each node $v \in V_{scc}$ reachable from $\text{MinV}$. It is important
Minimization of Back Pressure Arcs (MBPA) problem shown below:

Minimize the number of back pressure arcs to avoid the drawbacks addressed in Section 1. In addition, back pressure arcs can only be added along the existing forward arcs. Not all added arcs are qualified as back pressure arcs such as $v8 \rightarrow v0$ and $v8 \rightarrow v9$ in Fig. 9(a). The minimization of back pressure arcs (MBPA) problem can be formulated as the graph problem shown below:

**Minimization of Back Pressure Arcs (MBPA)**

**GIVEN:** A weakly connected acyclic graph $V_{scc}, A_{scc}$, $\delta, \delta'$, $MinV$.

**REQUIREMENT:** Extend $A_{scc}$ to $A'_{scc}$ (with restriction that $\forall u' \in A'_{scc}, A_{scc}, v \xrightarrow{a} u$, it must be that $3a \in A_{scc}$, $u \rightarrow v$) such that the following conditions hold.

1) $F \subseteq A$.
2) $T$ is a spanning tree for $G$ if the directions on the edges in $F$ are omitted.
3) For every node $v \in V$, there is a directed path in $T$ from $r$ to $v$.

If $T$ is an arborescence of a graph $G$ with respect to a node $r$, one can draw $T$ with $r$ as the root. For this reason, we will use the phrase “arborescence rooted at $r$" in preference to “arborescence with respect to $r$”. Given a directed graph $G(V, A)$ and a node $r$, the following lemma from [24] gives a necessary and sufficient condition for $G$ to have an arborescence rooted at $r$.

**Lemma 3:** Given a directed graph $G(V, A)$ and a node $r$, $G$ contains an arborescence rooted at $r$ if and only if for every node $v \in V$, there is a directed path from $r$ to $v$ in $G$.

Suppose $G(V, A)$ is a directed graph with a nonnegative cost $w_{e}$ for each directed edge $e \in A$. If $T(V, F)$ is an arborescence of $G$ rooted at a node $r$, then the cost of $T$ is the sum of the costs of the directed edges in $F$. We can now define the minimum cost arborescence (MCA) problem.

**Minimum Cost Arborescence (MCA)**

**GIVEN:** A directed graph $G(V, A)$ with nonnegative edge costs and a node $r \in V$.

**REQUIREMENT:** Find a minimum cost arborescence of $G$ rooted at $r$, if such an arborescence exists.

The MBPA problem can now be solved using the algorithm shown in Fig. 10. We can now prove the correctness of the algorithm.

**Theorem 7:** The solution provided by the algorithm shown in Fig. 10 is an optimal solution for MBPA problem.

**Proof:** To prove the theorem, we need to show the following:
Algorithm

Input: A weakly connected directed graph SCCG(V_{scc}, A_{scc}, \delta, \delta', MinV).

Output: An optimal extension SCCG_{opt}(V_{scc}, A_o, \delta, \delta', MinV) of SCCG.

Steps:
1) Construct an edge weighted directed graph SCCG1(V1, A1, \delta, \delta', MinV) from SCCG(V_{scc}, A_{scc}, \delta, \delta', MinV) as follows.
   (a) For each edge (u, v) \in A_{scc}, add both the directed edges (u, v) and (v, u) to A1.
   (b) The costs of edges in A1 are determined as follows: if (u, v) \in A_{scc}, w(u, v) = 1, otherwise w(u, v) = 0.
   (c) Add a node vx to V_{scc} to get V1 so that V1 = V_{scc} \cup \{vx\} and edges (ex, u) to A1, if u \in MinV. Let w(vx, u) = 0.
2) Obtain a minimum cost arborescence T(V_{scc}, F) of SCCG1(V1, A1, \delta, \delta') with root vx.
3) Remove edges (u, v) with w(u, v) = 1 from A1, if (u, v) \notin F.
4) Remove node vx from V1 to get back to V_{scc}.

Fig. 10. Steps of the Algorithm for MBPA

1) There is an arborescence T(V_{scc}, F) rooted at vx for the graph SCCG1 that results at the end of Step 1) of the algorithm.
2) Graph SCCG_{opt}(V_{scc}, A_o, \delta, \delta', MinV) satisfies the requirements of the MBPA problem.
3) The number of edges in |A_o| - |A_{scc}| is a minimum.

We prove these parts separately.

Part 1: From Lemma 3, it is only required to show that in SCCG1, there is a directed path from vx to each node in V1 - \{vx\} = V_{scc}. Since SCCG is weakly connected, there is an undirected path from each v0 \in MinV to each node in V_{scc} in the graph obtained by erasing the edge directions in SCCG. For each edge \{u, v\} of this undirected path, the graph SCCG1 contains both the directed edges (u, v) and (v, u) by Step 1a). Therefore, in SCCG1, there is a directed path from v0 to each node in V_{scc}. On the other hand, by Step 1c), a directed path from vx to v0 is created. Thus, there is a directed path from vx to each node in V_{scc}.

Part 2: We argue that the edge set A_o of SCCG_{opt} satisfies the requirements of MBPA problem (see Definition of MBPA). By Step 3), A_o contains all the edges in A_{scc} plus all the edges of cost 1 from the arborescence T. Further, for each edge (v, u) of cost 1 in T, the reverse edge (u, v) is in A_{scc}. Thus, A_{scc} \subseteq A_o and for each edge (u, v) \in A_o, either (u, v) or (v, u) is in A_{scc}. Further, in T, there is a directed path from vx to each node in V_{scc} = V1 - \{vx\}. By Lemma 2, V_{scc} is reachable from MinV. Therefore, SCCG_{opt} satisfies the requirements of MBPA problem.

Part 3: Here, we need to show that the number of new edges added to A_{scc} that is, |A_o| - |A_{scc}| is a minimum. By Lemma 2, we only need to consider the case when MinV has just one node. Thus, we can assume MinV = \{vx\} in the following argument. Let OPT\_aug denote the minimum number of edges to be added to A\_scc to obtain an optimal solution of MBPA. Let OPT\_arb denote the cost of the optimal arborescence T produced in Step 2) of the algorithm. Note that the only edges added to A_{scc} to produce A_o are the edges of cost 1 in T by Step 3). Thus, OPT\_arb = |A_o| - |A_{scc}|. Thus, we can complete the proof of Part 3 by showing that OPT\_aug \geq OPT\_arb.

We prove this by contradiction. Suppose OPT\_aug < OPT\_arb. Let SCCG^*(V_{scc}, A^*, \delta, \delta', MinV) be an optimal solution of SCCG. Since SCCG^* is optimal, OPT\_arb = |A^*| - |A_{scc}|. Also, by the requirement of MBPA, in SCCG^*, there is a directed path from vx to each node in V_{scc}. Therefore, by Lemma 3, SCCG^* contains an arborescence T^* rooted at vx. Note that T^* contains at most OPT\_aug directed edges which are not in A_{scc}. Further, each edge in A^* is also in A1. Therefore, T^* is also an arborescence rooted at vx for SCCG1. Further, because of our choice of edge costs in SCCG1, the cost of T^* is at most OPT\_aug. Since we assumed that OPT\_aug < OPT\_arb, we reach the conclusion that there is an arborescence T^* for SCCG1 whose cost is less than OPT\_arb. This contradicts the optimality of the arborescence T constructed in Step 2). This completes the proof of Part 3 and also that of the theorem.

Fig. 9(d) shows the back pressure arcs added by our algorithm with dashed arrows. The algorithm adds 5 back pressure edges compared to 14 if one inserts a back pressure edge for every a \in A_{scc}. Note that MBPA also guarantees every node is reachable from the reference node required by [10], [12]. In this case, reference node can be chosen from any nodes in v4 or v6.

5 Throughput Optimization for the Resulting Graph of MBPA

5.1 Problem of throughput reduction

Recall that our boundedness analysis relies on Assumption 1, however it was put aside by the process of MBPA, where we simply make B(a) = 2 when a back pressure arc a is added to an inter-SCC arc a. It may form new cycles with a cycle mean lesser than \delta_U, which violates Assumption 1. Failure to meet Assumption 1 may either lead to an unbounded system or throughput reduction. Example in Fig. 2 and Fig. 3 from [25] showed a case where five SCCs v1, v2, v3, v4 and v5 are merged into a single SCC scc_{merge} with added back pressure. Initially, \delta(v1) = 1/3, \delta(v2) = 1, \delta(v3) = 4/11, \delta(v4) = 1 and \delta(v5) = 1/2. Finally, a new critical cycle is formed
which makes $\delta(scc_m) = 2/7$. So $\delta(scc_m) < \min\{\delta(v_i)\} = 1/3$, $i = 1, 2, \ldots, 5$. We call these new SCCs violating Assumption 1 as problematic SCCs. Therefore, we need to increase the problematic SCCs’ throughputs to meet Assumption 1. This can be realized by buffer resizing [8], [9].

5.2 Mixed integer linear programming (MILP) for PBPG’s throughput increase

Works from [8], [9] tried to improve an LIS’s throughput with maximum performance buffer queue sizing. By representing the LIS throughput as a linear relation of the arcs’ buffer sizes, a mixed LP (MILP) formulation is proposed to achieve a target throughput with minimum buffer sizes. These techniques apply the MILP to the LIS where each arc has back pressure. We modified Theorem 1 in [8] for PBPG as Theorem 8.

Theorem 8: For any cycle $C$ in a PBPG $T$, $A$, $M$, $B$), $\delta(C) \geq \delta^*$ iff exists a function $r : T \rightarrow \mathbb{R}$, such that $r(v_{j}) - r(v_{i}) \leq M_0(a) - \delta^*$, where $a(v_{i}, v_{j}) \in A$.

The proof is similar to the proof of Theorem 1 in [8]. Limited by space, one is recommended to refer to [25] for the proofs details. Based on Theorem 8, an MILP formulation for optimal buffer resizing of a PBPG($T$, $A$, $M$, $B$) is given as:

$$\text{Minimize:} \sum_{a \in A_F} B(a),$$

Subject to:

$$r(v_{j}) - r(v_{i}) \leq M_0(a) - \delta^* \quad (1)$$

$$r(v_{i}) - r(v_{j}) \leq B(a) - M_0(a) - \delta^*, \quad (2)$$

$$B(a) \geq 2, \quad (3)$$

where $a(v_{i}, v_{j}) \in A_F$,

$$r(v_{j}) - r(v_{i}) \leq M_0(a) - \delta^*, \quad (4)$$

where $a(v_{i}, v_{j}) \in A_U$.

Recall the definition of PBPG, only arcs in $A_F$ have constraints on buffer capacities thus can be resized. Therefore, only arcs in $A_F$ appear in the objective function. Eq.(2) uses the relation $M_0(a^t) = B(a) - M_0(a)$ in Section 2.1.

5.3 Localized MILP formulation

We apply above formulation for each problematic $scc_m$ by setting $\delta^* = \min\{\delta(scc_i)\} \{i = 1, 2, \ldots, k\}$. where $scc_m$ is formed by merging $scc_1, scc_2, \ldots, scc_k$. This guarantees Assumption 1 so the resulted PBPG is bounded with MBPA. Meanwhile, since $\forall scc_m \in V_{scc_f}, \delta(scc_m) = \min\{\delta(scc_i) \geq \delta_U \ (i = 1, 2, \ldots, k\}$, it also prevents throughput reduction. We name this method as localized MILP (LMILP) formulation since we only apply MILP formulation to the problematic SCCs rather than the entire LIS graph as done in [8], [9]. LMILP is expected to have smaller sizes.

6 IMPLEMENTATION OF THE PROTOCOL

The implementation of LIS from its PBPG ($T$, $A$, $M$, $B$) model includes: (1) implementation of $a \in A$; (2) clock gating (firing) control of an IP block modeled by $t \in T$; (3) initialization. The implementation of any arc $a \in A_U \cup A_F$ is the implementation of output buffers of $a$ which will be our focus. Arcs in $A_D$ model back pressure of forward arcs in $A_F$, whose implementation is part of the implementation of arcs in $A_F$. We consider a state $S$ of a PBPG as a $[T] \cdot$ bit vector $S(s_0, s_1, \ldots, s_{|T| - 1})$. $s_k = 1$ ($s_k = 0$) indicates transition $t_k$ is enabled (disabled) in state $S$. A bounded PBPG will have periodic state sequence as $S = S_n(S_m)\omega$ with the initial sequence $S_0$ and iterative part $S_n$. This is due to the fact that PBPG is deterministic so that the next state of PBPG is determined by the current state. A bounded PBPG has finite reachable states so its states will repeat after an initial sequence. Accordingly, the firing sequence of each transition is also periodic after an initial phase. We can simulate the resulted PBPG with its initial markings to get $S = S_n(S_m)\omega$ by restoring every reached state until repeating phase $S_i$ is detected. Firing sequences are obtained at the same time. We determine $B(a)$ with respect to three cases:

1) $a \in A_U$ and $a$ is not an incoming arc of a join transition, a transition with multiple incoming arcs.

No more than one token can reside on $a$ at any time according to ASAP scheduling, so $B(a) = 1$;

2) $a \in A_U$ and $a$ is an incoming arc for a join transition. Multiple tokens may accumulate on $a$ due to barrier synchronization. However, its maximum token number is bounded if the PBPG is bounded. $B(a)$ can be derived from the firing sequence of $v_i$, $v_j$ ($a(v_i, v_j)$) and its initial marking $M_0(a)$ [26];

3) $a \in A_F$ which has back pressure. $B(a)$ is determined by solving the LMILP formulation discussed in Section 5.

$a$ should have unit latency irrespective of $B(a)$ according to [9] for performance. The implementation of $a$ can be classified into three cases: (1) $a \in A_U$ and $B(a) = 1$; (2) $a \in A_U$ and $B(a) > 1$; (3) $a \in A_F$. Case (1) can be implemented as the same way as in Fig. 2. Case (2) and case (3) can be implemented as skid buffers with the capacity of $B(a)$. We use $\text{FIFO}(n)$ and $\text{FIFO}_B(n)$ to denote the implementation of case (2) and case (3) if $B(a) = n$. The subscript “B” indicates whether the buffer has back pressure. Here we focus on $\text{FIFO}_B(n)$ and $\text{FIFO}(n)$ is a simplified version of $\text{FIFO}_B(n)$ without detecting the “full” condition. We also consider the implementation of case (1) as $\text{FIFO}(1)$.

Fig. 12(a) shows the schematic of $\text{FIFO}_B(4)$. Generally, $\text{FIFO}_B(n)$ has $n$ entries formed by shift registers, where the first entry is the pipeline of the original IP (we assume in Section 2.2 that each IP has one stage pipeline), and $n - 1$-stage are additional entries as in Fig. 12(a). The $n$ stage queue is controlled by “FIFO control”. Signal “head” points to the first valid data,
which should be read by the next block. cnt counts the number of valid data in queue. It receives \( \text{enq} (\text{deq}) \) from neighboring stages, updates head and generates “vout” and “sout”. \( \text{enq} = 1 \) (\( \text{deq} = 1 \)) denotes that a valid data enters (leaves) the FIFO. \( \text{enq} \) also clock gates the shift registers. vout and sout denote whether the queue is empty (\( \text{vout} = 0 \)) or full (\( \text{sout} = 1 \)). The finite state machine (FSM) of FIFO\(_B(n)\)'s controller is shown in Fig. 12(b). Computation of \( \text{enq} \) and \( \text{deq} \) depends on the interconnections with neighboring stages which will be addressed later. FIFO\(_n\) has the same implementation as FIFO\(_B(n)\) except that sout is not produced by FIFO\(_n\)'s controller when its cnt = n. Boundedness of FIFO\(_n\) is guaranteed by the system’s topology as discussed in Section 4 and Section 3.

![Diagram of FIFO controller](image)

Fig. 11. The Schematics of FIFO\(_B(4)\)(a), and state transition of FIFO controller(b)

Secondly, we consider the clock gating control of an IP block modeled as a transition \( T_{IP} \) with multiple incoming and outgoing arcs such as \( T_4 \) in Fig. 13(a). Arcs in \( T_{IP} \) and \( T_{IP}^* \) are implemented as FIFO or FIFO\(_B\) according to their types and buffer sizes. In Fig. 13(a), \( a_1, a_4 \in A_F \), \( a_2, a_3, a_5, a_6 \in A_U \), \( B(a_1), B(a_2), B(a_4), B(a_5) > 1 \) and \( B(a_3) = 1, B(a_6) = 1 \). \( a_1 \) and \( a_4 \) are implemented as FIFO\(_B(3)\) and FIFO\(_B(2)\); \( a_2 \) and \( a_5 \) are implemented as FIFO\(_B(2)\); \( a_3 \) and \( a_6 \) are implemented as FIFO\(_B(1)\) as shown in Fig. 13(b). We only show the implementation details of \( a_1, a_5 \) and \( a_6 \). The solid rectangular shows the boundary of the original IP (1P4) modeled by T4. Firing of \( T_{IP} \) removes a valid data and produces a new data to \( a \in T_{IP}^* \). So enq of \( a_1 \)'s \( \{a_1 \in T_{IP}^*\} \) should be the same as deq of \( a_1 \)'s \( \{a_1 \in T_{IP} \} \), which clock gate the IP block modeled by \( T_{IP} \) as shown in Fig. 13(b).

\[
\text{enq} = \text{deq} = \bigwedge_{a_1 \in T_{IP}} \text{vout}_{a_1} \cdot \bigvee_{a_2 \in T_{IP}^* \cup A_U} \text{sout}_{a_2} \tag{5}
\]

This means that the IP is only enable when all of its inputs are valid and none of its output buffer is full. For example, in Fig. 13(b), deq signal for \( a_1 \) and \( a_2 \), enq signal for \( a_4, a_5, a_6 \) are generated according to:

\[
\text{enq} = \text{deq} = \text{vout}_1 \land \text{vout}_2 \land \text{vout}_3 \land \text{sout}_4 \tag{6}
\]

Finally, we address the initialization. When an LIS restarts, the state (cnt) of each FIFO or FIFO\(_B\) should be reset to \( k \), if the corresponding arc \( a \) in PBPG has \( M_0(a) = k \) \((a \in A_F \cup A_U)\). It represents that this buffer contains \( k \) valid data. For FIFO\(_B(1)\), if the corresponding arc \( a \) has \( M_0(a) = 1 \), set its \( \text{vout} = 1 \) initially. After reset, the system will behave the same way as the ASAP scheduling of its PBPG model.

7 EXPERIMENTAL RESULTS

Our approach, which utilizes MBPA to make an LIS bounded with minimum back pressure arcs and constructs LMILP for problematic SCCs to recover throughput, has the following benefits: the number of back pressure arcs is minimized; the throughput upper bound \( \delta_U \) is achieved; the required buffer size is small; the approach has efficient solution. We evaluate all our techniques on tool LOLA-LIS, an extension of LOLA (a low level petri-net analyzer [27]) for LIS design. Experiments are done on a PC with Intel CPU T1300 at 1.66GHz with 1G memory.
7.1 Back pressure minimization using MBPA

The complexity of MBPA includes searching for all SCCs, computing their throughput and solving MCA problem. SCCs of a PBPG \((T, A, B, E)\) can be found in time \(O(|T| + |A|)\) using Tarjan’s algorithm [28]. Suppose each \(scc_k\) has \(n_k\) transitions and \(e_k\) arcs \((\sum n_k = |T| \text{ and } \sum e_k < |A|)\), the throughput for \(scc_k\) can be calculated as [29] with the complexity \(O(n_k(e_k + e_{\text{int}}))\) for the worst case. Therefore the complexity of constructing the SCCG has a worst case complexity of \(O(\sum n_k(e_k + e_{\text{int}}) + |A|)\). MCA problem has polynomial time algorithms [30]–[32]. We choose an open source package from [33], which has a running time \(O(|V|)\) for normal graphs. Therefore, the MBPA has a worst case complexity of \(O(\sum n_k(e_k + e_{\text{int}}) + |A| + |V_{\text{acc}}|^2)\). ISCAS’89 benchmarks \((s27 \sim s1494)\) and ITC benchmarks \((604 \sim b13)\) are used. We model each gate as a transition in PBPG model. Even though LISs are not generally synthesized at such a fine level of granularity, these benchmarks can still provide complex graph topology to evaluate our methods. Interconnect latency between two transitions is randomly generated, and relay stations are inserted based on the latency information. All test cases do not satisfy Theorem 5, so methods in [10], [12] can not be directly used to compute schedule.

In Table 1, column \(|AB|\) shows the number of back pressure arcs computed by MBPA. Column \(|AB|/|A_0|\) is the ratio between \(|AB|\) and total number of inter-SCC arcs. This ratio is less than 27.3% for all benchmarks and is below 11% when the SCCG becomes large. The result demonstrates that only a small percentage of inter-SCC arcs need back pressure.

7.2 Throughput improvement

We refer our approach which first uses MBPA then LMILP as LMILPMBPA, the approach of [8] which applies the MILP formulation for an LIS where all the arcs have back pressures as MILP\(_{HS}\), and the approach of [25] which uses A Reduced MILP (RMILP) formulation for the results of MBPA as RMILPMBPA. A group of ISCAS’89 and ITC benchmark which suffer throughput reduction with added back pressure are used for comparison in Table 2. We may have multiple test cases for the same benchmarks but with different interconnect delays, which are distinguished by the subscripts such as \(s349h_0\) and \(s349h_1\). Column \(\delta_U\) shows the throughput upper bound. Column \(\delta_{HS}\) and \(\delta_{MBPA}\) show the throughput with or without back pressure optimization. For all test cases, we have \(\delta_{HS} < \delta_U\) and \(\delta_{MBPA} < \delta_U\). It implies that MILP is solved for all approaches to recover \(\delta_U\). The size of MILP formulation in each approach, in terms of its number of integer variables, fractional variables and constraints, determines the run time, since solving MILP is NP-hard.

The number of integer variables of MILP formulation equals the number of back pressure arcs of the LIS. Therefore, LMILP\(_{MBPA}\) and RMILP\(_{MBPA}\) with back pressure minimization should have much less integer variables compared to MILP\(_{HS}\) as shown in column \(|I|\) of the corresponding approach. In addition, LMILP\(_{MBPA}\) applied to a subgraph of the LIS has no more integer variables than RMILP\(_{MBPA}\) which is applied to the entire LIS. For most cases, the fractional variables and constraints of RMILP\(_{MBPA}\) and LMILP\(_{MBPA}\) are also less than those of MILP\(_{HS}\) as shown in columns \(|F|\) and \(|cst|\), since both of them have reductions of the MILP’s size. For test cases with smaller PBPG, the difference of formulation size between RMILP\(_{MBPA}\) and LMILP\(_{MBPA}\) is not obvious. However, the number of constraints for RMILP\(_{MBPA}\) will increase dramatically if the corresponding PBPG is a sparse graph. So LMILP\(_{MBPA}\) has smaller MILP formulation when the test case is big. Finally, the execution time of the three approaches are shown in \(M\) (MILPHS), \(t_2\) (RMILPMBPA) and \(t_3\) (LMILPMBPA). For most cases, we have \(t_3 < t_2 < t_1\). When the PBPG becomes larger, MILP\(_{HS}\) did not converge and aborted because of exhausted memory consumption after 30 minutes execution, as indicated by \(>30M\). Above all, LMILP\(_{MBPA}\) has the best performance among the three approaches.

7.3 Buffer sizes computation and comparison

We also compare the buffer size between LMILP\(_{MBPA}\) and MILP\(_{HS}\) in Table 1. For MILP\(_{HS}\), each arc has back pressure and requires at least 2 buffer sizes \([1]\). So at least \(2 + |A|\) buffers are required. However, not all these buffers could be filled during computation. To get a fair comparison, we also compute its state sequence as discussed in Section 6 to get the actual number of buffer which could be ever occupied. The result is shown in column \(B_{HS}\). On the other hand, the buffer size of LMILP\(_{MBPA}\) can be computed by its state sequence and the result of solving LMILP. The length of state sequence, its initial phase and repetitive phase are shown in \(|S|\), \(|S_u|\) and \(|S_u|\) in Table 1. Generally, for LMILP\(_{MBPA}\) most arcs have \(B(a) = 1\). So the total number of buffer is expected to be smaller. The required buffer sizes of LMILP\(_{MBPA}\) is shown in columns \(B_{Su}\) of Table 1. For cases other than \(s344\), \(B_{HS} \geq B_{Su}\).

8 Conclusion

This paper proposed a new design flow for LIS optimization based on back pressure minimization. With the rigorously derived sufficient and necessary condition, MBPA technique finds the minimum set of back pressure arc necessary to make an LIS bounded. Along with LMILP formulation, our methodology efficiently yields an LIS with optimized back pressures and throughput with a small buffer size cost. In this paper, we have no assumption about the wiring delay which can vary among every single bit of any signals. However, if we assume bits of the same signal, such as data path, always have identical wiring delays, further reduction can be
achieved since these bits can share the same flow control channels. This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

**TABLE 1**

<table>
<thead>
<tr>
<th>Bench marks</th>
<th>LIS Int.</th>
<th>SCC Int.</th>
<th>Scc</th>
<th>Reachable States</th>
<th>Buffer Sizes</th>
<th>Performance Exec(s) Mem(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s04</td>
<td>737</td>
<td>1341</td>
<td>588</td>
<td>1090</td>
<td>13</td>
<td>0.6%</td>
</tr>
<tr>
<td>s09</td>
<td>998</td>
<td>1941</td>
<td>943</td>
<td>1773</td>
<td>1</td>
<td>0.02%</td>
</tr>
<tr>
<td>s06</td>
<td>56</td>
<td>98</td>
<td>34</td>
<td>60</td>
<td>2</td>
<td>2.0%</td>
</tr>
<tr>
<td>s07</td>
<td>441</td>
<td>806</td>
<td>117</td>
<td>298</td>
<td>2</td>
<td>0.4%</td>
</tr>
<tr>
<td>s10</td>
<td>193</td>
<td>331</td>
<td>169</td>
<td>244</td>
<td>12</td>
<td>2.2%</td>
</tr>
<tr>
<td>s09</td>
<td>170</td>
<td>306</td>
<td>59</td>
<td>85</td>
<td>2</td>
<td>0.0%</td>
</tr>
<tr>
<td>b10</td>
<td>206</td>
<td>376</td>
<td>67</td>
<td>121</td>
<td>12</td>
<td>5.7%</td>
</tr>
<tr>
<td>s11</td>
<td>770</td>
<td>1415</td>
<td>49</td>
<td>76</td>
<td>7</td>
<td>4.5%</td>
</tr>
<tr>
<td>s12</td>
<td>1076</td>
<td>2094</td>
<td>144</td>
<td>311</td>
<td>7</td>
<td>1.3%</td>
</tr>
<tr>
<td>s13</td>
<td>362</td>
<td>621</td>
<td>101</td>
<td>171</td>
<td>10</td>
<td>3.3%</td>
</tr>
</tbody>
</table>

**REFERENCES**


<table>
<thead>
<tr>
<th>Bench Mark</th>
<th>Throughput Int.</th>
<th>MILP$_{HS}$</th>
<th>RMILP$_{MBPA}$</th>
<th>LMILP$_{MBPA}$</th>
<th>Exec(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>δ$_{HS}$</td>
<td>δ$_{MBPA}$</td>
<td></td>
<td></td>
<td></td>
<td>t1</td>
</tr>
<tr>
<td>610</td>
<td>0.30</td>
<td>0.34</td>
<td>0.38</td>
<td>669</td>
<td>499</td>
</tr>
<tr>
<td>s349</td>
<td>0.74</td>
<td>0.74</td>
<td>0.79</td>
<td>320</td>
<td>217</td>
</tr>
<tr>
<td>s349</td>
<td>0.56</td>
<td>0.56</td>
<td>0.57</td>
<td>443</td>
<td>340</td>
</tr>
<tr>
<td>s382</td>
<td>0.67</td>
<td>0.78</td>
<td>0.8</td>
<td>358</td>
<td>213</td>
</tr>
<tr>
<td>s386</td>
<td>0.42</td>
<td>0.52</td>
<td>0.53</td>
<td>586</td>
<td>405</td>
</tr>
<tr>
<td>s400</td>
<td>0.67</td>
<td>0.71</td>
<td>0.77</td>
<td>375</td>
<td>220</td>
</tr>
<tr>
<td>s444</td>
<td>0.67</td>
<td>0.73</td>
<td>0.8</td>
<td>405</td>
<td>237</td>
</tr>
<tr>
<td>s713</td>
<td>0.63</td>
<td>0.73</td>
<td>0.75</td>
<td>731</td>
<td>568</td>
</tr>
<tr>
<td>s713</td>
<td>0.32</td>
<td>0.35</td>
<td>0.36</td>
<td>1366</td>
<td>1203</td>
</tr>
<tr>
<td>s820</td>
<td>0.33</td>
<td>0.36</td>
<td>0.39</td>
<td>1566</td>
<td>1116</td>
</tr>
<tr>
<td>s820</td>
<td>0.29</td>
<td>0.31</td>
<td>0.32</td>
<td>1689</td>
<td>1239</td>
</tr>
<tr>
<td>s822</td>
<td>0.32</td>
<td>0.33</td>
<td>0.34</td>
<td>1914</td>
<td>1450</td>
</tr>
<tr>
<td>s1488</td>
<td>0.53</td>
<td>0.53</td>
<td>0.54</td>
<td>2198</td>
<td>1463</td>
</tr>
<tr>
<td>s1488</td>
<td>0.39</td>
<td>0.39</td>
<td>0.4</td>
<td>2358</td>
<td>1632</td>
</tr>
<tr>
<td>s1488</td>
<td>0.32</td>
<td>0.32</td>
<td>0.33</td>
<td>3445</td>
<td>2719</td>
</tr>
<tr>
<td>s1488</td>
<td>0.32</td>
<td>0.36</td>
<td>0.37</td>
<td>2844</td>
<td>2118</td>
</tr>
<tr>
<td>s1488</td>
<td>0.27</td>
<td>0.33</td>
<td>0.34</td>
<td>3000</td>
<td>2274</td>
</tr>
<tr>
<td>s1494</td>
<td>0.51</td>
<td>0.51</td>
<td>0.52</td>
<td>2450</td>
<td>1712</td>
</tr>
</tbody>
</table>

TABLE 2
Experiment results on throughput improvement


Bin Xue Bin Xue received his B.S. and M.S. degrees from Beijing Institute of Technology, Beijing, China, in 2005 and 2007 respectively. He received his Ph.D. degree in Computer Engineering from Virginia Tech, Blacksburg, VA, in August 2011. He is a hardware engineer in formal verification at Nvidia Corp, Santa Clara.


S. S. Ravi S. S. Ravi received his B.E. and M.E. degrees from the Indian Institute of Science, Bangalore, India, in 1977 and 1979 respectively. He received his Ph.D. degree in Computer Science from the University of Pittsburgh, Pittsburgh, PA, in August 1984. Since September 1984, he has been with the Computer Science Department at the University at Albany – State University of New York, where he is currently a Distinguished Teaching Professor. His areas of interest include design and analysis of algorithms, social network analysis, data mining, fault-tolerant computing, wireless networks and discrete dynamical systems.