Counterexample-Guided SMT-Driven Optimal Buffer Sizing

Bryan A. Brady[1] Daniel Holcomb[1] Sanjit A. Seshia
UC Berkeley, EECS Department
{bbrady,holcomb,sseshia}@eecs.berkeley.edu

***Abstract*—The quality of network-on-chip (NoC) designs depends crucially on the size of buffers in NoC components. While buffers impose a significant area and power overhead, they are essential for ensuring high throughput and low latency. In this paper, we present a new approach for minimizing the cumulative buffer size in on-chip networks, so as to meet throughput and latency requirements, given high-level specifications on traffic behavior. Our approach uses model checking based on satisfiability modulo theories (SMT) solvers, within an overall counterexample-guided synthesis loop. We demonstrate the effectiveness of our technique on NoC designs involving arbitration, credit logic, and virtual channels.**

I. Introduction

Network-on-chip (NoC) is a promising paradigm for communication within large system-on-a-chip (SoC) designs. An NoC architecture consists of a network of interconnected nodes, where each node can be a processor core or a specialized IP block. Inter-node communication is performed by the transmission of data packets through routers, which typically have a number of buffers. In this paper, we address a key problem in the design and implementation of NoCs — the minimization of buffer size needed to guarantee a particular quality of service.

Buffers play a critical role in NoC design: increasing the sizes of the buffers can significantly reduce the average latency of packets and hence increase the overall throughput. However, even with scalable communication architectures where routers only exchange data with their neighbors (e.g., [1]), the size of each input-channel buffer has a serious impact on the overall area and power of a NoC router design. For example, Hu and Marculescu [2] indicate that changing the buffer size at each input channel from 2 words to 3 words will increase the overall area by 30% for a 4×4 network. The buffer sizing problem is further complicated by the heterogeneity of traffic patterns in NoCs. For example, one needs to allocate more buffers in a more heavily loaded channel. Hence, we need a way to judiciously allocate buffer capacity for each channel to match the traffic patterns characterizing various applications.

In this paper, we propose a formal technique for *minimizing the cumulative buffer size* while meeting design and performance constraints with respect to specified traffic patterns. Our approach has the following key characteristics. First, we describe an NoC design formally using components drawn from a small set of primitives, similar to the recent XMAS approach [3]. Second, we employ *term-level modeling* [4], [5] to symbolically represent queue sizes and model traffic patterns. Term-level modeling is a technique for representing designs in suitable fragments of first-order logic, which can then be analyzed using satisfiability modulo theories (SMT) solvers [6]. In our model, the traffic injected into the network is non-deterministic but obeys bounds on average rate and burst size. The destination addresses are described using uninterpreted functions. Given this formal model, the third characteristic of our approach is the use of SMT-based model checking to find the minimal buffer sizes that guarantee some throughput and latency for specific traffic patterns. The approach is based on counterexample-guided synthesis, where we repeatedly invoke an SMT-based model checker both for synthesizing buffer sizes and for checking whether the synthesized buffer sizes guarantee the performance property for specified traffic patterns. Buffer sizings found with our approach guarantee performance for all specified traffic patterns, without having to explicitly enumerate each one.

In summary, we make the following novel contributions in this paper.

1. We propose a new SMT-based model checking technique, based on counterexample-guided synthesis, for minimizing the cumulative buffer size in an NoC design.
2. We show how a formal term-level approach can be effective for modeling general NoC designs as well as traffic patterns.
3. We demonstrate the effectiveness of our technique on NoC designs involving arbitration, credit logic, and virtual channels.

Our paper is organized as follows. We review related literature in Section II. Section III describes our formal model for NoC designs and defines the buffer minimization problem. We describe our SMT-based approach in Section IV. Experimental results are presented in Section V and we conclude in Section VI.

II. Related Work

The problem of buffer minimization has been widely studied in the digital signal processing (DSP) community. Synchronous dataflow (SDF) models [7] in particular are used to reason about the minimum buffer size required for a feasible schedule to exist in order for the model to be deadlock-free and to conform to timing constraints [8]. In general, this buffer minimization problem is NP-complete [9].

Various techniques have been applied to address the NP-completeness. Poplavko et al. [10] use an SDF model of an NoC to perform timing analysis and for rate-optimal buffer sizing. Geilen et al [11] use model checking to determine whether there exists a buffer sizing smaller than some bound that admits a deadlock-free schedule; the minimal sizing is found using iterated calls to the model checker. Stuijk et al. [8] present a dynamic programming algorithm that generates a set of candidate buffer sizings that is guaranteed to contain all Pareto-optimal points in the buffer-size versus throughput space; the Pareto-optimal points among the set are then found by self-timed simulation. Wiggers et al [12] approximate minimal buffer sizing for a given throughput using a network-flow formulation, but the closeness of approximation is not bounded.

[1]Joint first authors

978-3-9810801-7-9/DATE11/

SDF can model only a limited class of NoCs. Assumptions of periodic sources and data-independent routing make SDFs well-suited to modeling multimedia NoCs, but not for general-purpose chip multiprocessor (CMP) NoCs. In a CMP, the injected traffic at each node can vary in burst size, have irregular periods, and choose destinations non-uniformly over time. Additionally, due to the lack of support for conditionals, SDFs are not expressive enough to model NoC designs with detailed routing and arbitrary logic.

Thus, analysis of general-purpose CMPs is typicaly based on simulation or probablistic reasoning. Using network analysis, injected traffic with bounded burstiness leads to bounds on required buffer sizes [13]. Stochastic automata networks (SAN) [14] have also been used to model network traffic in SoCs [15]. While SANs allow for efficient reasoning about average case results, they are not suitable for worst-case analysis. Addressing limitations in the probabilistic analysis of stochastic models, adversarial queuing theory has been proposed [16]. If traffic injection is modeled as a Poisson distribution, queuing theory provides a closed form solution to find the buffers most likely to be full [2]. For the same total buffer budget, increasing the size of these oft-used buffers has a better impact on latency than uniformly upsizing [2].

Our approach attempts to find some middle ground between limitations of SDF and the lack of guarantees from simulation-based approach. In particular, the following features differentiate our work from the rest:

- We reason about NoC models at the micro-architectural level. This means that instead of characterizing the network with routing probability at each node, we model the routing logic of each router exactly, with sufficient details of its control flow. This level of abstraction is similar to the Boolean Data Flow (BDF) model, while buffer minimization is more commonly investigated for the less expressive SDF model.
- We present the first SMT-based approach for determining the minimum buffer capacity for a NoC that satisfies the performance properties.

The counterexample-guided approach has been used before, for computing abstract models [17] and for program synthesis [18]. Our paper is the first to adapt this methodology for synthesizing buffer sizes in NoC designs.

III. Formal Model and Problem Definition

A. Formal Definitions

A micro-architectural model of an NoC $\mathcal{N}$ is a tuple $\langle \mathcal{I}, \mathcal{O}, \mathcal{B}, \mathcal{C}, Init \rangle$ where

- $\mathcal{I}$ is a finite set of input signals or *sources*;
- $\mathcal{O}$ is a finite set of output signals or *sinks*;
- $\mathcal{B}$ is a finite set of first-in first-out (FIFO) buffers;
- $\mathcal{C}$ is a finite set of arbitrary logic components, and
- $Init$ is a set of initial states.

A *traffic pattern*, P_i, is associated with each input (source) $i \in \mathcal{I}$, and controls *what* packets are injected into the network and *when*. Over a fixed number of cycles N, P_i is a sequence of pairs $\langle p_{i1}, p_{i2}, \ldots, p_{iN} \rangle$ where $p_{ij} = (b_{ij}, bv_{ij})$. Source i attempts to send a packet into the network for all cycles j such that b_{ij} is **true**. The data value that source i injects on cycle j is bv_{ij} (when $b_{ij} = \textbf{false}$ the value of bv_{ij} is irrelevant). The traffic patterns are generated by an environment, and in some cases contain non-deterministic assignments.

Each output (sink) $o \in \mathcal{O}$ also has an associated traffic pattern P_o. P_o is a sequence of pairs $\langle p_{o1}, p_{o2}, \ldots, p_{oN} \rangle$ where $p_{oj} = (b_{oj}, bv_{oj})$. Sink o attempts to consume a packet on all cycles j such that b_{oj} is **true**. Sink traffic patterns differ from those of sources in that only the b_{ij} variables are generated by the environment, and the assignment to these variables is purely deterministic. The bv_{ij} variables at the sink are passed in the opposite direction, from the network to the environment.

Following the XMAS-style design paradigm [3], each buffer $b_i \in \mathcal{B}$ is parametrized by a size s_i, which is the number of entries that can be stored in it. The components $c \in \mathcal{C}$ are modeled using XMAS primitives whenever possible and arbitrary sequential circuits (finite-state machines) when these are more precise. It is important to note that our component set $\mathcal{C}$ is not limited to just XMAS components. We leverage the XMAS approach mainly because in many cases it provides an elegant and intuitive netlist-style description of NoC designs.

Due to lack of space, we provide here only brief descriptions of the subset of XMAS components that we use. The interested reader is referred to [3]. It is important to note that the components we describe here are slightly modified versions of the components described in [3].

1. *Queue:* parametrized by its size k;
2. *Source:* an interface between the environment and network, a source generates and attempts to send packets as prescribed by its traffic pattern;
3. *Sink:* an interface between the environment and network, a sink attempts to consume packets as prescribed by its traffic pattern;
4. *Fork:* consumes i and produces $a = i$ and $b = i$;
5. *Join:* consumes a and b and produces $o = b$;
6. *Switch:* parameterized by state variable s, consumes i and produces i on a or b depending on the value of s; and,
7. *Merge:* accepts input packets a and b, consumes one of them, produces o.

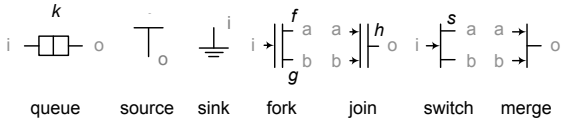

Fig. 1. **XMAS primitives.** The set of XMAS primitives. Inputs and outputs are written in gray and the parameters of the primitives are written in black [3].

B. Modeling Network Traffic

Traffic patterns for NoC designs vary widely depending on the environment in which the NoC is being used. NoC designs used in multimedia applications usually experience more regular traffic patterns. On the other hand, traffic patterns for NoC fabrics within CMP designs are less regular. It is not unreasonable to be concerned with NoC performance for specific classes of traffic patterns [19].

Modeling traffic patterns completely non-deterministically can allow unrealistic scenarios which require larger buffer sizes than are necessary for realistically constrained classes of traffic. For example, in an N-cycle execution, the behaviors of a source controlled by a completely non-deterministic pattern include $p_{ij} = (\mathbf{true}, x)$ for all $i \in \mathcal{I}$ and $0 \leq j \leq N$. This source could attempt to send packets on every cycle leading to an overly pessimistic buffer sizing.

We disallow such pathological worst-case traffic by using regulators to constrain the behavior of otherwise non-deterministic sources. A regulator $\mathcal{R}$ is defined by two parameters: its *rate* ρ and *burstiness* τ.[1] Figure 2 illustrates a source regulator. Source Src_ρ periodically tries to add a packet to the buffer once every ρ cycles, and succeeds whenever the buffer is not full. Src_{ND} generates packets according to its traffic pattern, which allows non-determinism. Whenever source Src_{ND} generates a packet, the packet is transmitted through the merge primitive if the regulator buffer is not empty.

Thus, note that the maximum number of packets sent through the merge in N cycles (after a possible initial burst) is $\lfloor \frac{N}{\rho} \rfloor$. Also, since only up to τ packets can accumulate in the buffer, τ constrains the burstiness of the flow exiting the merge. Specifically, (i) if $\tau \leq \rho$, then the maximum burst size is $\tau + 1$; and (ii) if $\tau > \rho$, then the maximum burst size is $\tau \cdot \frac{\rho}{\rho - 1}$. The latter condition (when $\tau > \rho$) arises since the worst case occurs when the buffer is full with τ packets, and while these packets drain, at most $\frac{\tau}{\rho}$ additional packets can be added, and so on, resulting in the sum of a geometric series.

We refer to a regulator with rate ρ and burstiness τ as a (ρ, τ)-regulator. Combining Src_{ND} with the regulator ensures that even a source with a fully non-deterministic traffic pattern cannot exhibit overly-adversarial behaviors. If the traffic pattern of Src_{ND} is inherently deterministic, then the regulator need not be used. One can record traces of source behaviors for use in future simulations by recording the output of the merge primitive. This trace is then applied directly to an unregulated source in the later simulations.

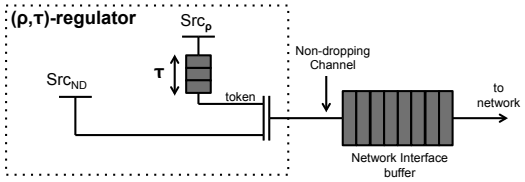

Fig. 2. **Traffic regulator** Average rate of source is constrained by ρ, and burstiness by τ

A *traffic model* $\mathcal{T}_i$ for a (regulated) source i is a set of traffic patterns. Similarly, each sink o has a corresponding set of traffic patterns $\mathcal{T}_o$. The overall traffic model $\mathcal{T}$ for the NoC is the collection of traffic models for all sources and sinks in the NoC. One can represent this formally as the cross product of the sets corresponding to the traffic models of each source and sink.

[1]Our model of traffic regulators is similar to that described by Dally and Towles [19], but it additionally incorporates a non-deterministic source that allows us to explore "bounded" adversarial traffic patterns in a manner suitable for formal verification.

C. Modeling FIFOs

Our model of FIFO buffers follows the queue model presented by Bryant et al. [4], that is based on a combination of the theory of arrays and linear arithmetic [6]. We present a very brief description here and refer the interested reader to [4] for details.

A FIFO of finite, but arbitrary length is modeled as a record Q having components $Q.contents$, $Q.head$, and $Q.tail$. $Q.contents$ is an unbounded array storing FIFO entries. $Q.head$ is an integer expression indicating the index of the head of the queue, i.e., the position of the oldest element in the queue. $Q.tail$ is an integer expression indicating the index at which to insert the next element. We require $Q.head \leq Q.tail$ as an invariant property.

The FIFO is bounded by imposing constraints on $Q.head$, and $Q.tail$. Given a symbolic size s on a FIFO, we impose the constraint that entries can only be added (pushed) to the FIFO if it is not full. To bound a queue to a maximum length of s, our FIFO model always maintains the invariant $Q.tail < Q.head + s$.

As we will see in Section IV-A, an SMT solver can be used to solve for values of buffer sizes s so as to meet performance properties for a set of traffic patterns.

D. Performance Properties

While we can model any performance property that can be formulated as a safety property, we focus on *latency* and *non-dropping*.

A latency constraint, $\phi_{latency}$, is parameterized by a maximum number of cycles x allowed for packet transit. Property $\phi_{latency}$ is **true** in cycle n_i if and only if no buffer slot stores a flit that was injected into the network before time $n_i - x$ and no output channel contains a flit that was injected into the network before time $n_i - x$.

A non-dropping constraint enforces that packets are not dropped on the interface between the environment and network. Property $\phi_{non-drop}$ is **true** on cycle n_i if and only if the network is ready to accept packets from each source that wants to send a packet on cycle n_i. This property is defined by Chatterjee *et al.* [20] (they refer to this as the *non-blocking* property).

A correct NoC must satisfy the following property ϕ in all reachable states :

$$\phi = \phi_{latency} \wedge \phi_{non-drop}$$

E. SMT-based Buffer Size Synthesis

Given a formal NoC model $\mathcal{N}$ as described in Sections III-A and III-C, a traffic model $\mathcal{T}$ as described in Section III-B, and a performance property ϕ as described in Section III-D, the *buffer size synthesis problem* is to compute a buffer sizing S such that the composition of the sized NoC model and the traffic model $\mathcal{T}$ satisfies ϕ.

A *buffer sizing* S is defined as a mapping from buffers to sizes $S : \mathcal{B} \rightarrow \mathbb{N}$, where $S(b_i)$ denotes the size of b_i, and its cumulative size is denoted $|S| = \sum_{b_i \in \mathcal{B}} S(b_i)$. An NoC $\mathcal{N}$ with buffer sizing S is referred to as $\mathcal{N}[S]$, and its composition with $\mathcal{T}$ is denoted $\mathcal{N}[S] \| \mathcal{T}$. Thus, a *correct* sizing of an NoC is one that satisfies $\mathcal{N}[S] \| \mathcal{T} \vDash \phi$.

For a sized NoC $\mathcal{N}[S]$, the problem of determining whether $\mathcal{N}[S]\|\mathcal{T} \models \phi$ can be solved by SMT-based bounded model checking (BMC). The need for SMT arises from the presence of symbolic variables in the composite model $\mathcal{N}[S]\|\mathcal{T}$, including non-deterministic choice variables used to model traffic patterns and abstract terms or uninterpreted functions used to model packet content. When solving this problem from a fixed initial state, bounded model checking is sufficient for this problem (as opposed to unbounded model checking) because the property ϕ is a safety property and the latency constraint in ϕ imposes a finite bound on the number of cycles to explore from an initial state. When solving this problem from an arbitrary initial state, BMC can still be sufficient if k-induction is used with a general initial state.

Building on the above SMT-based verification method, we will next present our SMT-based method for optimal buffer size synthesis.

IV. The CEBUS Approach

Our counterexample-guided buffer size synthesis (CEBUS) approach finds the *minimum cumulative buffer sizing* S such that $\mathcal{N}[S]\|\mathcal{T} \models \phi$, and $|S|$ is minimized among all possible solutions.

We do this by iteratively solving two problems:

1. *Buffer size synthesis (BSS):* For an NoC model $\mathcal{N}$ and a set of traffic patterns $P \subseteq \mathcal{T}$, this step computes a buffer sizing S that has minimal cumulative size $|S|$ among all solutions to $\mathcal{N}[S]\|P \models \phi$.
2. *Buffer size verification (BSV):* This step tries to find a traffic pattern $p \in \mathcal{T}$ that disproves $\mathcal{N}[S]\|\mathcal{T} \models \phi$ where S is computed in the previous BSS step. If such a traffic pattern is found, it is added to set P and BSS is repeated.

As shown in Fig. 3, the iterations between BSS and BSV continue until the process terminates upon finding a minimal buffer sizing that ensures correctness for all traffic pattern, or else finding a set of patterns for which no sizing can ensure correctness.

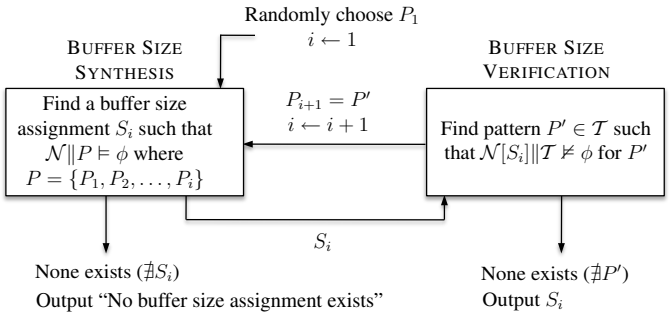

Fig. 3. **Optimal buffer size synthesis with CEBUS.** Starting with an initial traffic pattern P_1, find a buffer size assignment S_i such that $\mathcal{N}[S_i]\|P \models \phi$ holds. Then find a traffic pattern $P' \in \mathcal{T}$ that causes $\mathcal{N}[S_i]\|\mathcal{T} \not\models \phi$. Add the traffic pattern P' to the set of all traffic patterns and repeat the process.

A. Buffer Size Synthesis

The goal of buffer size synthesis (BSS) is to find a buffer sizing S such that $|S|$ is minimum and $\mathcal{N}[S]\|P \models \phi$ for a set of traffic patterns $P \subseteq \mathcal{T}$. BSS is broken down into two steps, satisfiability and minimization. We refer to the satisfiability problem as BSS-SIZE and the minimization problem as BSS.

BSS-SIZE ($\mathcal{N}$,P,$size$) is the problem of determining whether there exists a sizing S such that $\mathcal{N}[S]\|P \models \phi$ holds true and $|S| \leq size$. More precisely, BSS-SIZE($\mathcal{N}, P, size$) returns an S such that $\mathcal{N}[S]\|P \models \phi \wedge |S| \leq size$, if one exists; otherwise, it returns $\perp$.

To solve BSS-SIZE, note that we only require a satisfiability check since P is a finite set of traffic patterns, and we can construct the SMT formula as the conjunction of BMC unrollings for the patterns in P.

A solution to BSS-SIZE, if one exists, is a buffer sizing S such that $\mathcal{N}[S]\|P \models \phi$, but this S is not necessarily the minimum solution. In order to find the minimum sizing S such that $\mathcal{N}[S]\|P \models \phi$, BSS performs a binary search over $size$ using a sequence of calls to BSS-SIZE ($\mathcal{N}$,P,$size$). Algorithm 1 shows pseudo code for BSS. Intermediate variables $MinBufSize$ and $BufSizeLB$ keep track of the minimum buffer size seen thus far where ϕ holds, and the maximum buffer size such that $\neg\phi$, respectively. This procedure terminates when $MinBufSize$ is the minimum cumulative buffer size. At this point, $MinBufSize = BufSizeLB + 1$, and there does not exist an S smaller than $MinBufSize$ such that $\mathcal{N}[S]\|P \models \phi$.

Algorithm 1 Procedure BSS($\mathcal{N}, P$): Compute minimum cumulative buffer size for NoC model $\mathcal{N}$ and set of traffic patterns P.

```
1:  MinBufSize = UB        {UB is a known upper bound}
2:  BufSizeLB = 0
3:  while BufSizeLB + 1 ≠ MinBufSize do
4:      size = (MinBufSize − BufSizeLB)/2 + BufSizeLB
5:      S = BSS-SIZE(N, P, size)
6:      if S ≠ ⊥ then
7:          MinBufSize = |S|
8:      else
9:          BufSizeLB = size
10:     end if
11: end while
```

B. Buffer Size Verification

The solution to BSS provides a minimal buffer sizing S such that $\mathcal{N}[S]\|p \models \phi$ for all traffic patterns $p \in P$, but this sizing may not ensure correctness for all $p \in \mathcal{T}$. Buffer size verification (BSV) addresses this by trying to find a traffic pattern $p \in \mathcal{T}$ that disproves $\mathcal{N}[S]\|\mathcal{T} \models \phi$

BSV ($\mathcal{N}[S]$,$\mathcal{T}$) is the problem of determining whether $\mathcal{N}[S]\|\mathcal{T} \not\models \phi$. As noted earlier, this can be solved as a BMC problem. A solution (satisfying assignment) to this problem is a traffic pattern $p \in \mathcal{T}$ that causes $\neg\phi$.

If BSV has no solution (returns $\perp$) then no traffic pattern exists for sized NoC $\mathcal{N}[S]$ that will cause ϕ to fail. If BSV has a solution, it produces a traffic pattern for which the sized NoC is insufficient for meeting the performance property, and a new sizing will need to be synthesized.

C. Optimal Buffer Sizing

By combining BSS and BSV we are able to tackle the problem of finding the minimum cumulative buffer size for NoC model $\mathcal{N}$ and traffic patterns $p \in \mathcal{T}$. We refer to this procedure as CEBUS.

Let $P_{init} \subseteq \mathcal{T}$ be the initial set of traffic patterns. CEBUS starts by calling BSS ($\mathcal{N}$,P) with $P = P_{init}$ in order to obtain the minimum buffer sizing S such that $\mathcal{N}[S] \| P \vDash \phi$ holds. Next, BSV ($\mathcal{N}[S]$,$\mathcal{T}$) is called to find a traffic pattern $p \in \mathcal{T}$ such that $\mathcal{N}[S] \| \mathcal{T} \nvDash \phi$. Let the traffic pattern found in BSV ($\mathcal{N}[S]$,$\mathcal{T}$) be $P_{violate}$. Add $P_{violate}$ to the set of traffic patterns P and repeat this process until CEBUS terminates due to one of the following reasons: 1) BSS ($\mathcal{N}$,P) returns $\perp$, which means no buffer size exists for the current set of traffic patterns P; or 2) BSV ($\mathcal{N}[S]$,$\mathcal{T}$) returns $\perp$, meaning that no traffic pattern exists that causes $\mathcal{N}[S] \| \mathcal{T} \nvDash \phi$. Algorithm 2 gives the pseudo code for this procedure.

Algorithm 2 Procedure CEBUS $(\mathcal{N}, \mathcal{T})$: Compute the optimal buffer sizing for NoC model $\mathcal{N}$ and traffic model $\mathcal{T}$.

```
P = P_init where P_init ⊆ T
while true do
    S = BSS(N, P)
    if S ≠ ⊥ then
        P_violate = BSV(N[S], T)
        if P_violate ≠ ⊥ then
            P = P ∪ P_violate
        else
            ∄P_violate ∈ T such that N[S] ∧ ¬φ
            Return "success, optimal buffer sizing found"
        end if
    else
        ∄S such that φ for all P ∈ T.
        Return "failure, no buffer sizing exists"
    end if
end while
```

V. Experimental Results

We performed two case studies to evaluate our approach. The first is a small example which uses a credit-based scheme to control packet flow through a router. The second example sends traffic through a more realistic chip-multiprocessor (CMP) router. In each example, we verify the performance properties described in Section III-D.

Our experiments were performed by creating UCLID [21] models of the example circuits using the ATLAS [5] system. The UCLID verifier is then used to perform bounded-model checking within the CEBUS loop, using its internal SMT solver using the MiniSat SAT solver as a back end. Experiments were run on a Linux workstation with 64-bit 3.0 GHz Xeon processors and 2 GB of RAM.

A. Credit Logic

A common NoC design pattern is the use of credit logic to control channel usage within a router. Figure 4 shows a simple example using credit-logic where master router M sends tokens to target router T. M can send a packet to T when one or more entries are in the credit FIFO in M. T controls the maximum number of outstanding tokens by sizing the FIFO in the credit logic. If the size of this outstanding tokens FIFO is C, then no more than C packets can be in M and T. A non-pipelined delay of 7 cycles is added to the channel carrying tokens from T to M.

The setup for the credit logic experiment is as follows. The traffic model is $(\rho, \tau) = (5, 4)$, and the maximum allowed latency is 4 cycles. Both the network interface and ingress queues begin empty; the token queue in the regulator, and the token queues for outstanding tokens and available credits are all initially filled to their maximum respective capacities. The traffic model allows a burst of up to 4 packets to be sent. Because it takes 7 cycles for a token to be returned, and the latency bound is 4 cycles, it is clear that the latency property $\phi_{latency}$ will be violated if each packet must wait for the returned token from the previous packet (and the non-dropping property $\phi_{non-drop}$ precludes any solutions that would side-step the problem by simply dropping packets). Table I shows the CEBUS problem sizes and runtimes for each iteration of the credit logic example. The the size of the CNF problem for solving BSS increases as a larger set traffic patterns is considered at each iteration, while the size of the CNF problem for solving BSV is the same for all iterations.

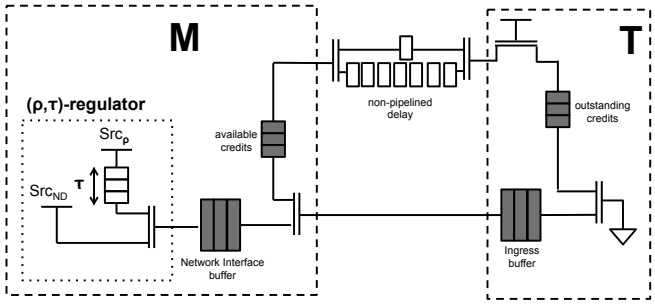

Fig. 4. **Credit-based NoCs.** This example shows how to model credit-logic commonly used in NoCs. The FIFOs that are being sized are shaded.

B. Single Router

This second buffer sizing case study is a chip-multiprocessor (CMP) router model based on a design by Peh [22]. This CMP design is an on-chip communication fabric that can be used to connect processors, memories, and other IP blocks to one another. While the original CMP design has 5 input and 5 output ports, we use a simplified version with 2 input ports and two output ports. The router's function is to direct incoming packets to the proper output port. When head flits are at the front of either ingress FIFO the router reserves the output channel, grants it to one of the packets, and updates the priority table. The output channel remains reserved for a particular packet until that packet's tail flit passes through the router signaling to the router than the channel is available again.

The original CMP router is not an XMAS-style design. So, we constructed a CMP router where the datapath is made purely out of XMAS components, and the control logic is made of XMAS components when convenient. Each of the two inputs and each of the two outputs comprises a credit logic loop, as shown in Fig. 4.

The experiment setup is as follows. As in the credit logic example,

Benchmark	Iteration i	$\|S\|$	*Buffer-Size Synthesis (BSS)*						*Buffer-Size Verification (BSV)*					
			CNF Size		Runtime (sec)				CNF Size		Runtime (sec)			
			Vars	Clauses	SAT	Enc	SSim	Total	Vars	Clauses	SAT	Enc	SSim	Total
Credit	0	0	-	-	-	-	-	-	100K	299K	0.29	7.29	0.32	7.90
	1	2	2K	8K	0.05	14.72	2.04	16.81	100K	299K	0.52	7.22	0.33	8.07
	2	11	36K	109K	0.74	31.84	8.19	40.77	100K	299K	0.48	7.38	0.33	8.19
	3	15	68K	206K	1.52	32.84	9.51	43.87	100K	299K	2.14	7.54	0.32	10.00
	4	17	153K	459K	4.80	79.01	14.03	97.84	100K	299K	1.68	7.39	0.34	9.41
	5	21	240K	720K	9.70	118.18	21.82	149.70	100K	299K	7.57	7.44	0.32	15.33
	6	25	303K	908K	11.96	164.16	29.83	205.95	100K	299K	9.14	6.64	1.23	17.01
CMP	0	0	-	-	-	-	-	-	151K	454K	0.48	16.01	0.86	17.35
	1	10	62K	185K	6.38	56.18	5.56	68.12	151K	454K	0.57	15.85	0.82	17.24
	2	14	152K	455K	17.99	121.90	12.85	152.74	151K	454K	0.64	15.80	0.88	17.32
	3	21	214K	643K	22.03	141.17	19.49	182.69	151K	454K	1.33	15.90	0.86	18.09
	4	26	271K	814K	29.78	144.27	22.81	196.86	151K	454K	1.77	15.78	0.81	18.36
	5	38	422K	1265K	97.70	270.12	51.41	419.23	151K	454K	4.23	15.80	0.84	20.87

TABLE I

CEBUS experiments On each iteration i, BSS uses a sequence of calls to BSS-SIZE in order to find the minimal sizing S that is correct for all patterns in set P_i, and BSV finds a counterexample traffic pattern that demonstrates $\mathcal{N}[S]\|\mathcal{T} \not\models \phi$. The remaining column headings are as follows: Benchmark denotes corresponding benchmark; Iteration is the iteration number for BSS and BSV, respectively; $|S|$: the minimum cumulative buffer size for the current iteration in BSS and the candidate buffer size in BSV; Vars and Clauses: represent the number of variables and clauses, respectively, in the corresponding SAT problem(s); SAT is the time spent in satisfiability solving; Enc is the time taken encoding the word-level input to CNF; SSim is the time taken during symbolic simuation; Total is the overal time spent in the respective iteration. Note that the problem sizes for BSS are the average problem sizes for all satisfiability calls in the current iteration. Additionally, the times for BSS are the total time spend in each step over all calls to the verification engine during the current iteration.

all flit buffers are initially empty, and credit and outstanding token buffers are initially filled. Each path where tokens are returned has a 7-cycle non-pipelined delay. Unlike the previous example, the distinction between head flits and tail flits is important for this example. To accomodate this, the traffic model in this example is slightly modified from the previous one. Each source now injects head flits and tail flits, with a tail flit always immediately following a head flit. The destination addresses of head flits are generated from uninterpreted functions. In this example, each of the two sources is a (8,1)-regulated source, and only head flits consume a token from the regulator's queue. The latency bound checked by property $\phi_{latency}$ is 12 cycles. Table I shows the problem sizes and runtimes for the CMP router experiment.

VI. Conclusion

We have presented a new approach for minimizing the cumulative buffer size in on-chip networks, so as to meet throughput and latency requirements, given high-level specifications on traffic behavior. Our approach uses model checking based on satisfiability modulo theories (SMT) solvers, within an overall counterexample-guided synthesis loop. Experimental results on NoC designs show the promise of the proposed technique.

Acknowledgments. This research was supported in part by SRC contract 2045.001, an Alfred P. Sloan Research Fellowship, the UC MICRO program, and the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

References

[1] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *IEEE Symposium on VLSI*, 2002, pp. 117–124.

[2] J. Hu and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 354–361.

[3] S. Chatterjee, M. Kishinevsky, and U. Ogras, "Quick formal modeling of communication fabrics to enable verification," in *HLDVT '10*, 2010.

[4] R. E. Bryant, S. K. Lahiri, and S. A. Seshia, "Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions," in *Proc. Computer-Aided Verification (CAV'02)*, ser. LNCS 2404, July 2002, pp. 78–92.

[5] B. A. Brady, R. E. Bryant, S. A. Seshia, and J. W. O'Leary, "ATLAS: automatic term-level abstraction of RTL designs," in *Proc. the Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, July 2010.

[6] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 4, ch. 8.

[7] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[8] S. Stuijk, M. Geilen, and T. Basten, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," *Computers, IEEE Transactions on*, vol. 57, no. 10, pp. 1331 –1345, oct. 2008.

[9] S. Bhattacharyya, P. Murthy, and E. Lee, *Software synthesis from dataflow graphs*. Springer, 1996.

[10] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman, "Task-level timing models for guaranteed performance in multiprocessor networks-on-chip," in *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*. New York, NY, USA: ACM, 2003, pp. 63–72.

[11] M. Geilen, T. Basten, and S. Stuijk, "Minimising buffer requirements of synchronous dataflow graphs with model checking," in *DAC '05: Proceedings of the 42nd annual Design Automation Conference*. New York, NY, USA: ACM, 2005, pp. 819–824.

[12] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Efficient computation of buffer capacities for cyclo-static dataflow graphs," in *DAC '07: Proceedings of the 44th annual Design Automation Conference*. New York, NY, USA: ACM, 2007, pp. 658–663.

[13] R. Cruz, "A calculus for network delay, part II: Network analysis," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 132–141, 1991.

[14] B. Plateau and K. Atif, "Stochastic automata network of modeling parallel systems," *IEEE Trans. Softw. Eng.*, vol. 17, no. 10, pp. 1093–1108, 1991.

[15] R. Marculescu, U. Ogras, and N. Zamora, "Computation and communication refinement for multiprocessor SoC design: A system-level perspective," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 3, p. 592, 2006.

[16] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson, "Adversarial queuing theory," *J. ACM*, vol. 48, no. 1, pp. 13–38, 2001.

[17] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *12th International Conference on Computer Aided Verification (CAV*.

[18] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. A. Saraswat, "Combinatorial sketching for finite programs," in *Proc. 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM Press, 2006, pp. 404–415.

[19] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[20] S. Chatterjee and M. Kishinevsky, "Automatic Generation of Inductive Invariants from High-Level Microarchitectural Models of Communication Fabrics," in *Proc. of Intl. Conf. on Computer Aided Verification*, 2010.

[21] "UCLID Verification System." Available at `http://uclid.eecs.berkeley.edu`.

[22] L.-S. Peh, "Flow control and micro-architectural mechanisms for extending the performance of interconnection networks," Ph.D. dissertation, Stanford University, August 2001.