Scalable Progress Verification in Credit-Based Flow-Control Systems

Sayak Ray, Robert K. Brayton
Department of EECS, University of California, Berkeley
{sayak, brayton}@eecs.berkeley.edu

***Abstract*—Formal verification of liveness properties of practical communication fabrics are generally intractable with present day verification tools. We focus on a particular type of liveness called 'progress' which is a form of deadlock freedom. An end-to-end progress property is broken down into localized safety assertions, which are more easily provable, and lead to a formal proof of progress. Our target systems are credit-based flow-control networks. We present case studies of this type and experimental results of progress verification of large networks using a bit-level formal verifier.**

I. Introduction

Micro-architectural inter-connection networks *a.k.a.* communication fabrics are prone to deadlock. There have been extensive studies over decades on how to avoid deadlock while designing communication fabrics. As a result, today we know an impressive compendium of techniques to design deadlock-free communication fabrics. Books like [6], [8] have become standard references on this topic.

Although designing a deadlock-free communication fabric has become almost a routine task now, it may be noted that the techniques used for the purpose work at the micro-architecture level. As a result, fabrics are proved to be deadlock-free only at that level. But the actual bit-level implementation derived from the micro-architectural model still remains vulnerable to deadlock. This bug could be due to implementation or compilation error, or due to some scenario that was missed during the manual proof process worked out at the micro-architecture level. It is, therefore, very important to have a scalable proof technique and tool that can prove deadlock freedom of such bit-level, detailed implementations. Unfortunately, very little work has been done for automatically proving deadlock-freedom of the actual bit-level implementations of these communication fabrics. Random simulation is typically used to ensure correctness of these implementations. But as we know, simulation is an incomplete technique which is good for bug-hunting, but not for proving correctness.

In this work, we are proposing a *scalable methodology* for proving deadlock-freedom of bit-level implementations of a collection of communication fabrics that use *credit based flow control*. It is a switch-level flow control technique, widely used across various types of communication fabrics for avoiding deadlock, especially in wormhole switching systems (See Chapter 13 of [6] for reference). In this paper, we will focus on two categories of fabrics as case studies, viz. *virtual channels* and *hardware scoreboards*. These fabrics are heavily used as micro-architectural idioms in complex hardware systems. We will demonstrate how the high-level knowledge that these circuits use credit based flow control can be leveraged to produce a scalable proof of their deadlock freedom. In this work, we are interested in one particular formal specification of deadlock property as explained in detail in Section V. It captures an end-to-end progress behavior of the fabrics. We will use the terms 'deadlock freedom' and 'progress' interchangeably in this paper, though our solution will strictly adhere to the formal specification in Section V.

The methodology that we are proposing in this paper is based on the principle of 'proving *liveness* using intermediate *safety* properties'. The reason we took this approach is the following: a generic progress property of a bit-level implementation of any communication fabric can be expressed as a liveness property using a suitable temporal logic (*eg.* linear temporal logic or LTL), and model checkers may be used to verify the property. There are many academic and industrial model checkers that are actively being used in practice for formal verification of hardware designs. But it turns out that these tools have extensive support for safety verification, whereas have only a rudimentary support for liveness, if any. While some kind of structural deadlock may be expressed as safety property, generic formulation of deadlock freedom calls for liveness verification support. This renders the current state-of-the-art formal verification tools ineffective for liveness verification of communication fabrics. The main reasons of failure of available tools on these fabrics are the huge state-spaces of these fabrics, and inherent algorithmic complexity of liveness verification as implemented in the current tools. This motivates us to approach our problem of liveness verification from an alternative stand-point. We will demonstrate that the end-to-end progress property of a fabric can be broken down into a collection of safety properties which are potentially easy-to-verify obligations, and these safety properties, once proved on the designs, together imply the overall progress property.

Contributions: Our contributions in this work are summarized as follows:

1) We are showing how structural and functional safety properties of communication fabrics can be derived and leveraged for verification of their liveness properties. This work demonstrates how we can capture architect's insight (why she thinks that her fabric will not deadlock), and use them to prove progress properties using model checkers. This approach offers a scalable verification solution.
2) Our liveness verification framework is publicly available, and it works on models written in industry standard language like Verilog or standard bit-level description formats like AIGER. We believe that this tool will bridge the gap of not having a bit-level analysis tool for communication fabrics, where high-level theorem provers or graph theoretic reasoning tools have remained as the only options as proof

978-3-9810801-8-6/DATE12/

engines.

The paper is organized in the following sections. Section II presents related works. Section III introduces the formal model used for modeling communication fabrics. Section IV briefly explains credit based flow control and one of its properties (BUFFER RELATION) that will play an important role in the subsequent sections. Section V presents the mathematical formulation of the progress property that we want to verify. Section VI presents our methodology of proving liveness properties using intermediate safety invariants on virtual channels. The same methodology is demonstrated on hardware scoreboards in Section VII. Section VIII discusses our experiments. At the end, Section IX outlines future works and concludes this paper.

II. Related Work

Our work may be viewed as a blend of model checking and theorem proving. While theorem proving approach has been widely used for deadlock verification of network-on-chips [11], literature based on bit-level model checking is rather sparse in this domain. We have addressed some case studies which are not regular in structure, whereas most of the existing theorem proving techniques rely on the regularity of fabric structures. Petri net based techniques are also quite common in literature [2], but our formalism adheres to the current industry practice which is distant from Petri net oriented modeling. [9] is perhaps the closest to our work. The major difference between [9] and our work is that [9] is a bottom-up approach and ours is a top-down approach. The most significant benefit of our approach is that it leverages a way of formally capturing designer's own conviction for deadlock freedom. Hence, it is a more natural way of verification, bugs found in our approach will be more meaningful to the designer, and as a result, debugging will be easier.

III. Formal Model

Communication fabrics are typically constructed using finite *FIFO buffers*, *sources* and *sinks* of flits [1], *function blocks* acting on flits, decision primitives like *switches* and *arbiters*, and synchronization primitives like *forks* and *joins*. While these structural components can have various implementations, we will follow a particular formal model called xMAS [5]. xMAS provides a clear logical specification of all these components. Fabrics designed in this formalism become synchronous Boolean circuits triggered by a single global clock. Due to lack of space, we refer to [5] for details of the formal specification of the components. Here we provide only an informal summary of the semantics of the components, with their symbols shown in Figure 1. A *fork* takes a flit from its input, and produces two new flits in two outputs, while a *join* does the dual operation. An *arbiter* arbitrates between two of its inputs, breaking ties as per its own (*fair*) arbitration policy. A *switch* sends the input flit to either of its outputs based on the type of data in the flit. *Sources* and *sinks* are producers and consumers of flits respectively. All sources and sinks considered in this paper are *non-deterministic*. *Function* units are computation blocks that may transform the control and/or data part of the flit. *FIFO buffers* are standard first-in-first-out buffers that can store finite number of flits. A communication fabric is made of these components by connecting them with *channels*. Each channel consists of three kinds of signals, viz. *req, gnt*, and *data*. A channel has one xMAS component (called initiator) on one side that initiates a transaction, and another component (called target) on the other side that receives the transaction. *req* and *data* signal go from initiator to target, and *gnt* goes from target to initiator. These signals are persistent, *i.e.* if a *req* is asserted, it remains asserted until the corresponding *gnt* is asserted. We will rely upon this property heavily during our proofs. A fair understanding of the semantics of xMAS components is assumed in the rest of the paper.

[1] flits are units of data transfer in communication fabrics, see [6] for details

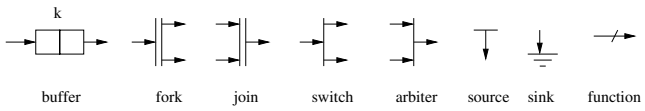

Fig. 1. xMAS symbols for structural components of communication fabrics

IV. Credit Mechanism

Credit based flow control is perhaps the most popular technique for switch-level flow control, especially in wormhole switching systems. Figure 2 illustrates the basic credit based flow control mechanism where a source (*data_source*) wants to send flits to a sink (*data_sink*) through a buffer (B_2), and the flow of packet across B_2 is controlled by the *Credit_logic* sub-circuit. Main purpose of *Credit_logic* is to allow only a restricted number of flits to enter the system which it is guarding (in this case, the buffer B_2). The particular synchronization achieved by forks and joins in Figure 2 gives rise to the following invariant:

$$num(B_1) + num(B_2) = num(B_c)$$

where $num(B)$ denotes the number of flits that is currently residing in buffer B. We will call this relation BUFFER RELATION in the subsequent sections. While Figure 2 is a very simple example of credit based flow control, the same principle is used in more complicated industrial designs. In Section VI and Section VII, we will discuss two different, but more complicated, circuit families that leverage the same credit principle shown in Figure 2. Interestingly, BUFFER RELATION with suitable modifications continues to hold on all such credit based circuits.

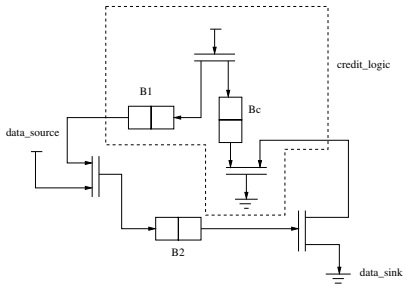

Fig. 2. Credit Mechanism

We mention here that these BUFFER RELATIONs, as first identified in [4], are remarkably important for bit-level verification of safety properties of credit based systems. These seemingly intuitive relations are quite non-trivial to mine from a bit-level implementation of a fabric, unless they are explicitly hinted by the architects. These relations have been used to expedite safety verification so far [4]. In the context of progress verification,

it was not immediately clear how BUFFER RELATIONS can be leveraged. Our main contribution in this work is to demonstrate how other intermediate invariants can be deduced as corollaries of BUFFER RELATION that can lead to a formal proof of progress properties.

V. PROGRESS FORMULATION

The mathematical formulation of the progress property that we are interested in is presented in this section. Instead of introducing the formalism in an abstract set up, we choose to illustrate it around an example of virtual channel. [2]

A. Formulation of Deadlock

Consider the model shown in Figure 3. It is an implementation of virtual channel, and represented using xMAS notations. Here a single physical channel (channel e) is shared between two sources (sources A_1 and A_2). Virtual channel is a fundamental building block of on-chip communication networks which was invented to mitigate head-of-line (HOL) blocking problem in wormhole switching [7]. See texts like [6], [8] for details. Figure 3 shows how two sources A_1 and A_2 share the virtual channel e to transfer flits to their respective sinks ($sink_1$ and $sink_3$ respectively) while credit logic blocks CL1 and CL2 control flow of flits from source A_1 and A_2 respectively.

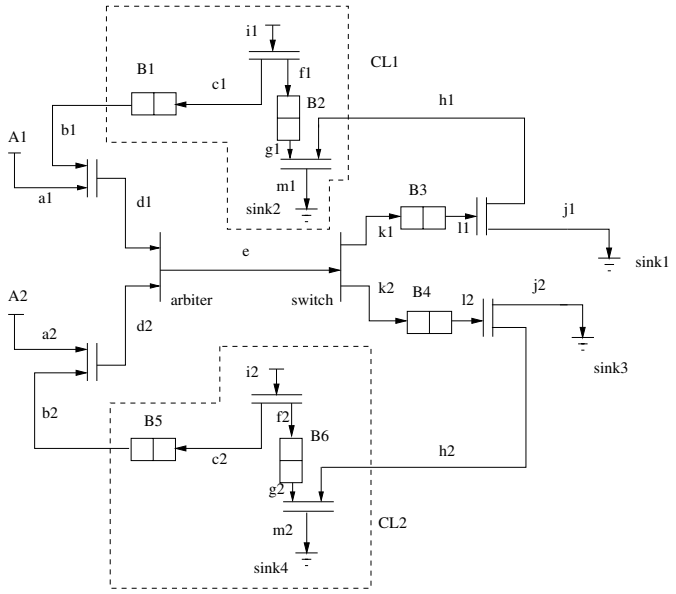

Fig. 3. Virtual Channel

As per xMAS formalism, each channel c consists of three types of signals, viz. request ($c.req$, 1-bit signal), grant ($c.gnt$, 1-bit signal), and data ($c.data$, bit-vector signal). In Figure 3, the sender at the source A_1 asserts the signal $a_1.req$ when it wants to send a flit. When the network is ready to accept the flit from A_1, it will assert the signal $a_1.gnt$. Therefore, the designer's objective will be to prove that whenever the source A_1 makes a request to send a packet, it will be eventually granted. This will prove that A_1 will never be blocked. In LTL, it can be written as $\mathbf{G}(a_1.req \Rightarrow \mathbf{F}(a_1.gnt))$. It is a well-known liveness property. While model checking this property, one needs to make some *fairness assumptions* on the sinks and (some of) the sources that they will never cease of work. Otherwise, one may come up with a rather uninteresting buggy scenario where some request from A_1 is never granted simply because one or more sinks stopped to drain flits, or credit source in CL1 stopped to supply credits. This contradicts a basic assumption that the sinks and the sources model the (non-deterministic) environment that may not react immediately, but will not cease to react either. For the virtual channel of Figure 3, these necessary fairness assumptions may be written in LTL as $\mathbf{GF}(m_1.gnt)$, $\mathbf{GF}(m_2.gnt)$, $\mathbf{GF}(j_1.gnt)$, and $\mathbf{GF}(j_2.gnt)$ for the sinks, and $\mathbf{GF}(i_1.req)$ and $\mathbf{GF}(i_2.req)$ for the credit sources. The overall proof objective $\phi_{progress}$ (or ϕ_p in short) is, therefore,

$$\phi_p := ((sink\_fair \wedge source\_fair) \Rightarrow progress)$$

where

$$\begin{aligned} sink\_fair &:= \mathbf{GF}(m_1.gnt) \wedge \mathbf{GF}(m_2.gnt) \\ &\quad \wedge \mathbf{GF}(j_1.gnt) \wedge \mathbf{GF}(j_2.gnt) \\ source\_fair &:= \mathbf{GF}(i_1.req) \wedge \mathbf{GF}(i_2.req) \\ progress &:= \mathbf{G}(a_1.req \Rightarrow \mathbf{F}(a_1.gnt)) \end{aligned}$$

Same formulation applies to channel A_2 as well. Note that the above formulation treats the design almost as a blackbox, and does not refer to its internal signals or topology. Therefore, by careful selection of fairness constraints, the above generic formulation can be easily applied to any other xMAS design.

Discussion : The above generic formulation is a very popular way of expressing progress property, mainly due to the neat way it captures designer's intent. Unfortunately, the price that this neat specification has to pay comes from the complexity of its verification algorithm. Traditional algorithms used to verify the above property would use either nested fixpoint based approach or cycle detection based approach. Both of them are prohibitively inefficient on real designs. These algorithms hardly converge unless the design is extremely simple. Liveness-to-safety conversion [10] is an alternative to these traditional algorithms that promises more scalability. But it also failed to converge on our experiments with real communication fabrics. All these existing approaches turn out to be grossly unscalable for our applications. This motivates us to investigate further, and find out fine-grained ways of making liveness verification scalable for communication fabrics. We have thus chosen to split the end-to-end liveness verification problem into a set of intermediate safety verification problems as presented below.

VI. VIRTUAL CHANNELS

We begin with the basic virtual channel shown in Figure 3. We will demonstrate how BUFFER RELATIONs can be leveraged to derive auxiliary safety assertions for this design, and how satisfaction of these auxiliary safety assertions leads to a proof of ϕ_p. BUFFER RELATIONs for Figure 3 are the following:

$$num(B_1) + num(B_3) = num(B_2)$$

$$num(B_4) + num(B_5) = num(B_6)$$

Leveraging these relations, and by careful analysis of the xMAS model of the fabric, we have derived four safety properties; they

[2]The formulation is uniform across the entire family of fabrics, and has no particular relation to the virtual channel design or implementation that we are considering here. For example, the same formulation applies to all other fabrics considered in this paper.

are tabulated in Table I as Lemmas 1 through 4. These properties are supposed to hold on the bit-level implementation of the design, and a safety verification engine may prove them on the bit-level model. We will discuss our experiments and observations later in Section VIII. We will demonstrate at the end of this section how satisfaction of Lemmas 1 through 4 contribute to the formal proof of ϕ_p.

We now introduce below (Theorem 1) another safety property called '*non-blocking property of channel* e'. It is a non-trivial property, perhaps not apparent from the design immediately; it is rather a consequence of carefully architecting the fabric using credit based flow control (or to put in other words, it is a consequence of BUFFER RELATIONS). It contributes significantly to the deadlock freedom of the fabric in Figure 3.

Theorem 1 (Non-blocking e): Whenever $e.req$ is asserted, $e.gnt$ is asserted in the same cycle. In LTL, $\mathbf{G}(e.req \Rightarrow e.gnt)$.

Justification: suppose $e.req$ is asserted in some cycle. Since it is coming from the arbiter, $e.req$ must correspond to either $d_1.req$ or $d_2.req$. Without any loss of generality, let us assume that it corresponds to $d_1.req$. Which implies both $a_1.req$ and $b_1.req$ are asserted in that cycle, and buffer B_1 has at least one flit in it. Since due to buffer relation $num(B_1) + num(B_3) = num(B_2)$ and $Size(B_2) = Size(B_1) = Size(B_3)$, it is easy to see that buffer B_3 has at least one slot free. Therefore, $k_1.gnt$ must be asserted in that cycle, and in turn, $e.gnt$ gets asserted in the same cycle satisfying the property $e.req \Rightarrow e.gnt$. The argument holds for every cycle irrespective of whether the arbiter schedules d_1 or d_2, making $\mathbf{G}(e.req \Rightarrow e.gnt)$ an invariant. □

We will show how this non-blocking property, in association with other safety properties, can lead to a formal proof of ϕ_p. This non-blocking property of credit-based virtual circuit is a well-understood property to an architect, but has not been leveraged for formal verification of progress property so far. This non-blocking property alone, in spite of being the key behind deadlock freedom of virtual channel, does not immediately lead to a formal proof of ϕ_p. In order to bridge this gap, we need additional invariants. We claim that Lemmas 1 through 4 is such a set of auxiliary safety invariants that are sufficient to achieve a formal proof of satisfaction of ϕ_p starting from Theorem 1. The following theorem justifies this claim.

Theorem 2: If the model shown in Figure 3 satisfies Theorem 1, and Lemmas 1 through 4, then it also satisfies ϕ_p.

Proof: In order to show that ϕ_p holds on the fabric, it is *sufficient* to show that the signal $b_1.req$ is asserted infinitely often. This is a sufficient condition since whenever $a_1.req$ is asserted, if it is eventually accompanied by an assertion of $b_1.req$, then $d_1.req$ will be asserted. Since the arbiter is fair and every $e.req$ is immediately granted (due to Theorem 1), $d_1.req$ will be eventually scheduled and $d_1.gnt$ will be asserted. Hence, $a_1.gnt$ will be asserted in turn. This shows why every $a_1.gnt$ will be eventually granted if $b_1.req$ is asserted infinitely often.

Now by fairness assumptions, gnt signals of $sink_1$ and $sink_2$ will be asserted infinitely often. Suppose the gnt signal of $sink_1$ (i.e. $j_1.gnt$) is asserted in some time step t_1 and that of $sink_2$ (i.e. $m_1.gnt$) is asserted in some other time step t_2. Without any loss of generality, let us assume that $t_1 < t_2$. Now, due to persistence of signal and due to the logic of the fork, $j_1.gnt$ will remain asserted till time step t_2. Then due to Lemma 1, one empty slot will be created in B_2 *by* cycle $t_2 + 1$. By Lemma 2, this creates an empty slot in B_1. By Lemma 3, these empty slots will be preserved if the credit source of CL1 does not push a credit. By fairness of the credit sources, a credit will be eventually pushed, and by Lemma 4, this will result in an assertion of $b_1.req$. Due to the nature of the fairness constraints, this chain of events will never cease to work. Hence, $b_1.req$ will be asserted infinitely often. □

A. Variants of Virtual Channel

The same principle of BUFFER RELATION and intermediate safety assertions works for proving ϕ_p for more complex virtual channels as well. Figure 4 and Figure 5 represent buffered virtual channel, and virtual channel with ordering respectively. Buffered virtual channel is the most common form of virtual channel used in network-on-chips, whereas virtual channel with ordering are used in cache interface logic. For descriptions of working principles of these xMAS models, see [5]. We claim that all Lemmas 1 through 4, and Theorem 1 hold on these designs as well. For Theorem 1, the channel of interest is the one that leaves the arbiter in both Figure 4 and Figure 5. We also claim that theorems analogous to Theorem 2 can be formulated (as presented below) and proved for these complex virtual channels. But the formulation of BUFFER RELATIONS will change depending on the structures of the fabrics.

Theorem 3: If the model shown in Figure 4 satisfies Theorem 1, and Lemmas 1 through 4, then it also satisfies ϕ_p.

Theorem 4: If the model shown in Figure 5 satisfies Theorem 1, and Lemmas 1 through 4, then it also satisfies ϕ_p.

Due to lack of space, justification of these theorems are left to the reader as they are analogous to the case of simple virtual channel. We include verification results of Theorem 1 and Lemmas 1 through 4 for these fabrics in Section VIII.

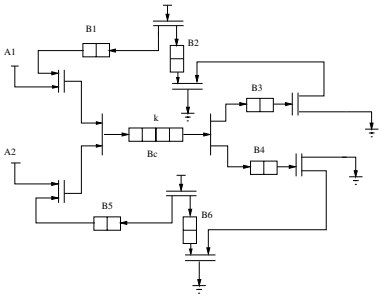

Fig. 4. Buffered virtual channel

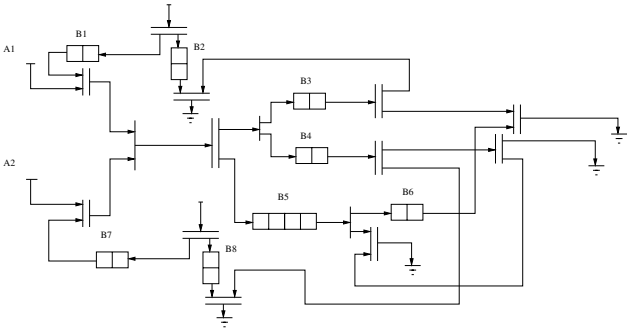

Fig. 5. Virtual Channel with ordering

We conclude this section by presenting an example (Figure 6) where the virtual channel actually deadlocks, and does not satisfy

Lemma	Statement	LTL formulation
Lemma 1	If both $sink_1$ and $sink_2$ are ready to drain a flit, then either buffer B_2 gets at least one empty slot in the next cycle or $sink_1$ and $sink_2$ persist their states in the next cycle	$j_1.gnt \wedge m_1.gnt \Rightarrow \mathbf{X}(not\_full(B_2) \vee (j_1.gnt \wedge m_1.gnt))$
Lemma 2	If B_2 has at least one empty slot, it implies that B_1 also has at least one empty slot	$not\_full(B_2) \Rightarrow not\_full(B_1)$
Lemma 3	For $i = 1, 2$, if B_i has an empty slot and the credit source is not ready to push a flit, the empty slot is preserved in the next cycle	$not\_full(B_i) \wedge !i_1.req \Rightarrow X(not\_full(B_i))$
Lemma 4	If both B_1 and B_2 have empty slots and the credit source is ready to push a flit, then in the next cycle $b_1.req$ must be asserted	$not\_full(B_1) \wedge not\_full(B_2) \wedge i_1.req \Rightarrow X(b_1.req)$

TABLE I
AUXILIARY SAFETY ASSERTIONS

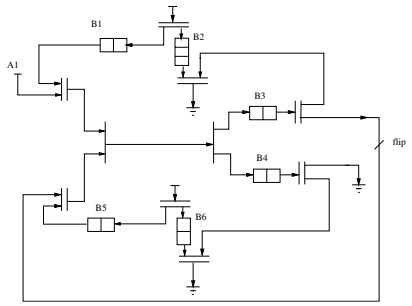

Fig. 6. A deadlocking virtual channel

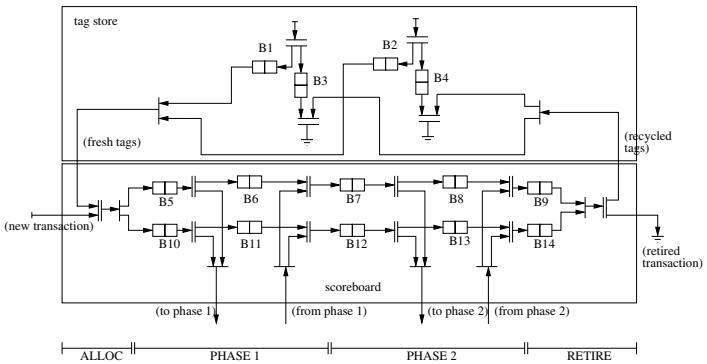

Fig. 7. A two entry scoreboard

ϕ_p. Interestingly, it does not satisfy the non-blocking property (Theorem 1). Once our framework detects that, it declares that satisfaction of ϕ_p cannot be concluded, and draws attention of the designer with a counter-example. It is a variation of the virtual channel shown in Figure 3, with a feedback joining the upper fork with the lower source. This feedback channel goes through a function (*flip*) that flips the bit of the flit that determines the output branch of the switch. Note that the upper credit buffer B_2 has three slots. A careful analysis would reveal that this particular buffer size for B_2 is the cause of the violation of ϕ_p. The same fabric with two slots in B_2 would work fine without any deadlock. Fabric in Figure 5 has a similar scenario too, *viz.* if capacity of B_5 were less than the sum of capacities of B_3 and B_4, it will violate ϕ_p. It is quite common to make such mistakes in the bit-level implementations and to run into deadlocks.

VII. HARDWARE SCOREBOARD

Figure 7 shows how a two-entry scoreboard may be modeled using the primitives of Section III. An incoming transaction on the left needs to obtain a tag before it can enter the scoreboard. Different tags are used to distinguish different in-flight transactions in the scoreboard. In this example, the scoreboard supports two simultaneous in-flight transactions and hence there are two tag sources. These tag sources are modeled using credit logic. Once the transaction enters the scoreboard it competes with the other transaction (if there is one) to enter the first phase of processing. The results of this phase may return out of order: tags are used to match a result with the corresponding transaction in the scoreboard. Once the result of the first phase is returned, the transaction moves on to the second phase. After the second phase is done, the transaction becomes eligible for retirement. When it wins arbitration, it retires and releases its tag which is then recycled for use by a future transaction. For details, see [5].

Theorem 1 and Lemmas 1 through 4 (with appropriate modifications) can again be defined on this structure due to the particular synchronization of credit logic. Theorem 1 is defined on the channel between the join and the switch at the entry of the scoreboard. Lemmas 1 through 4 need to include the role of the arbiter at the exit of the scoreboard. These are only cosmetic changes to the structure of the lemmas. We can again define a theorem analogous to Theorem 2 for the scoreboard, and the role of Theorem 1 and Lemmas 1 through 4 in its proof would remain the same even after the necessary cosmetic modifications.

VIII. EXPERIMENTAL RESULTS

We have developed Verilog implementations of the models of virtual channels and scoreboards. Our XMAS library implementation closely follows the logical specification provided in [5] which makes our benchmarks easily reproducible. We have used ABC [1] as our safety verification engine. Our experiments were performed on a laptop with 1.2 GHz Intel Celeron processor, and 2 GB RAM.

We present in Table III run-times (in sec.) of ABC on various models we considered. In all our experiments, ABC conjuncted all safety properties (Theorem 1, and Lemma 1 though 4) as a single proof obligation, and tried to prove it. We present two sets of experiments. In the first set, we have enabled the BUFFER RELATIONs as assumptions, and in the second set, these assumptions were disabled. From the experimental results, it is clear that BUFFER RELATIONs play a crucial role in verification. For the virtual channel examples, BUFFER RELATIONs made the proof obligation one-step inductive. For the scoreboard example, they sped up the proof. It will be interesting to investigate further

on invariants for scoreboards that would make their proof one-step inductive too; it is left as a future work. The fact that BUFFER RELATIONs are making proofs one-step inductive for many designs is the key factor that makes this approach scalable. The designs that we are considering are parameterizable, and serve as components of larger designs. In order to ensure scalability, ideally we need a proof technique whose complexity will be (almost) independent of the design size. One-step induction meets this goal. The BUFFER RELATIONs for other virtual channels and scoreboard that we used in our proofs are shown in Table II. In those relations, $num_{A_i}(B)$ represents the number of flits in buffer B that came from source A_i, for $i = 1, 2$. For the fabrics which are not discussed in [4], the relations have been derived manually by inspecting the designs.

model	BUFFER RELATIONs
buffered vc	$num(B_1) + num(B_3) + num_{A_1}(B_{ch}) = num(B_2)$ $num(B_5) + num(B_4) + num_{A_2}(B_{ch}) = num(B_6)$
ordered vc	$num(B_1) + num(B_3) = num(B_2)$ $num(B_7) + num(B_4) = num(B_8)$ $num(B_3) = num_{A_1}(B_5) + num(B_6)$ $num(B_4) = num_{A_2}(B_5)$ $num(B_5) = num_{A_1}(B_5) + num_{A_2}(B_5)$
scoreboard	$num(B_1) + \sum_{i=5}^{9} num(B_i) = num(B_3)$ $num(B_2) + \sum_{i=10}^{14} num(B_i) = num(B_4)$

TABLE II
ASSUMPTIONS USED IN THE PROOFS

Proof with Assumption					
model	PI	f/f	AND	One-step Induction	Other (PDR)
virtual channel (VC)	8	213	4383	0.65	-
buffered VC ($k = 2$)	10	383	8089	1.7	-
buffered VC ($k = 4$)	12	385	8288	1.78	-
ordered VC	13	424	9253	2.26	-
scoreboard	47	1293	18596	failed	38.80
Proof without Assumption					
model	PI	f/f	AND	One-step Induction	Other (PDR)
virtual channel	8	211	3449	failed	0.73
buffered VC ($k = 2$)	10	315	4579	failed	99.2
buffered VC ($k = 4$)	12	317	4652	failed	68.81
ordered VC	13	350	5236	failed	26.94
scoreboard	47	1289	14942	failed	57.25

TABLE III
EXPERIMENT ON VARIOUS COMMUNICATION FABRICS

In Table III, few things may be noted. The number of flip-flops and AND gates in the circuit increases once the BUFFER RELATIONs are put as assumptions. But from the run-time of ABC, it is evident that this increase in logic does not hamper the proof, rather facilitates it. In all cases when induction failed, interpolation with default resource limits in ABC failed as well. But in all cases, property directed reachability (PDR) [3] succeeded. For the cases where one-step induction succeeded, we ignored run-time for PDR. For all cases where assumptions were supplied, ABC independently proved the assumptions within the time shown in the table. In Figure 4, size k of the channel buffer B_{ch} is parameterizable. We are providing run-times for $k = 2$ and 4.

IX. CONCLUSION AND FUTURE WORK

We have presented a way of verifying progress property of credit-based flow-control networks by deriving and proving intermediate safety properties. The intermediate safety properties were derived by leveraging the high-level structure of the network. A crucial set of invariants, called BUFFER RELATIONs, were derived. These relations made the proof one-step inductive for many examples, and sped up the proof for others. We believe this work opens up new opportunities of research in formal verification of communication fabrics. The current work-flow of our framework is to prove Theorem 1 and Lemma 1-4 automatically using a safety verification engine and then take refuge to Theorem 2 to conclude the satisfaction of ϕ_p. This leads to the question whether we can automate the proof of Theorem 2 as well; it is left as a future work. Also, necessity and adequacy issues of Lemmas 1 through 4 are yet to be studied. One important question that arises naturally is whether these safety properties can be mined automatically and whether they are effective for proving other kind of progress properties. Finally, it needs to be studied whether the approach that we have presented could be utilized to prove progress property of a chip-wide communication network. Compositional reasoning, along with our approach, may proved to be useful in this context as credit-based flow-control is used to ensure smooth flow in each component of such big networks.

X. ACKNOWLEDGEMENT

We acknowledge Satrajit Chatterjee, Alexander Gotmanov, Mike Kishinevsky, and Alan Mishchenko for many helpful discussions, Jiang Long for his help on bit-blasting our models, and our anonymous reviewers for their suggestions of improvements over the initial draft of the paper. This work has been supported in part by SRC, NSA, and our industrial sponsors: Actel, Altera, Atrenta, Cadence, Calypto, IBM, Intel, Jasper, Magma, Oasys, Real Intent, Synopsys, Tabula, and Verific.

REFERENCES

[1] Abc verification system: http://www.eecs.berkeley.edu/ alanmi/abc/.
[2] Kamel Barkaoui, Claude Dutheillet, and Serge Haddad. An efficient algorithm for finding structural deadlocks in colored petri nets. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, pages 69–88, London, UK, 1993. Springer-Verlag.
[3] A. Bradley. Sat-based model checking without unrolling. In *VMCAI*. Springer Verlag, 2011.
[4] Satrajit Chatterjee and Michael Kishinevsky. Automatic generation of inductive invariants from high-level microarchitectural models of communication fabrics. In *CAV*. Springer Verlag, 2010.
[5] Satrajit Chatterjee, Mike Kishinevsky, and Umit Ogras. Quick formal modeling of communication fabrics to enable verification. In *HLDVT*, pages 42–49, 2010.
[6] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
[7] William J. Dally. Interconnection networks for high-performance parallel computers. chapter Virtual-channel flow control, pages 493–504. IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.
[8] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
[9] A. Gotmanov, S. Chatterjee, and M. Kishinevsky. Verifying deadlock-freedom of communication fabrics. In *VMCAI*. Springer Verlag, 2011.
[10] Viktor Schuppan and Armin Biere. Efficient reduction of finite state model checking to reachability analysis. *Int. J. Softw. Tools Technol. Transf.*, 5(2):185–204, 2004.
[11] F. Verbeek and J. Schmaltz. Formal specification of networks-on-chips: deadlock and evacuation. In *DATE*, 2010.