Formal Methods in Computer-Aided Design 2020

Effective System Level Liveness Verification

Alexander Fedotov*, Jeroen J.A. Keiren*, Julien Schmaltz†
*Eindhoven University of Technology
Eindhoven, The Netherlands
{a.fedotov, j.j.a.keiren}@tue.nl
†ICT Group
Eindhoven, The Netherlands
julien.schmaltz@ict.nl

***Abstract*—The language xMAS has been designed by Intel with the purpose of modelling and verification of hardware. Recently, the language was extended with finite state machines to make it more expressive [19]. Furthermore, it was shown how to prove liveness of such extended xMAS networks [19]. Unfortunately, we demonstrate that the proof technique is unsound. We provide an alternative approach which we have carefully proven to be correct. Moreover, we show that our approach scales very well, which makes it possible to prove liveness properties at the system level. In particular, we show that using our approach, it is possible to verify a power control architecture composed of 1299 state machines representing 50 power domains where each domain contains 5 master and 5 slave devices. Proving liveness of this system takes less than 10 minutes.**

***Index Terms*—Formal verification, liveness, communication networks, finite state machines**

I. Introduction

Formal verification has been successfully introduced in many design flows of hardware and software systems. More and more often, the sign-off decision for hardware blocks is taken solely on the results of formal proofs, the so-called *formal sign-off*. However, scaling formal verification to the system level remains a challenge.

Originally proposed by researchers at Intel, the xMAS language [7] and associated techniques for invariant generation [6], property checking [6], and deadlock hunting [12, 16][1] have been developed to address this challenge. These techniques are very efficient and were extended to performance validation [15], asynchronous circuits [4], progress verification [8], generalized to language families [17], and directly related to the Register Transfer Level [11, 13, 14].

Regarding liveness analysis, the key contribution of Gotmanov *et al.* [12] is to encode the existence of a deadlock state as a satisfiability problem. This technique is sound and can prove the absence of deadlock states. It is incomplete because a satisfiable solution does not necessarily represent a reachable state of the xMAS network. Checking reachability of potential xMAS deadlock states is efficient [20].

Verbeek *et al.* introduced state machines into xMAS together with extensions of liveness analysis techniques [19]. Their extension enables the modeling and analysis of complex cooperating state machines under the constraints imposed by micro-architectural choices. They demonstrated the verification of large systems consisting of nodes running cache coherence protocols and communicating via a Network-on-Chip. Inspired by Gotmanov *et al.*, Verbeek *et al.* encode liveness verification of xMAS extended with (finite) state machines to satisfiability. As we will show in this paper, their method is *unsound*.

We present a counter-example that is composed of a network with a deadlock that is not found by the technique of Verbeek *et al.* [19]. Subsequently, we propose an alternative encoding of liveness into a satisfiability problem. We carefully prove that if an xMAS network has a path to a state with a deadlock, there exists a satisfying assignment to the satisfiability problem we generate, i.e., our encoding is sound. Finally, we introduce two sets of benchmarks including a simplified power control architecture inspired by industrial case-studies.

The benchmarks and our implementation are publicly available [2]. A network with 1299 state machines can be proven live in less than 10 minutes.

We introduce xMAS, liveness of channels and idle and block equations in Section II. In Section III we introduce xMAS finite state machines, and show why the approach from [19] is unsound. Our approach using idle and block equations is described in Section IV. Our implementation is evaluated in Section V. We conclude in Section VI.

II. Preliminaries

A. xMAS syntax

xMAS [7] is a graphical language aimed at modeling and verifying communication fabrics. An xMAS network comprises a set of primitives connected by typed channels. The progress of messages between primitives is controlled by a simple handshake protocol. Each channel consists of three signals, one for data and two boolean control signals, $\mathbf{irdy}$ and $\mathbf{trdy}$. Consider the transfer of data from primitive A to primitive B via channel x. When primitive A is ready to transfer datum d through channel x, it sets $x.\mathbf{data}$ to d, and $x.\mathbf{irdy}$ to true, indicating the *initiator* is ready to transfer data. Whenever B is ready to accept data, it sets $x.\mathbf{trdy}$ to true, indicating the *target* is ready to receive. The data transfer happens if and only if $x.\mathbf{irdy} \wedge x.\mathbf{trdy}$, i.e., the initiator is

[1]Note that in the literature related to liveness verification of xMAS networks, it is common to call states with non-live channels deadlock states. We adhere to this terminology, although it is different from the conventional notion of deadlock.

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_7

ready to send and the target is ready to receive. The core xMAS primitives are shown in Figure 1.

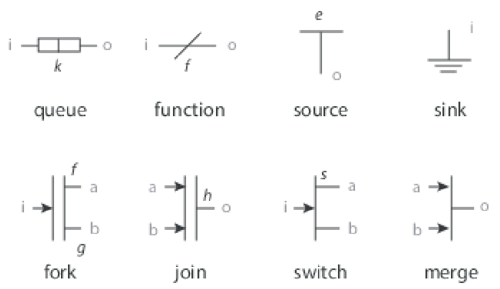


Figure 1: Core xMAS primitives

We provide detailed descriptions of the source, sink and queue primitives as they are used directly in this paper. For details of the other primitives the reader is referred to [7].

A source non-deterministically injects data into the network infinitely often. This is modelled using the unconstrained primary input **oracle**. Once a source decides to transfer datum d, it will keep trying until the transfer succeeds. This is modelled using the standard synchronous operator $\mathbf{pre}$ that returns the value of its argument in the previous clock cycle, and $false$ in the very first cycle. Formally, the source is described as follows:

$$o.\mathbf{irdy} := \mathbf{oracle} \vee \mathbf{pre}(o.\mathbf{irdy} \wedge \neg o.\mathbf{trdy})$$
$$o.\mathbf{data} := d.$$

A sink consumes data from the network infinitely often:

$$i.\mathbf{trdy} := \mathbf{oracle} \vee \mathbf{pre}(i.\mathbf{trdy} \wedge \neg i.\mathbf{irdy}).$$

A queue is a FIFO buffer with k places. A queue is ready to write data to the output when it is not empty. The data the queue is ready to write is the head of the queue. A queue is ready to accept data when it is not full. Formally,

$$o.\mathbf{irdy} := \neg\mathbf{is\_empty}, \qquad o.\mathbf{data} := \mathbf{head},$$
$$i.\mathbf{trdy} := \neg\mathbf{is\_full},$$

where i and o are the input and output channels of the queue respectively.

Example 1. Consider the simple xMAS network depicted in Figure 2. We use this network as a running example. The network consists of a source, a queue, and a sink. The source produces tokens t. The source controls the $x.\mathbf{irdy}$ and $x.\mathbf{data}$ signals of its output channel x. The queue controls the $x.\mathbf{trdy}$ signal of channel x, and $y.\mathbf{irdy}$ and $y.\mathbf{data}$ of its output channel y. The sink controls the $y.\mathbf{trdy}$ signal of channel y. The signals are defined as follows.

$$x.\mathbf{irdy} := \mathbf{oracle}_{src} \vee \mathbf{pre}(x.\mathbf{irdy} \wedge \neg x.\mathbf{trdy})$$
$$x.\mathbf{data} := t$$
$$x.\mathbf{trdy} := \neg\mathbf{is\_full}$$
$$y.\mathbf{irdy} := \neg\mathbf{is\_empty}$$
$$y.\mathbf{data} := \mathbf{head}$$
$$y.\mathbf{trdy} := \mathbf{oracle}_{snk} \vee \mathbf{pre}(y.\mathbf{trdy} \wedge \neg y.\mathbf{irdy}).$$

The semantics of an xMAS network consists of a combinatorial and a sequential phase. In the first, all **data**, **irdy** and **trdy** signals are updated. In the second all components with state update their state. The global state of an xMAS network is the product of the local states of all components. We write $\vec{s} \xrightarrow{X} \vec{s'}$ for the transition between global states $\vec{s}$ and $\vec{s'}$, where X is a set of (channel,data) pairs representing the simultaneous data transfers in the current clock cycle.

B. *Liveness of channels*

Liveness of channels is defined using linear temporal logic (LTL). LTL and its semantics are considered standard, and we refer to text books such as [3] for the details. To interpret LTL on xMAS networks, we first define paths and maximal paths in such networks. In the rest of this paper, we implicitly fix an xMAS network $N = (P, G)$, where P is the set of primitives, and G is the set of channels. Given a channel $x \in G$, by $C(x)$ we denote the set of all possible values of $x.\mathbf{data}$. By C we denote the set of all data of N, that is $C = \bigcup_{x \in G} C(x)$.

Definition 1. A *path* is a possibly infinite sequence of global states $\pi = \vec{s}_0, \vec{s}_1, \vec{s}_2, \ldots$ such that for all $j > 0$, $\vec{s}_{j-1} \xrightarrow{X} \vec{s}_j$ for some X. The set of paths starting in a state $\vec{s}$ is denoted using $\mathsf{Paths}(\vec{s})$, and for xMAS network N we write $\mathsf{Paths}(N)$ to denote $\mathsf{Paths}(\vec{s}_0)$, where $\vec{s}_0$ is the initial state of the network N. For finite paths $\pi = \vec{s}_0, \ldots, \vec{s}_n$ we define $\mathsf{last}(\pi) = \vec{s}_n$. A path π is called *maximal* if and only if it is infinite, or it is finite and $\mathsf{last}(\pi)$ has no outgoing transitions.

A channel is *live* whenever, always when its initiator is ready to transfer data, the transfer will eventually be successful.

Definition 2 ([12][2])**.** Channel $x \in G$ is *live* for $d \in C(x)$ iff

$$N \models \mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d) \implies \mathsf{F}(x.\mathbf{irdy} \wedge x.\mathbf{trdy} \wedge x.\mathbf{data} = d)).$$

We henceforth make the (standard) assumption that channels are (forward) persistent. This means that whenever the initiator is ready to send d along x, it will remain ready to do so until the transfer is successful. Formally, the network satisfies $\mathsf{G}((x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge \neg x.\mathbf{trdy}) \implies \mathsf{X}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d))$. Under this assumption, channel x is live if and only if it is live for all $d \in C(x)$.

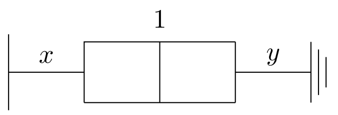


Figure 2: xMAS example

[2]Gotmanov et al. [12] use property $\mathsf{G}(x.\mathbf{irdy} \implies \mathsf{F}\ x.\mathbf{trdy})$, which does not guarantee that the transfer eventually succeeds if persistency is not assumed.

We recall the notions *idle* and *block* from [12]. A channel is *idle* for d if eventually the initiator will never send message d along that channel, and it is *blocked* if eventually the target will never be able to receive a message along that channel.

Definition 3 **([12]).** Let $x \in G$ and $d \in C(x)$. We define

$$\begin{aligned} \mathbf{idle}(x(d)) &:= \mathsf{FG}(\neg x.\mathbf{irdy} \vee x.\mathbf{data} \neq d) \\ \mathbf{block}(x) &:= \mathsf{FG}\neg x.\mathbf{trdy} \end{aligned}$$

A local deadlock is defined as a *dead* channel, where a channel is dead for value d if and only if it is not live for d. This means there exists a path in the xMAS network to a state that satisfies $\neg\mathbf{idle}(x(d)) \wedge \mathbf{block}(x)$. In other words, a channel is dead whenever its initiator is ready to transfer datum d and its target will never be ready to accept the data.

Definition 4. Let N be an xMAS network, with x a forward persistent channel in N, and $d \in C(x)$. We define

$$\begin{aligned} \mathbf{live}(x(d)) &:= \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x) \\ \mathbf{dead}(x(d)) &:= \neg\mathbf{live}(x(d)) \\ \mathbf{live}(x) &:= \bigwedge_{d \in C(x)} \mathbf{live}(x(d)) \\ \mathbf{dead}(x) &:= \bigvee_{d \in C(x)} \mathbf{dead}(x(d)) \end{aligned}$$

Persistency now allows us to simplify the definition of liveness using the following theorem adapted from [12].

Theorem 1. *For all channels $x \in G$ and $d \in C(x)$, let*

$$\begin{aligned} \mathbf{live}(x(d)) &:= \mathbf{idle}(x(d)) \vee \neg\mathbf{block}(x) \\ \mathbf{dead}(x(d)) &:= \neg\mathbf{live}(x(d)). \end{aligned}$$

Then, for all persistent channels $x \in G$ and $d \in C(x)$,

1) *x is live for d iff $N \models \mathbf{live}(x(d))$, and*
2) *x is* dead *for d iff $\exists \pi \in \mathsf{Paths}(N).\pi \models \mathbf{dead}(x(d))$.*

Note that the formula for *live* channels is evaluated over the *network* (i.e., over *all* paths), and the formula for *dead* channels is evaluated over a *path* due to the LTL semantics.

In Definition 3, we only defined $\mathbf{block}(x)$. We can refine this definition by introducing $\mathbf{block}(x(d))$ as follows.

$$\mathbf{block}(x(d)) := \mathsf{FG}(\neg x.\mathbf{trdy} \vee x.\mathbf{data} \neq d)$$

It is easy to see that $\mathbf{block}(x)$ implies $\mathbf{block}(x(d))$ for any $d \in C(x)$. The following then follows immediately.

Lemma 1. *For all persistent channels $x \in G$, $d \in C(x)$, and all paths $\pi \in \mathsf{Paths}(N)$, $\pi \models \mathbf{dead}(x(d))$ implies $\pi \models \bigwedge_{e \in C(x)} \mathbf{block}(x(e))$.*

C. Idle and block equations

The main contribution of Gotmanov *et al.* [12] is to express deadlock conditions for each primitive using equations over boolean variables. If these *idle and block equations* are satisfiable, a (possible) deadlock has been detected; otherwise, the network is guaranteed to be deadlock free. The method is sound but incomplete: if the equations are satisfiable, the assignment to the boolean variables may constitute an unreachable deadlock state. This is alleviated to some extent by using invariants to approximate the reachable states.

The boolean variables express the conditions under which a primitive will eventually never try to output value d, denoted using variable $\mathbf{idle}_x^d$, or eventually never try to read from channel x, denoted using variable $\mathbf{block}_x$. The encoding essentially approximates the LTL specifications of idle and block defined before. In particular, if there exists a path π in the xMAS network such that $\pi \models \mathbf{dead}(x(d))$, then there is a satisfying assignment to the variables in the idle and block equations in which $\mathbf{idle}_x^d$ is *false*, and $\mathbf{block}_x$ is *true*.

Example 2. Recall the network from Example 1. Sources are never idle, and sinks are never blocked. The input channel of the queue, x, is blocked when the queue is full and its output channel y is blocked. The output channel of the queue is idle when the queue is empty and its incoming channel x is idle. This results in the following equations.

$$\begin{aligned} \mathbf{idle}_x &\equiv \bot & \mathbf{block}_x &\equiv \mathbf{full} \wedge \mathbf{block}_y \\ \mathbf{idle}_y &\equiv \mathbf{empty} \wedge \mathbf{idle}_x & \mathbf{block}_y &\equiv \bot \\ \mathbf{dead}_x &\equiv \neg\mathbf{idle}_x \vee \mathbf{block}_x & \mathbf{dead}_y &\equiv \neg\mathbf{idle}_y \vee \mathbf{block}_y \end{aligned}$$

We can conclude that neither x nor y is dead.

III. Life and death of state machines in xMAS

A. xMAS finite state machines

Verbeek *et al.* describe an extension of xMAS with finite state machines for the integrated verification of, for instance, cache coherence protocols together with their underlying communication fabric [18, 19]. The xMAS automata allow for the symbolic description of the channels and data read and written along transitions. However, every transition reads and writes (at most) one channel. In this paper we require explicit definition of every datum read/written on a transition to simplify the presentation. The results could equally be expressed using symbolic notation as in [18, 19]. However, since the number of channels and the data transferred are typically assumed to be finite, they can be expanded in the FSM, and this change does not alter the expressive power.

Definition 5. A *finite state machine (FSM)* is a tuple (S, s_0, I, O, T), where S is a finite set of states; $s_0 \in S$ is an initial state; I is a finite set of input channels; O is a finite set of output channels; and $T \subseteq S \times (I \times C) \times (O \times C) \times S$ is the total transition relation.

Since T is total, every state has at least one outgoing transition. We use names $s, s', s_1, \ldots$ for states. We write $s \xrightarrow{x(d)/y(e)} s'$ for $(s, (x, d), (y, e), s') \in T$. We sometimes write $?x(d)$ and $!y(e)$ to stress d is read from channel x, and e is written to y. For state $s \in S$, $in(s)$ and $out(s)$ denote the sets of incoming and outgoing transitions of s, respectively. Likewise, for channels $x \in (I \cup O)$, and data $d \in C(x)$, $read(x, d)$ and $write(x, d)$ represent the sets of transitions reading d from x and writing d to x, respectively.

Note that the requirement that every transition reads from and writes to exactly one channel is not fundamental. Transitions $t = s \xrightarrow{!y(e)} s'$ that do not read from an input channel can be modeled by introducing a new channel x_t that is connected to a source and the FSM, and be replaced by $s \xrightarrow{?x_t/!y(e)} s'$. Transitions that do not write to an output channel and transitions that do not read or write any channel can be modeled in a similar way.

In an FSM, exactly one state is current at a time, this state is denoted $cur(s)$. A transition $s \xrightarrow{x(d)/y(e)} s'$ is enabled if and only if s is the current state, the input channel x is ready to send d, and the output channel y is ready to receive. Note that whether the input and output channels are ready depends on the environment of the FSM.

Definition 6. Given FSM (S, s_0, I, O, T), transition $s \xrightarrow{x(d)/y(e)} s' \in T$ is enabled, denoted $enabled(s \xrightarrow{x(d)/y(e)} s')$ iff $cur(s) \wedge x.\mathbf{irdy} \wedge x.\mathbf{data} = d \wedge y.\mathbf{trdy}$.

In any state, there can be multiple enabled transitions. To resolve this non-determinism, a scheduler sel is introduced that, at every clock cycle, selects an enabled transition. If transition t is selected, this is denoted $sel = t$.

The FSM needs to indicate to its environment whether it is ready to send along an outgoing channel, or to read along an incoming channel. This is defined in terms of **irdy**, **trdy** and **data** as follows.

Definition 7. Given FSM (S, s_0, I, O, T), for $x \in I, y \in O$:

$$x.\mathbf{trdy} := \exists s \xrightarrow{x(d)/y(e)} s' \in T.sel = s \xrightarrow{x(d)/y(e)} s'$$

$$y.\mathbf{irdy} := \exists s \xrightarrow{x(d)/y(e)} s' \in T.sel = s \xrightarrow{x(d)/y(e)} s'$$

$$y.\mathbf{data} := \begin{cases} e & \text{if } \exists s \xrightarrow{x(d)/y(e)} s' \in T.sel = s \xrightarrow{x(d)/y(e)} s' \\ \bot & \text{otherwise} \end{cases}$$

Since the scheduler non-deterministically chooses between enabled transitions, and **irdy** is only set for the output channel of a selected transition, whenever **irdy** is set for an output channel of an FSM, the target of that channel is ready to receive, i.e., **trdy** is set. Non-determinism of the scheduler could lead to an enabled transition being ignored for an infinite amount of time. However, we assume scheduler sel to be fair, i.e., if state s is visited infinitely often with $s \xrightarrow{x(d)/y(e)} s'$ enabled, then $s \xrightarrow{x(d)/y(e)} s'$ will be selected infinitely often. We therefore only verify liveness of the xMAS network along fair paths. Such paths are defined as follows.

Definition 8. Given a path π, we say that π is fair if and only if for all FSM primitives $M = (S^M, s_0^m, I^M, O^M, T^M)$ and local transitions $t \in T^M$, we have $\pi \models (\mathsf{GF} enabled(t)) \implies (\mathsf{GF} sel = t)$

B. Idle and block equations by Verbeek et al.

Verbeek *et al.* define a SAT encoding using idle and block equations for xMAS automata as follows. Given $M = (S, s_0, I, O, T)$, for $s \xrightarrow{x(d)/y(e)} s' \in T$, $x \in I$, $y \in O$, $d \in C(x)$, $e \in C(y)$, they define the following.

$$\begin{aligned} \mathbf{dead}_{s \xrightarrow{x(d)/y(e)} s'} &\equiv \mathbf{idle}_x^d \vee \mathbf{block}_y \\ \mathbf{dead}_s &\equiv cur_s \wedge \bigwedge_{t \in out(s)} \mathbf{dead}_t \\ \mathbf{dead}_M &\equiv \bigvee_{s \in S} \mathbf{dead}_s \\ \mathbf{block}_x^d &\equiv \mathbf{dead}_M \vee (read(x,d) = \emptyset) \\ \mathbf{idle}_y^e &\equiv \mathbf{dead}_M \vee (write(y,e) = \emptyset) \end{aligned}$$

Here, cur_s are boolean variables, aimed at reflecting the current state of the FSM.

Intuitively, Verbeek *et al.* propose to encode that input (output) channels of an FSM are blocked (idle) as follows. An input channel x is blocked for d if either the FSM has no transition which reads d from x or the FSM is dead. Likewise, an output channel y is idle for e if either the FSM has no transition which writes e to y or the FSM is dead. With the notion of dead FSM, Verbeek *et al.* intend to encode the existence of a state (a dead state, using the terminology of the authors), which can eventually be reached, and at the same time cannot be left anymore, since all outgoing transitions are dead. In such a situation, the FSM can neither read from its inputs nor write to its outputs.

C. Life and death of state machines: a counter-example

Unfortunately, there are xMAS networks with FSMs that are deadlock free according to these idle and block equations that do contain a deadlock. This is illustrated by the following example.

Example 3. Consider the state machine, depicted in Figure 3. It has two input channels x and y, connected to sources, and two output channels o and z, connected to sinks. All channels only transfer datum d. Initially, in s_0, the FSM can either read d from channel x and produce d on channel o, and stay in s_0, or it can read d from y *once* and produce d on z, and go to s_1. In s_1, the FSM never reads from y nor writes to o, and only reads from x, writes to z, and stays in s_1.

According to the definition by Verbeek *et al.*, the FSM is not *dead*: channels o and z are not blocked, and since channel x is not idle, neither of the self-loops is dead. Consequently, neither s_0 nor s_1 is dead, and the FSM is not dead. However, once s_1 is reached, messages waiting on channel y will never be read, so y should be blocked.

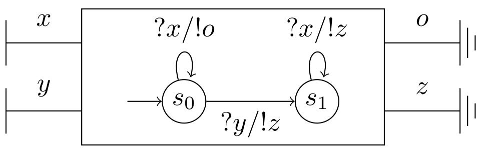


Figure 3: Counterexample to method by Verbeek et al.

The example shows that, although channel y is dead for d, since $\mathbf{block}_y^d$ is false, this is not detected using the idle and block equations. The encoding from [19] is therefore unsound.

Generally, the issue lies in the definition of $\mathbf{dead}_M$. Even when none of the input channels are idle, and no output channel is blocked, a state machine can block an input channel. This happens, *e.g.*, when source states of transitions reading from a particular channel are reached only *finitely many times*. Output channels can become idle for similar reasons.

IV. Idle and block equations for xMAS FSMs

We propose alternative idle and block equations for FSMs in the spirit of [12]. An input channel x of an FSM is dead, when eventually all transitions reading x become disabled. There are two possible causes for this. First, the source state of the transition can eventually never be reached anymore. Second, whenever the source state of the transition is current, the environment disables the transition since the output channel is blocked. We capture this intuition by saying that states that are eventually never reached again are idle, and transitions that are eventually never enabled are dead.

Definition 9 (Idle states and dead transitions)**.** Consider FSM (S, s_0, I, O, T). For $s \in S$ and $t \in T$ we define the following.

$$\mathbf{idle}(s) := \mathsf{FG}\neg \mathit{cur}(s)$$
$$\mathbf{dead}(t) := \mathsf{FG}\neg \mathit{enabled}(t)$$

Formally, transitions eventually never become enabled along a path if and only if either the source state or the input channel of the transition is idle, or the output channel is blocked.

Lemma 2. *Let $M = (S, s_0, I, O, T)$ be an FSM in N. For all $t = s \xrightarrow{x(d)/y(e)} s' \in T$, global states $\vec{s}$, and paths $\pi \in$ $\mathsf{Paths}(\vec{s})$,*

$$\pi \models \mathsf{FG}\neg \mathit{enabled}(t) \text{ iff } \pi \models \mathbf{idle}(s) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y).$$

Proof sketch (for the full proof see [10]). Fix an arbitrary transition $t = s \xrightarrow{x(d)/y(e)} s'$, global state $\vec{s}$, and path $\pi \in \mathsf{Paths}(\vec{s})$. We prove both directions separately.

- $\Rightarrow$ Assume that $\pi \models \mathsf{FG}\neg \mathit{enabled}(t)$. Towards a contradiction, suppose $\pi \not\models \mathbf{idle}(s) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y(e))$. We know that $\neg\mathbf{idle}(s) \equiv \mathsf{GF}\mathit{cur}(s)$, $\neg\mathbf{idle}(x(d)) \equiv \mathsf{GF}(x.\mathbf{irdy} \wedge x.\mathbf{data} = d)$, $\neg\mathbf{block}(y) \equiv \mathsf{GF}y.\mathbf{trdy}$. From this, using the semantics of LTL formulas we derive that $\pi \models \mathsf{GF}\mathit{enabled}(t)$, which is a contradiction.
- $\Leftarrow$ Suppose $\pi \models \mathbf{idle}(s) \vee \mathbf{idle}(x(d)) \vee \mathbf{block}(y)$. We split the three cases and use Definitions 3, 6, and 8 to show that $\pi \models \mathsf{FG}\neg \mathit{enabled}(t)$ in each of these cases. $\square$

Due to the way the semantics of FSMs resolve non-determinism, output channels of an FSM are never dead.

Lemma 3. *Given FSM (S, s_0, I, O, T) in N, for all global states $\vec{s}$ and for channels $y \in O$ and $e \in C(y)$, we have for all paths $\pi \in \mathsf{Paths}(\vec{s})$, $\pi \not\models \mathbf{dead}(y(e))$.*

We now specify the idle and block equations for FSMs used in a SAT encoding. The equations refer to variables **idle** of incoming channels and **block** of outgoing channels that are defined in other components.

Definition 10 (Idle and block equations for FSMs)**.** Consider an FSM $M = (S, s_0, I, O, T)$. For $s \in S$, $x \in I$, $y \in O$, $d \in C(x)$, $e \in C(y)$, and $s \xrightarrow{x(d)/y(e)} s' \in T$ we define the following equations.

$$\mathbf{dead}_{s\xrightarrow{x(d)/y(e)}s'} \equiv \mathbf{idle}_s \vee \mathbf{idle}_x^d \vee \mathbf{block}_y$$
$$\mathbf{idle}_s \equiv \neg \mathit{cur}_s \wedge \bigwedge_{t \in \mathit{in}(s)} \mathbf{dead}_t$$
$$\mathbf{block}_x^d \equiv \bigwedge_{t \in \mathit{read}(x,d)} \mathbf{dead}_t$$
$$\mathbf{block}_x \equiv \bigwedge_{d \in C(x)} \mathbf{block}_x^d$$
$$\mathbf{idle}_y^e \equiv \bigwedge_{t \in \mathit{write}(y,e)} \mathbf{dead}_t$$
$$\mathbf{idle}_y \equiv \bigwedge_{e \in C(y)} \mathbf{idle}_y^e$$

$\mathsf{SAT}(M)$ consists of the conjunction of all the idle and block equations for all states, transitions and channels in FSM M. Similarly we write $\mathsf{SAT}(N)$ for network N, which is the conjunction of all formulas for all the primitives of N, where for non-FSM components, the encoding from [12] is used.

We additionally use the invariants from [19] to reduce the number of false deadlocks. For example, $\sum_{s \in S} \mathit{cur}_s = 1$ dictates that the FSM is always in exactly one state.

The intuition behind the encoding is as follows. If a state is not current, and eventually none of its incoming transitions can ever become enabled, the state is effectively unreachable, thus the state is idle. In turn, a transition is dead if it ultimately never becomes enabled. This is the case if either its source state or its incoming channel is idle, or its outgoing channel is blocked. An input channel is blocked for a given datum if no transition will read that datum from the channel. Likewise an output channel is idle for a datum if that datum is never written to it. An output channel is idle if it is idle for all values, meaning that no value will ever be written to it. An input channel is blocked if it is blocked for all values. This follows from Lemma 1: a dead channel is blocked for all data.

We say assignment σ is *consistent* with path π and a set of components if for all input channels x, output channels y and data e of these components, $\sigma(\mathbf{block}_x) = \top$ iff $\pi \models \mathbf{block}(x)$ and $\sigma(\mathbf{idle}_y^e) = \top$ iff $\pi \models \mathbf{idle}(y(e))$.

We finally prove our idle and block equations are sound: if there is a channel that is dead for a particular value, then there is a satisfying assignment to the boolean equations showing this. We only consider input channels of FSMs, since output channels of FSMs cannot be dead as shown in Lemma 3.

Recall that a maximal path can either be finite or infinite, and in an infinite path in an xMAS network, the FSM can be stuck in a state locally. We construct assignments for each of these cases, and prove that each of the assignments is a satisfying assignment. We first construct assignment σ_s for the case where a (fair) maximal path in a network containing the FSM is such that the FSM is stuck locally in state s.

Definition 11. Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N, $\pi \in \mathsf{Paths}(M)$ and $s \in S$. Assignment σ_s is defined as follows, where we write $v := w$ if σ_s assigns w to v. For states $s' \in S$, transitions $t \in T$, channels $x \in I$, $y \in O$, and $d \in C(x)$, $e \in C(y)$, let:

$$\begin{array}{ccc} cur_{s'} := s = s' & \mathbf{idle}_{s'} := s \neq s' & \mathbf{dead}_t := \top \\ \mathbf{block}^d_x := \top & \mathbf{block}_x := \top & \mathbf{block}_y := \top \\ \mathbf{idle}^d_y := \top & \mathbf{idle}_y := \top & \mathbf{idle}^d_x := \bot \end{array}$$

and σ_s is consistent with π for all components other than M.

Whenever the FSM is stuck locally, σ_s gives a satisfying assignment for the encoding to SAT.

Lemma 4. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N and $s \in S$. If $\pi \in \mathsf{Paths}(N)$ is a* fair maximal path *such that either*

- *π is finite and $\mathsf{last}(\pi) \models cur(s)$, or*
- *$\pi \models \mathsf{FG}\left(cur(s) \wedge \bigwedge_{t \in out(s)} \neg enabled(t)\right)$*

then σ_s is a satisfying assignment for $\mathsf{SAT}(N)$.

The proof of this lemma and Lemma 5 are omitted due to space restrictions. Details can be found in [10].

For a fair maximal path π on which the FSM is not stuck locally, we construct a satisfying assignment σ_π based on π.

Definition 12. Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N and $\pi \in \mathsf{Paths}(N)$. Assignment σ_π is defined as follows. For states $s' \in S$, transitions $t \in T$, channels $x \in I$, $y \in O$, and $d \in C(x)$, $e \in C(y)$, let:

$$\begin{aligned} cur_{s'} &:= s = s' \\ \mathbf{idle}_{s'} &:= \forall 0 \le k \le n. \pi[i+k] \models \neg cur(s') \\ \mathbf{dead}_t &:= \forall 0 \le k \le n. \pi[i+k] \models \neg enabled(t) \\ \mathbf{block}^d_x &:= \forall t \in read(x,d). \forall 0 \le k \le n. \\ &\qquad \pi[i+k] \models \neg enabled(t) \\ \mathbf{block}_x &:= \forall d \in C(x). \forall t \in read(x,d). \forall 0 \le k \le n. \\ &\qquad \pi[i+k] \models \neg enabled(t) \\ \mathbf{idle}^d_y &:= \forall t \in write(y,e). \forall 0 \le k \le n. \\ &\qquad \pi[i+k] \models \neg enabled(t) \\ \mathbf{idle}_y &:= \forall e \in C(y). \forall t \in write(y,e). \forall 0 \le k \le n. \\ &\qquad \pi[i+k] \models \neg enabled(t) \end{aligned}$$

and σ_π is consistent with π for all components other than M.

Whenever the FSM is not stuck locally, σ_π gives a satisfying assignment for the encoding to SAT.

Lemma 5. *Let $M = (S, s_0, I, O, T)$ be an FSM that appears in an xMAS network N. If $\pi \in \mathsf{Paths}(N)$ is an* infinite fair maximal paths *such that for all $s \in S$, $\pi \models \mathsf{GF}\left(cur(s) \implies \bigvee_{t \in out(s)} enabled(t)\right)$, the assignment σ_π is a satisfying assignment for $\mathsf{SAT}(N)$.*

We finally prove soundness of our encoding, assuming that idle and block equations for non-FSM components are sound.

Theorem 2. *Let $M = (S, s_0, I, O, T)$ be an FSM in xMAS network N. For all channels $x \in I$ and data $d \in C(x)$, if there exists a fair maximal path $\pi \in \mathsf{Paths}(N)$ such that $\pi \models \mathbf{dead}(x(d))$, then $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$ is satisfiable.*

Proof. Fix arbitrary channel $x \in I$, and datum $d \in C(x)$, and let $\pi \in \mathsf{Paths}(N)$ be a fair maximal path such that $\pi \models \mathbf{dead}(x(d))$. We distinguish three cases:

- π is finite. Let $\mathsf{last}(\pi) \models cur(s)$ for some $s \in S$. According to Lemma 4, σ_s is a satisfying assignment for $\mathsf{SAT}(N)$. Note that $\mathbf{block}_x = \top$ and since σ_s is consistent with π for non-FSM components, $\mathbf{idle}^d_x = \bot$. So σ_s is a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$.
- π is infinite and $\pi \models \mathsf{FG}(cur(s) \wedge \bigwedge_{t \in out(s)} \neg enabled(t))$ for some $s \in S$. Let s be such. According to Lemma 4, σ_s is consistent with $\mathsf{SAT}(N)$. Using similar reasoning as in the previous case, we can conclude that σ_s is a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$.
- π is infinite and for all $s \in S$, we have $\pi \not\models \mathsf{FG}(cur(s) \wedge \bigwedge_{t \in out(s)} \neg enabled(t))$, i.e., $\pi \models \mathsf{GF}(cur(s) \implies \bigvee_{t \in out(s)} enabled(t))$.
  According to Lemma 5, σ_π is consistent with $\mathsf{SAT}(N)$. Note that since $\pi \models \mathbf{dead}(x(d))$, $\pi \models \mathbf{block}(x(e))$ for all $e \in C(x)$, according to Lemma 1. Consider arbitrary $e \in C(x)$, we show that the assignment satisfies $\mathbf{block}^e_x$. From this and the definition it immediately follows that it satisfies $\mathbf{block}_x$.
  Let i be the index that signals the start of the loop of the lasso on π. Since $\pi \models \mathbf{block}(x(e))$, $\pi \models \mathsf{FG}(\neg x.\mathbf{trdy} \vee x.\mathbf{data} \neq e)$. By definition of *enabled*, this implies $\pi \models \mathsf{FG}(\neg enabled(t))$ for all $t \in read(x, e)$. Hence, for all $0 \le k \le n$, $\pi[i+k] \models \neg enabled(t)$ for all $t \in read(x, e)$. By definition of σ_π, we then have $\mathbf{block}^e_x = \top$. Since this holds for all e, by definition also $\mathbf{block}_x = \top$, and σ_π is a satisfying assignment for $\mathsf{SAT}(N) \wedge \neg\mathbf{idle}^d_x \wedge \mathbf{block}_x$. □

We illustrate our approach using an example.

Example 4. Recall the FSM from Example 3. The environment guarantees $\mathbf{idle}_x = \mathbf{idle}_y = \mathbf{block}_o = \mathbf{block}_z = \bot$. The idle and block equations are as follows.

$$\begin{aligned} \mathbf{idle}_{s_0} &\equiv \neg cur_{s_0} \wedge \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} \\ \mathbf{idle}_{s_1} &\equiv \neg cur_{s_1} \wedge \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} \wedge \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} \\ \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} &\equiv \mathbf{idle}_{s_0} \vee \mathbf{idle}_x \vee \mathbf{block}_o \\ \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} &\equiv \mathbf{idle}_{s_0} \vee \mathbf{idle}_y \vee \mathbf{block}_z \\ \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} &\equiv \mathbf{idle}_{s_1} \vee \mathbf{idle}_x \vee \mathbf{block}_z \\ \mathbf{block}_x &\equiv \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} \wedge \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} \\ \mathbf{block}_y &\equiv \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} \\ \mathbf{idle}_o &\equiv \mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} \\ \mathbf{idle}_z &\equiv \mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} \end{aligned}$$

We correctly detect that y is dead. This is witnessed by the following satisfying assignment for these equations, that also satisfies $\mathbf{block}_y = \top$, and thus $\neg\mathbf{idle}_y \wedge \mathbf{block}_y$.

$$\begin{aligned}
cur_{s_0} &:= \bot & cur_{s_1} &:= \top \\
\mathbf{idle}_{s_0} &:= \top & \mathbf{idle}_{s_1} &:= \bot \\
\mathbf{dead}_{s_0 \xrightarrow{x/o} s_1} &:= \top & \mathbf{dead}_{s_0 \xrightarrow{y/z} s_1} &:= \top \\
\mathbf{dead}_{s_1 \xrightarrow{x/z} s_1} &:= \bot & & \\
\mathbf{block}_x &:= \bot & \mathbf{block}_y &:= \top \\
\mathbf{idle}_o &:= \top & \mathbf{idle}_z &:= \bot
\end{aligned}$$

V. Experiments

We have implemented the idle and block equations described in Section IV. Given an xMAS model as input, our tool automatically generates a SAT problem that incorporates the idle and block equations [2]. The SAT problem is solved using Z3 [9] to verify liveness. Our tool can also generate an SMV model that encodes the xMAS network and its behaviour. This model uses idle and block equations as invariants. Reachability of a state in which a channel of the given xMAS model is checked using the nuXmv model-checker [5].

A. Experimental setup

We evaluate our implementation using two sets of models: one inspired by "go/no go" testing, the other inspired by power domains architectures. All models also have a version in which deadlocks have been introduced. A detailed description of the models can be found in [1].

Go/no go models are balanced binary trees of go/no go blocks. Each block has two binary inputs and one binary output, and consists of a pair of interconnected FSMs. The output of the block is ok if the data consumed from both input channels are ok, and it is nok otherwise.

Models of n levels of go/no go blocks (each block consists of two FSMs) are constructed by composing $2^n - 1$ go/no go blocks as a balanced binary tree. The output channels of two adjacent blocks on one level are used as input channels of a block on the next level in the tree.

Go/no go models with deadlocks are obtained by modifying one FSM in a go/no go block whose inputs are not connected to another block as follows. We add a new state with a self-loop reading ok from the first input channel i. We add a transition that reads nok from i from the initial state of the FSM to this new state. The new state is reachable, channel i is blocked for nok, and all output channels are idle.

Power domains are used to improve power efficiency of systems on chip. A power control architecture turns power domains on and off depending on the needs of an application. We model a dynamic power management policy that is an abstraction of industrial practice.

Our models consist of a number of power domains, each of which has a domain power controller. If the model has multiple power domains it also has a top power controller. Every power domain contains a number of device-controller pairs. In Figure 4a, we depict a device controller, which turns on its device (depicted in Figure 4b) if the device indicates activity (generated by the FSM depicted in Figure 4c). If a device shows no activity, its device controller requests to turn off the device, and the device can non-deterministically accept the request or decline it. A domain power controller powers on the domain when one of the devices in the domain shows activity. It powers off the domain when all device controllers in the power domain indicate their devices are turned off. The top power controller powers on if one of the domain power controllers indicates it needs power. It powers off if all domain power controllers indicate that all devices are turned off.

To obtain power domain models with a deadlock, one of the device controller FSMs is changed such that in its *off* state it expects to read $act(0)$, and in its on state, it expects to read $act(1)$, which leads to synchronisation issues and deadlocks.

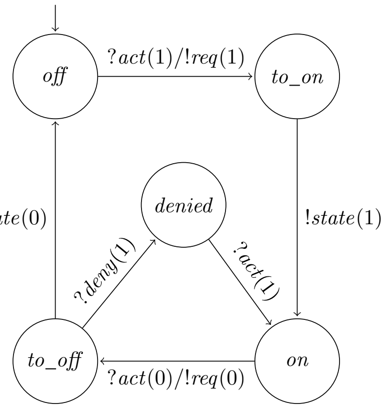
All experiments were executed on a MacBook Pro 2015, 2,7GHz Intel Core i5, 16Gb RAM, running MacOS Catalina 10.15.4. For SAT solving, we use the Z3 solver, version 4.8.0 64-bit [9]. For reachability checks, we use nuXmv, version 2.0.0 64-bit [5]. Instructions to reproduce the experiments and the script used to obtain our results are available at [1].

B. Results

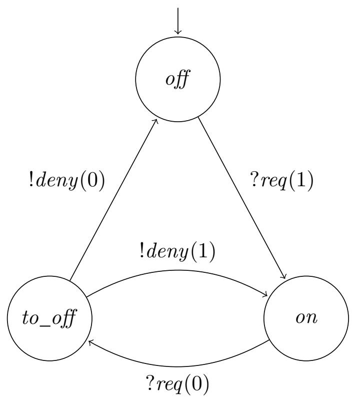
Model	#FSMs	Live	SAT		Reachability	
			Res.	Time (s)	Res.	Time (s)
gonogo_1	2	✓	✓	0.1	✓	0.2
gonogo_1_dl	2	✗	✗	0.1	✗	0.3
gonogo_2	6	✓	✓	0.1	✓	0.5
gonogo_2_dl	6	✗	✗	0.1	✗	2.4
gonogo_3	14	✓	✓	0.3	✓	1.5
gonogo_3_dl	14	✗	✗	0.3	✗	5.9
gonogo_4	30	✓	✓	0.6	✓	5.5
gonogo_4_dl	30	✗	✗	0.6	✗	18.0
gonogo_5	62	✓	✓	2.0	✓	21.2
gonogo_5_dl	62	✗	✗	1.9	✗	54.9
gonogo_6	126	✓	✓	7.9	✓	92.7
gonogo_6_dl	126	✗	✗	6.7	✗	221.7
power1_5	25	✓	✓	0.4	✓	1.6
power1_5_dl	25	✗	✗	0.2	✗	2.1
power10_5	259	✓	✓	14.0	✓	120.0
power10_5_dl	259	✗	✗	10.1	✗	104.2
power20_5	519	✓	✓	57.5	✓	564.8
power20_5_dl	519	✗	✗	50.3	✗	451.1
power30_5	779	✓	✓	352.5	✓	1597.4
power30_5_dl	779	✗	✗	262.9	✗	1107.3
power40_5	1039	✓	✓	410.2	✓	n/a
power40_5_dl	1039	✗	✗	245.6	✗	n/a
power50_5	1299	✓	✓	542.2	✓	n/a
power50_5_dl	1299	✗	✗	481.1	✗	n/a

Table I: Experimental results

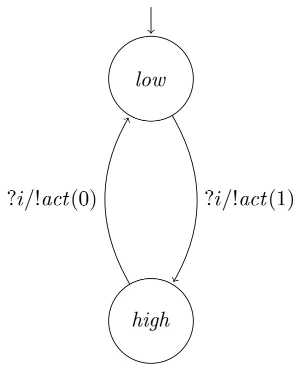
The times required for the experiments are reported in Table I. The *Model* column indicates the model that is evaluated. For go/no go models, the number in the name signifies the number of blocks. For power domain models, the first and second number in the name denote the number of power domains and device-controller pairs in every domain, respectively. *#FSMs* reports the number of FSMs in the model. In the *Live* column, ✓ indicates that the model is deadlock free, ✗ indicates it is not. For each instance, we list the result



(a) Controller.

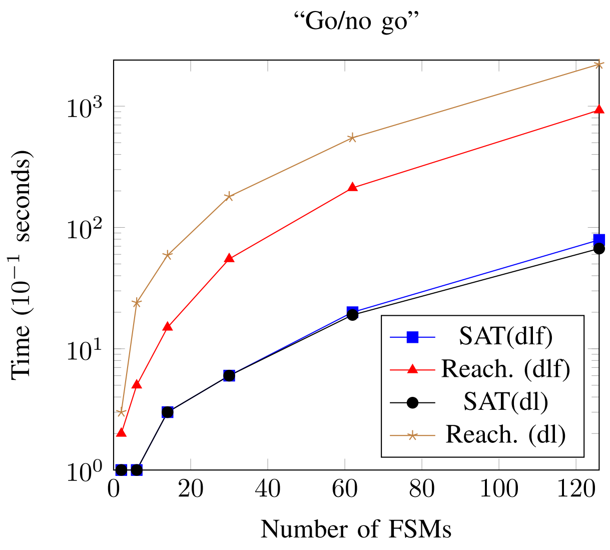


(b) Device.



(c) Activity generator.

Figure 4: Some of the power domain FSMs



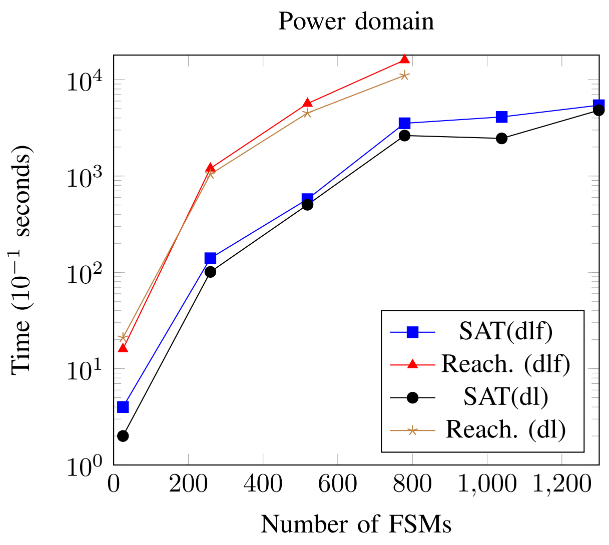


Figure 5: Number of FSMs vs time for all experiments.

reported by the tool (*Res.*), where ✓ and ✗ represent absence and presence of deadlocks, respectively. Running time for each instance is reported in seconds.

For both sets of models, SAT and reachability correctly report absence and existence of deadlocks in all models. The largest go/no go models contain 126 FSMs. Liveness of the largest deadlock free go/no go model is proven using SAT in 7 seconds. Reachability analysis takes 1 minute 32 seconds for the same go/no go model. For the largest go/no go model with a deadlock, a deadlock is reported using SAT in 6 seconds. Using reachability it can be proven that a deadlock state is reachable in 3 minutes 41 seconds. As for the power domain experimental set, the largest models (both with and without deadlock) contain 1299 FSMs. For the largest model without a deadlock, SAT proves liveness in 9 minutes and 2 seconds. Analysis of the largest power domain model with a deadlock takes 8 minutes and 1 second using SAT. Reachability analysis for the power domain models with numbers of power domains larger than 30 was not possible in our case. This was caused by NUXMV exceeding the maximum allowed stack on MacOS.

C. Discussion

The results show that using our technique we can prove liveness of large xMAS models with FSMs. We plot the performance results on both sets of models in Figure 5. Note that we use the log-scale for the y-axis. In addition, we use deciseconds instead of seconds in order to avoid values less than 1 for the y-axis. The results show that both methods scale exponentially in the number of FSMs. However, using SAT for liveness verification significantly outperforms reachability for xMAS extended with FSMs. This is in line with our expectations, and aligns with results for standard xMAS [12].

Although we do not encounter false deadlocks in our experiments, the fact that our method is incomplete implies that finding false deadlocks using SAT is possible. If SAT reports a deadlock, it is not known if the deadlock is reachable or not. In that case, reachability analysis is necessary.

VI. Conclusions

We demonstrated that the approach to verify liveness of xMAS networks with FSMs proposed by Verbeek *et al.* [19] is unsound. We proposed new idle and block equations for xMAS networks containing FSMs, and proved their soundness. Our experimental evaluation shows that deadlock detection using satisfiability outperforms reachability analysis using symbolic model checking in nuXmv. In case deadlocks are found, the latter can, however, verify their reachability reasonably efficiently. Although our method is incomplete, this was not observed during the experiments.

As future work, we plan to investigate ways to make the method complete. In particular, an alternative encoding to SAT based on bounded model checking, could make the method complete provided an appropriate bound can be derived. Additionally, the FSMs presented in this paper always read from and write to exactly one channel. This restriction could be relaxed to read and write multiple channels on a single transition to enable more compact modeling of some FSMs.

References

[1] Description of go/no go and power domain models (MaDL github wiki). https://github.com/MaDL-DVT/madl-dvt/wiki/FMCAD20-Experiments.

[2] MaDL design and verification tools. https://github.com/MaDL-DVT/madl-dvt.

[3] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] F. Burns, D. Sokolov, and A. Yakovlev. GALS synthesis and verification for xMAS models. In *DATE 2015*, pages 1419–1424. EDA Consortium, 2015.

[5] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv symbolic model checker. In *CAV 2014*, pages 334–342, 2014.

[6] S. Chatterjee and M. Kishinevsky. Automatic generation of inductive invariants from high-level microarchitectural models of communication fabrics. In *CAV 2010*, pages 321–338. Springer, Berlin, Heidelberg, 2010.

[7] S. Chatterjee, M. Kishinevsky, and U. Y. Ogras. xMAS: Quick formal modeling of communication fabrics to enable verification. *IEEE Design Test of Computers*, 29(3):80–88, 2012.

[8] S. Das, C. Karfa, and S. Biswas. xMAS based accurate modeling and progress verification of NoCs. In *VLSI Design and Test*, pages 792–804. Springer, Singapore, 2017.

[9] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS 2008*, LNCS, pages 337–340, Berlin, Heidelberg, 2008. Springer.

[10] A. Fedotov, J. J. A. Keiren, and J. Schmaltz. *Sound idle and block equations for finite state machines in xMAS*. Computer science reports. Technische Universiteit Eindhoven, 11 2019.

[11] A. Fedotov and J. Schmaltz. Automatic generation of hardware checkers from formal micro-architectural specifications. In *DATE 2018*, pages 1568–1573, 2018.

[12] A. Gotmanov, S. Chatterjee, and M. Kishinevsky. Verifying deadlock-freedom of communication fabrics. In *VMCAI 2011*, pages 214–231. Springer, Berlin, Heidelberg, 2011.

[13] S. J. C. Joosten and J. Schmaltz. Generation of inductive invariants from register transfer level designs of communication fabrics. In *Proc. MEMOCODE 2013*, pages 57–64, 2013.

[14] S. J. C. Joosten and J. Schmaltz. Automatic extraction of micro-architectural models of communication fabrics from register transfer level designs. In *Proc. DATE 2015*, pages 1413–1418, 2015.

[15] Z. Lu and X. Zhao. xMAS-based QoS analysis methodology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(2):364–377, 2018.

[16] F. Verbeek and J. Schmaltz. Hunting deadlocks efficiently in microarchitectural models of communication fabrics. In *Proc. FMCAD 2011*, pages 223–231, 2011.

[17] F. Verbeek and J. Schmaltz. Automatic generation of deadlock detection algorithms for a family of microarchitecture description languages of communication fabrics. In *Proc. HLDVT 2012*, pages 25–32. IEEE, 2012.

[18] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh. ADVOCAT: Automated deadlock verification for on-chip cache coherence and interconnects. In *DATE 2016*, pages 1640–1645, 2016.

[19] F. Verbeek, P. M. Yaghini, A. Eghbal, and N. Bagherzadeh. Deadlock verification of cache coherence protocols and communication fabrics. *IEEE Transactions on Computers*, 66(2):272–284, 2017.

[20] S. Wouda, S. J. C. Joosten, and J. Schmaltz. Process algebra semantics & reachability analysis for micro-architectural models of communication fabrics. In *Proc. MEMOCODE 2015*, pages 198–207. IEEE, 2015.