Post-reboot Equivalence and Compositional Verification of Hardware

Zurab Khasidashvili, Marcelo Skaba, Daher Kaiss, Ziyad Hanna
Intel, IDC, Haifa, Israel
{zurabk, smarcelo, dkaiss, zhanna}@iil.intel.com

Abstract

We introduce a finer concept of a Hardware Machine, *where the set of post-reboot operation states is explicitly a part of the FSM definition. We formalize an ad-hoc flow of combinational equivalence verification of hardware, the way it was performed over the years in the industry. We define a concept of* post-reboot bisimulation, *which better suits the Hardware Machines, and show that a right form of combinational equivalence is in fact a form of post-reboot bisimulation. Further, we show that alignability equivalence is a form of post-reboot bisimulation, too, and the latter is a refinement of alignability in the context of compositional hardware verification. We find that post-reboot bisimulation has important advantages over alignability also in the wider context of formal hardware verification, where equivalence verification is combined with* formal property verification *and with* validation of a reboot sequence*. As a result, we propose a more comprehensive, compositional, and fully-formal framework for hardware verification. Our results are extendible to other forms of labeled transition systems and adaptable to other forms of bisimulation used to model and verify complex hardware and software systems.*

1. Introduction

This work addresses formal hardware verification. The aim of hardware *equivalence verification* is to check for "functional equivalence" of two design models according to some concept of equivalence. The equivalence of two models does not guarantee that they do what they are designed for, and it is a task of formal *property verification* and dynamic validation to provide a level of confidence that the designs have the desired functionality. In practice, equivalence verification and property verification are closely related and are often performed under a common methodology umbrella and tool set. To date, however, there has not been any significant research towards providing a unifying theory that would combine equivalence verification with property verification using *fully formal, full-proof methods* (and not relying on a non-exhausting simulation).

Since verification of complex hardware is not imaginable without employing abstraction and compositional methods, hardware equivalence verification in practice consists in the following steps:

1. Decompose the specification and implementation models using mapped cut points in them.
2. Use boundary constraints to make the corresponding component slices equivalent.
3. Build a reboot sequence that via 3-valued simulation [HC98] brings the models into states satisfying the boundary constraints (and possibly other properties).
4. Check that all properties remain valid post-reboot, using (non-exhaustive, 3-valued) simulation.

The most widespread equivalence verification method in the industry was and still is some form of *combinational verification* [KvE04], where the slices are combinational, i.e., they contain no internal state elements. Hardware is represented as a Finite State Machine (FSM) [Koh78, HS96]. In its classical definition, combinationally equivalent FSMs M_1 and M_2 are actually the same FSMs: their transition relations and output functions are defined via the same Boolean functions that may be implemented differently in M_1 and M_2. In practice, however, a form of assume-guarantee framework is used (like the one outlined above), where the requirement of component equivalence is weakened to a form of conditional equivalence under "don't cares", or under combinational or temporal logic assumptions. Thus, combinational equivalence in practice does not correspond to its classical definition.

For hardware FSMs designed to operate correctly after simulating them with a *reboot sequence*, several concepts of equivalence have been developed (besides combinational equivalence), such as *sequential hardware equivalence,* also called *alignability equivalence* [Pix92], *delayed safe replaceability* [SPAB01], *exact 3-valued equivalence* [RSSB99], and *steady-state equivalence* [KH02]. The latter two forms of equivalence have been used for *retiming verification* [LS91], which is the second most widespread form of hardware equivalence verification. Retiming verification of sequential components is often combined with combinational verification of combinational components. It is unclear what kind of equivalence is proved between the specification and implementation FSMs as a result of such a combination of verification methods. Furthermore, step 4 of the above outlined procedure of compositional verification, which relates reboot sequence with the used boundary properties, was never considered as part of formal equivalence verification. Indeed, it is currently based on a non-exhaustive dynamic simulation and thus cannot guarantee a full proof.

The main results of this work are:

1. *From practice to theory*: We formalize several dominating hardware equivalence verification methods into a unified theory (we identify and fill the verification holes along the way) – the outcome is *post-reboot equivalence.*
2. *From theory to practice*: We propose (fully) formal, practically applicable hardware equivalence verification algorithms and methodology allowing combination of formal equivalence verification with formal property verification.
3. *Relevance in practice*: We present experimental evidence demonstrating that the new theory makes difference in the practice of full-chip verification.
4. *Extendible and adaptive concepts*: While we focus on *bit-wise* hardware machines, the new equivalence concept is defined in terms of bisimulation, which allows (1) extending the results to (possibly non-deterministic) Labeled Transition Systems, into which both hardware and software can be modeled at higher levels of abstraction, and (2) adapting it to other useful forms of (bi)simulation.

The concept of post-reboot equivalence proposed here is a refinement of alignability equivalence [Pix92]. The compositional verification framework that we propose is based on a more recent work [KSKH04] that proves *weak compositionality* of alignability via defining a concept of *stable decomposition* of the specification and implementation FSMs. The latter framework assumes, however, that the two FSMs are weakly synchronizable (WS for short [PR96]) with the same input sequence, but it does not provide any practical algorithm for formally verifying whether or not a given input vector sequence is a ws-sequence. This makes the methodology proposed in [KSKH04] incomplete. This incompleteness is caused by the fact that the latter work used the alignability equivalence as the basis, and it is unclear how a practical, formal verification method can be proposed for verifying ws-sequences without adopting the post-reboot equivalence concept, as we do here. Furthermore, alignability equivalence is not satisfactory in practice, since it is not enough for a reboot sequence to be a ws-sequence: to make the hardware work properly, the reboot sequence should bring it to a designated set of states that meet some architectural requirements. For example, physical addresses in the memory, initial values of counters, and some states at the internal sub-systems must have specific values. The concept of WS does not capture these requirements. Finally, alignability equivalence is not expressive enough to relate the provability of commonly observable temporal logic formulas in one FSM to their validity in an equivalent FSM. These points will later be clarified in detail.

The theoretical contribution of our work can briefly be summarized as follows: We introduce a concept of *post-reboot bisimulation* as a pair (π, B), where B is a bisimulation between compatible FSMs M_1 and M_2 (i.e. the FSMs have the same inputs and outputs) and π brings any pair of states of $M_1 \times M_2$ into a pair in B. We formalize the concept of *post-reboot combinational equivalence* and show that it is a post-reboot bisimulation. We show that alignability is a post-reboot bisimulation, too. We also show that the set of all post-reboot bisimulations (when it is non-empty) forms a complete lattice [DP90]. Its top element corresponds to bisimulations formed from all ws-states, and the bottom element corresponds to the bisimulations formed from the states in *sink strongly connected components* [PR96] of M_1 and M_2. Post-reboot combinational verification and compositional alignability verification correspond to building post-reboot bisimulations that are between the top and bottom bisimulations, and therefore these forms of verification are feasible in practice.

Further, we define an upper semi-lattice [DP90] of weak-synchronizing sequences. It is in fact this order that allows us to demonstrate how "the strength" of a reboot sequence can affect the post-reboot validity of a temporal specification of the design – and this indeed clarifies the subtle differences between post-reboot equivalence and alignability: Since a reboot sequence π must bring any state pair of $M_1 \times M_2$ into a non-empty bisimulation B, the sequence π must be a ws-sequence for both M_1 and M_2. The converse need not be true: some of the ws-sequences cannot serve as useful reboot sequences because they may not meet some *non-functional* requirements that should be satisfied during the post-reboot operation of the circuit. Examples of non-functional requirements (for the FSM formalism) are power and timing constraints, as well as the architectural requirements mentioned above, which may be satisfied in some but not all ws-states. Non-functional requirements can also be temporal logic specifications that were not encoded into the circuit design as observable output behaviors. Thus, post-reboot bisimulation can be viewed as a useful *refinement* of alignability equivalence, especially when equivalence verification is considered in a wider context of formal verification of hardware designs, and may be combined with verification of temporal properties and validation of (candidate) reboot sequences.

In the next section, we recall the FSMs and concepts related to alignability and introduce Hardware Machines. In section 3, we introduce post-reboot bisimulation, relate it to alignability, and give its lattice-theoretic characterization. In Section 4, we propose a revised definition of combinational equivalence. We discuss the advantages of post-reboot bisimulation for verification of hardware machines in Section 5. Our conclusions appear in Section 6.

2. Hardware Machines

In this section, we introduce Hardware Machines, to reflect the fact that the set of operation states of a hardware design is a subset, usually proper, of the WS-states.

Definition 2.1 [Koh78] A Finite State Machine (FSM) M is a tuple $(S,\Sigma,\Gamma,\delta,\lambda)$, where S is a finite set of states (ranged

over by s, t, s_1, …); Σ is a finite input alphabet (ranged over by a, …); Γ is a finite output alphabet (ranged over by e, …); $\delta: S \times \Sigma \rightarrow S$ is a state transition (or next-state) function; and $\lambda: S \times \Sigma \rightarrow \Gamma$ is an output function. Here, Σ and Γ correspond to the Boolean vectors of input and output variables (i.e., bits that can be 0 or 1) and, similarly, states are Boolean vectors of state (i.e., latch) variables.

*Notation***:** Below, unless otherwise stated, M_1 and M_2 denote *compatible* FSMs, i.e., M_1 and M_2 have the same sets of inputs and outputs. We denote the state set of M by S(M). Further, π and ρ denote input sequences for M. We write a: $s \rightarrow t$ if $\delta(s,a) = t$, and we write π: $s \rightarrow^* t$ if π transforms s into t (here $\rightarrow^*$ denotes transitive reflexive closure of →). Recall that a: $s \rightarrow t$ means that input a brings M from state s (the current state) to state t (the next state); and $\lambda(s,a) = e$ means that at current state s, if the input is a, the output value (at current state) is e. Finally, $O_M(s,\pi)$, or simply $O(s,\pi)$, denotes the output value of M after simulating M with π, where M is initially at state s, and $S(s,\pi)$ denotes the state into which π brings s.

We recall that the *product* $M_1 \times M_2$ of M_1 and M_2 has the same inputs and outputs as the two FSMs; its states are pairs of states of M_1 and M_2, and its output function and state transition function are pairs of the output functions and state transition functions of M_1 and M_2, respectively.

Definition 2.2

- [Koh78, HS96] Let M_1 and M_2 be FSMs. States $s_1 \in S(M_1)$ and $s_2 \in S(M_2)$ are *equivalent*, written as $s_1 \approx s_2$, if $\forall \pi: O(s_1, \pi) = O(s_2, \pi)$. The state (s_1,s_2) is then called an *equivalent state* of the product machine $M_1 \times M_2$.
- [PR96] A *weak synchronizing sequence* (*ws-sequence* for short) of an FSM M is an input sequence that brings M from any state to a subset of equivalent states $\{s_1,\ldots,s_m\}$. Each such state s_i is called a *ws-state* of M, and M is called weakly-synchronizable.

Below, $StateEq(M_1,M_2)$, or simply StateEq, denotes the state equivalence relation on $S(M_1) \times (M_2)$; ≈ is the infix notation for StateEq. By equivalent states we always mean state-equivalence. Note that every state reachable from a ws-state of M is a ws-state. WS(M), or simply WS, denotes the set of all ws-states of M.

Definition 2.3 [Pix92] Let M_1 and M_2 be FSMs.

- A binary input sequence π is an *aligning sequence* for a state (s_1,s_2) of $M_1 \times M_2$ if it brings $M_1 \times M_2$ from state (s_1,s_2) to an equivalent state.
- M_1 and M_2 are *alignable*, $M_1 \approx_{aln} M_2$, if every state of $M_1 \times M_2$ has an aligning sequence.

It is shown in [Pix92] that $M_1 \approx_{aln} M_2$ iff there is a sequence (called a *universal aligning sequence)* that aligns each state of $M_1 \times M_2$. The following theorem is an easy consequence of the results of [Pix92].

Alignment Theorem: FSMs M_1 and M_2 are alignable if and only if each FSM is weakly synchronizable and there is an equivalent pair $s_1 \approx s_2$ of states in M_1 and M_2. The concatenation of ws-sequences of M_1 and M_2 is a ws-sequence for both of them and it weakly synchronizes M_1 and M_2 into equivalent ws-states (when $M_1 \approx_{aln} M_2$).

If an FSM is not weakly-synchronizable, for any input sequence ρ, there always exist power-up states s_1 and s_2 such that the states $O(s_1, \rho)$ and $O(s_2, \rho)$ are not equivalent. This means that, whatever the ρ, the FSM exhibits a non-deterministic observational (output-) behavior after ρ. Therefore, in the alignability equivalence, weak synchronization is a necessary condition for an FSM to be equivalent to another FSM (or to itself). Below in the discussion, we will only consider such FSMs.

Since alignability equivalence is only concerned with the output behavior, and equivalent states of an FSM cannot be distinguished by observing the outputs, Pixley [Pix92] worked with equivalence classes of states $[s]_{\approx}$. He showed that, for any non-equivalent ws-states s and t, there is a transition path from (an element of) $[s]_{\approx}$ to (an element of) $[t]_{\approx}$ and vice versa. Hence, whatever the ws-sequence π is chosen to synchronize an FSM M, the set of its post-ws states, up-to ≈, is always the same – it coincides with the set of equivalence classes of WS(M). Thus, for alignability equivalence, the set of all ws-states is (implicitly) considered as the set of operation states.

In practice, equivalence verification between FSMs M_1 and M_2 is usually combined with verifying that the specification model M_1 (written in a hardware description language) satisfies its temporal logic specification, say P. In this wider context, working with equivalence classes of states is inadequate, as if a state s satisfies P, its equivalent states need not to. And for ws-states, it is no longer valid that there is a transition path between any pair of ws-states. Therefore, it does make a difference which ws-sequence is chosen to weakly synchronize the FSMs – the resulting sets of post-ws operation states may be different. For one ws-sequence π, the respective post-ws operation states of M_1 may satisfy P while for another ws-sequence ρ, the resulting post-ws operation states might not satisfy P. A simple example of this is given in Section 5. This observation led us to introduce the following concept:

Definition 2.4 A *Hardware Machine* (HM) is a pair H=(M, R), where M is an FSM and $R \subseteq WS(M)$ is closed under transition; R is called the set of *operation states* of H.

In the above definition, R must be understood as a set of states into which a ws-sequence ρ brings H from any state. We could have chosen to make ρ a part of definition of an

HM. We will see below that R can be defined as a set of constraints on the boundaries of component slices of a suitable decomposition of M, thus we found it more natural to use R than ρ as part of the HM definition.

3. Post-reboot equivalence

In this section, we introduce post-reboot bisimulation (PRB) and post reboot equivalence, and construct the lattice of PRBs. We show how PRB is related to alignability and discuss how the two differ (theoretically), by defining an upper-semi-lattice on ws-sequences. We also relate PRB with FSM bisimulation as defined in [AGM01].

The following concept of bisimulation for compatible FSMs is induced by the bisimulation concept for LTSs [Par81, Mil89]. Recall that an FSM can be viewed as an LTS by considering a pair $(a, \lambda(s,a))$ as the label for transition $s \rightarrow \delta(s,a)$, where a is an input [HS96].

Definition 3.1: Let $M_i = (S_i, \Sigma, \Gamma, \delta_i, \lambda_i)$, i = 1,2, be FSMs, and let $B \subseteq S_1 \times S_2$ be a relation such that:

- $B(s_1,s_2) \Rightarrow \forall a \in \Sigma$: $\lambda_1(s_1,a) = \lambda_2(s_2,a)$ & $B(\delta_1(s_1,a),\delta_2(s_2,a))$.

Then, B is called an FSM *bisimulation* and M_1 and M_2 are called *bisimilar* with respect to B. States $(s_1,s_2) \in S_1 \times S_2$ are called *bisimilar*, written as $s_1 \sim s_2$, if they are contained in a bisimulation on $S_1 \times S_2$.

State equivalence is an early form of FSM-bisimulation: $s_1 \sim s_2$ iff $s_1 \approx s_2$. Therefore, the state equivalence relation StateEq $\subseteq S(M_1) \times S(M_2)$ is the largest (w.r.t. $\subseteq$) bisimulation between M_1 and M_2. It in interesting to note that FSMs M_1 and M_2 are *replaceable*, or *FSM-equivalent* [Koh78], iff StateEq(M_1,M_2) is a non-empty bisimulation.

Corollary 3.1 Let M_1 and M_2 be FSMs, let $S_1 \subseteq S(M_1)$ and $S_2 \subseteq S(M_2)$ be closed under transition, and let StateEq(S_1,S_2) = StateEq$\cap(S_1 \times S_2)$. Then StateEq(S_1,S_2) is the largest bisimulation between M_1 and M_2 contained in $S_1 \times S_2$.

Definition 3.2 Let $M_i = (S_i, \Sigma, \Gamma, \delta_i, \lambda_i)$, i = 1,2, be FSMs, let π be a sequence of inputs from Σ, and let $B \subseteq S_1 \times S_2$ be a bisimulation between M_1 and M_2. A pair (π,B) is a *post-reboot bisimulation* (or *PRB*) between M_1 and M_2 iff:

- $\forall (s_1,s_2) \in S_1 \times S_2$. ($\pi$: $s_1 \rightarrow^* t_1$ & π: $s_2 \rightarrow^* t_2$) $\Rightarrow B(t_1,t_2)$.

M_1 and M_2 are called *post-reboot bisimilar* or *post-reboot equivalent* if there is a PRB between them.

Theorem 3.1 The set of all post-reboot bisimulations between FSMs M_1 and M_2, when it is a non-empty set, is a complete lattice w.r.t. the partial order defined by: $(\pi_1,B_1) \leq (\pi_2, B_2)$ iff $B_1 \subseteq B_2$.

Let (π,B) be a PRB between M_1 and M_2; then one can associate with π a smallest bisimulation B_π such that (π,B_π) is a PRB; B_π is the intersection of all B_i such that (π, B_i) is a PRB. Therefore, we can define a (strict) order on input sequences as follows: $\pi_1 \prec \pi_2$ iff $B_{\pi 1} \supset B_{\pi 2}$; that is, π_1 cannot transfer all state pairs of $M_1 \times M_2$ into $B_{\pi 2}$, whereas π_2 can; therefore, we call π_2 a *stronger* reboot sequence than π_1. We write $\pi_1 \equiv \pi_2$ iff $B_{\pi 1} = B_{\pi 2}$. When $M_1 = M_2$, the order $\prec$ is in fact an order on the ws-sequences of M_1. The order $\prec$ has upper bounds but need not have a bottom element, thus need not be a lattice. Indeed, consider the FSM $M_{\therefore}$ in Figure 1. Note that $s_1 \approx s_4$ and $s_2 \approx s_3$. Therefore, input sequences 1 and 0 are both ws-sequences of $M_{\therefore}$ and so is 10. Note that $1 \equiv 11 \equiv 111 \ldots$; $0 \equiv 00 \equiv 000\ldots$; and $01 \equiv 10 \equiv 100\ldots$. Thus, quotient $\prec / \equiv$ has three elements in $M_{\therefore}$, ordered as in Figure 1.a.

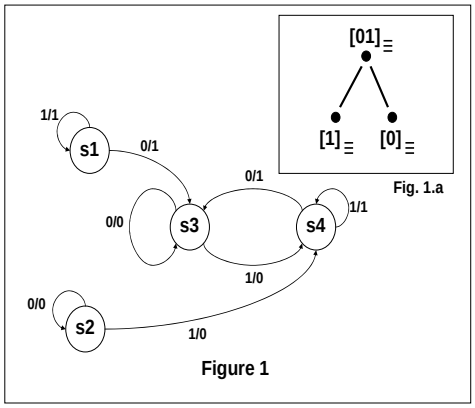

Figure 1

The deep analysis of weak synchronization via *strongly connected components* (or SCCs) of the state transition graphs presented in [PR96] is closely related to our lattice-theoretic characterization of PRB: Recall that an SCC is a set of states where between any two states there is a state transition path. A *sink* SCC is one from which there is no exiting transition. Then, in any smallest PRB (π, B), states in B belong to sink SCCs of M_1 and M_2 and π is a strongest ws-sequence. This follows easily from the construction of ws-sequences that bring any state into a sink SCC, in [PR96], and from Corollary 3.1.

Post-reboot bisimulation relates to the bisimulation concept for FSMs with start states [AGM01] as follows. If an FSM M comes with a start state $s^I \in S(M)$, we assume that there is an input sequence π^I, often of length 1, such that $\forall s \in S(M)$. π^I:$s \rightarrow^* s^I$; that is, π^I is a synchronizing sequence for M. Such an assumption is natural for hardware FSMs, since hardware can power up at any state, and assumption of a start state actually implies assumption of a ws-sequence that brings the FSM into the start state. Now let M_1 and M_2 be FSMs with start states s^I_1 and s^I_2, respectively. Then [AGM01] requires (s^I_1,s^I_2) to belong to any bisimulation $B \subseteq S_1 \times S_2$, which is equivalent to requiring that (π^I,B) is a PRB for any non-empty bisimulation $B \subseteq S_1 \times S_2$, where π^I is such that $\forall s_1 \in S_1$. π^I: $s_1 \rightarrow^* s^I_1$ and $\forall s_2 \in S_2$. π^I: $s_2 \rightarrow^* s^I_2$; but any non-empty bisimulation B should anyway contain (s^I_1,s^I_2) because B is closed under transition and π^I transfers every state pair into (s^I_1,s^I_2).

Definition 3.2 Let M_1 and M_2 be FSMs. Let $S_1 \subseteq S(M_1)$ and $S_2 \subseteq S(M_2)$. We call a bisimulation $B \subseteq S(M_1) \times S(M_2)$ an *on-to bisimulation on $S_1 \times S_2$* iff $B \subseteq S_1 \times S_2$ and

- $\forall s_1 \in S_1.\ \exists s_2 \in S_2.\ B(s_1,s_2)$ & $\forall s_2 \in S_2.\ \exists s_1 \in S_1.\ B(s_1,s_2)$.

Post-reboot equivalence is related to alignability as follows.

Theorem 3.2 Let WS_1 and WS_2 be weak-synchronization states of FSMs M_1 and M_2, respectively ($WS_1 = \varnothing$ if M_1 is not weakly synchronizable, and similarly for WS_2 and M_2). Further, let $StateEq(WS_1,WS_2) = StateEq \cap (WS_1 \times WS_2)$. Then the following are equivalent:

1. M_1 and M_2 are alignable;
2. $StateEq(WS_1,WS_2) \neq \varnothing$;
3. $StateEq(WS_1,WS_2)$ is a non-empty on-to bisimulation, on $WS_1 \times WS_2$, between M_1 and M_2.
4. M_1 and M_2 are post-reboot equivalent.

It is easy to see that in any PRB (π, B) between M_1 and M_2, π is a universal aligning sequence for M_1 and M_2, thus $\prec$ orders universal aligning sequences of M_1 and M_2.

4. Combinational equivalence as post-reboot equivalence

In this section, we examine combinational verification of state-matching FSMs, expose the verification holes in its current formalism, and relate it to post-reboot equivalence in an attempt to come up with a satisfactory formalism.

To do this, we explain in an example the compositional alignability verification framework proposed in [KSKH04]. The specification and implementation FSMs M_1 and M_2 are decomposed into components, as in Figure 2 below.

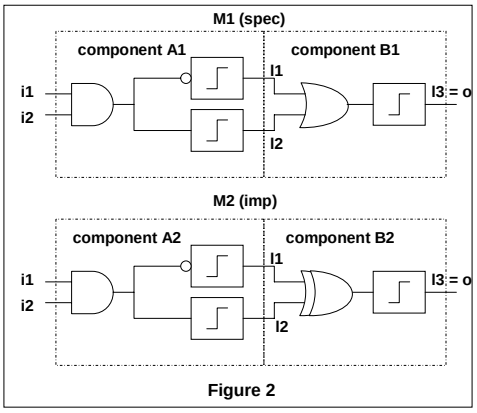

Figure 2

A 1-1 correspondence between the component slices of M_1 and M_2 is defined via mapping the corresponding slice boundaries, where the boundary signals are latches. For example, the boundary signals of components A_1 and A_2 are mapped (they have the same names), and so are the boundary signals of B_1 and B_2. FSM decomposition is needed to reduce equivalence verification for M_1 and M_2 to equivalence verification of the components, and for this to work, boundary properties are added to the components to eliminate behaviors of the components that will never happen during a post-reboot behavior of the FSMs. For example, by using the constraint $l_1 = \neg l_2$, it is possible to prove that the *conditional* FSMs obtained from B_1 and B_2 by restricting the allowed input sequences are alignable. To make usage of such a constraint sound, one must insure that the constraint is valid in all post-reboot operation states. Indeed, in this example it is enough to use any ws-sequence as a reboot sequence to ensure the constraint. Since the constraint is actually the output constraint for components A_1 and A_2, its validity in all post-reboot states of M_1 and M_2 is proved locally in the components A_1 and A_2 – the constraint is valid in all post-ws states of A_1 and A_2.

There is another condition for a safe usage of boundary properties – the resulting conditional FSMs must be stable [KSKH04]. The intuition is that, in stable conditional FSMs, an input vector is allowed in a state transition path iff it is allowed in all state transition paths. Such a conditional FSM can be mapped to an equivalent (non-conditional) FSM whose input signature is a subset of that of the conditional FSM, therefore, the alignability theory is valid for stable conditional FSMs. Components B_1 and B_2 constrained with $l_1 = \neg l_2$ are stable conditional FSMs, because input vectors $l_1 = l_2 = 0$ and $l_1 = l_2 = 1$ are never allowed while the remaining two input vectors are always allowed.

Only a subset of boundary properties is used for the assume-guarantee compositional proofs. Such properties are called *verification properties*. Their conjunction is denoted by VP_D. A decomposition D of (M_1, M_2) is called *stable* if all the components are stable under the verification properties, and the output properties of each component are valid in ws-states of the component (constrained by respective input properties). When D is stable, VP_D determines a relation $R_D \subseteq S(M_1) \times S(M_2)$ as follows: $(s_1,s_2) \in R_D$ iff (s_1,s_2) satisfies VP_D and the induced state of each component in D is a ws-state for that component (constrained with VP_D). It is assumed that the same name (mapped) latches are assigned the same values in (s_1,s_2).

Now recall that two FSMs are called *state-matching* if there is a 1-1 mapping between their latches. Often, for state-matching FSMs, it is allowed that a latch in one model is mapped to more than one latch in the other model. Sometimes, a mapping may have a *polarity*: a latch in one model may be mapped on a negation of a latch in the other model. This slightly more general treatment can easily be reduced to a situation where the latch mapping is 1-1, and no polarity is involved, and we adopt these assumptions.

When performing combinational equivalence verification, a mismatch in functionality of a component pair is allowed if the supporting components (or the supporting logic) in M_1 and M_2 can never generate a value combination that produces the mismatch. For example, the outputs l_1 and l_2 of components A_1 and A_2 of FSMs M_1 and M_2 in Figure 2 can only generate values satisfying the "mutex" property $l_1 = \neg l_2$ (except the start time 0 when the

latches l_1 and l_2 may have arbitrary values). To formalize this intuition, let us call a decomposition D of $M_1 \times M_2$ *combinational* if D is stable and each latch of M_1 and M_2 is an output of a component of M_1 or M_2.

Definition 4.1 Let M_1 and M_2 be state-matching FSMs. We call M_1 and M_2 *combinationally equivalent* if there is a combinational decomposition D of (M_1, M_2) such that:

- Outputs of any matching component pair (A_1,A_2) in D are equal at the next time if their inputs satisfy VP_D at the current time and their outputs are equal at the current time.
- The verification properties on outputs of any component pair (A_1,A_2) in D are valid at the next time if their inputs satisfy VP_D at the current time and their outputs are equal at the current time.

As already explained on an example above, by allowing the properties on component boundaries, one actually assumes a reboot sequence, and expects the FSMs' behaviors to match *post-reboot*. Therefore, unless $M_1 \times M_2$ can be driven from an arbitrary state into a state satisfying R_D, combinational equivalence w.r.t. D is *meaningless* – it is vacuous. Hence the following definition:

Definition 4.2 Let M_1 and M_2 be combinationally equivalent w.r.t. a combinational decomposition D, and let there be an input sequence π that brings any state pair of $M_1 \times M_2$ into a state pair in R_D. Then we call M_1 and M_2 *post-reboot combinationally equivalent* (w.r.t. D).

The problem of verifying that a reboot sequence will bring the circuits into a bisimulation has not been addressed by formal methods and traditionally this was not considered as part of combinational equivalence verification, which is a verification gap. In Section 5, we will discuss how to formally verify whether a pair (π, B) is a PRB, implying that π is a legal reboot sequence for bisimulation B.

Theorem 4.1 State-matching FSMs M_1 and M_2 are combinationally equivalent iff there is a combinational decomposition D of (M_1,M_2) such that R_D is a bisimulation.

Note that, for a combinational decomposition D, if and only if R_D is a bisimulation, every component pair has matching equivalent ws-states, and thus is alignable (by the Alignment Theorem). Hence, we conclude from Theorem 4.1 that state-matching FSMs M_1 and M_2 are combinationally equivalent w.r.t. a combinational decomposition D iff all component pairs of D (constrained by VP_D) are alignable. Finally, we show that post-reboot combinational equivalence is a PRB.

Theorem 4.2 Post-reboot combinationally equivalent state-matching FSMs M_1 and M_2 are post-reboot equivalent.

As already mentioned, combinational verification is often combined with retiming verification on the same design. For weakly synchronizable FSMs, steady-state equivalence, used as the semantics for retiming verification, implies alignability [KH03], thus post-reboot equivalence. Therefore, the retiming verification with steady-state semantics can be safely used as part of compositional post-reboot equivalence verification, provided the retimed components are first proven to be weakly-synchronizable.

5. Verification of Hardware Machines

Now, we consider verification of hardware machines in a wider context, where equivalence verification is combined with assertion verification and reboot sequence validation. We do not intend to cover all methodological aspects; however, we demonstrate the advantages and adequacy of post-reboot equivalence for this task.

Definition 5.1 We call HMs $H_1=(M_1,R_1)$ and $H_2=(M_2,R_2)$ *equivalent* if there is a post-reboot bisimulation (π,B) between FSMs M_1 and M_2 such that $B \subseteq R_1 \times R_2$. We call a CTL* formula [CGP99] *valid* in H_1 iff it is valid in all states in R_1.

Consider the following FSM M_{PR}, taken from [PR96]. (We use the original notation for states: A,B,C, etc.)

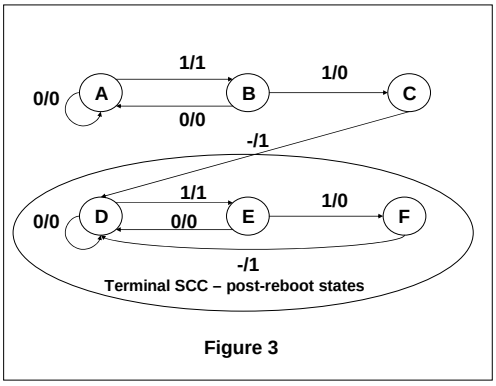

Figure 3

State pairs (A, D), (B, E) and (C, F) are equivalent. Since $A \approx D$, sequence 0 is a ws-sequence for M_{PR}, bringing states {A, B} into A and states {C, D, E, F} into D. Since all states are accessible from A and D, all states are ws-states. Clearly, one has $0 \equiv 01 \equiv 010 \prec 011 \prec 0111 \equiv 0110$. If the designer wants M_{PR} to operate in states {D, E, F} after reboot, he/she can choose a strongest reboot sequence, e.g. 0111, which transfers any state into the sink SCC {D, E, F}. Assume P is a property that is not valid at state C and is valid in {D, E, F}. P can be seen as a behavioral specification for a design that the designer wants to implement. Then, M_{PR} meets its behavioral specification with respect to the post-reboot semantics (for the reboot sequence 0111, for instance) but it does not meet its behavioral specification relative to the alignability semantics (e.g. when 0 is chosen as the ws-sequence). In this respect, adopting post-reboot semantics has a

significant impact on the verification methodology: since alignability does not distinguish between ws-sequences, adopting the alignability semantics forces the designer to modify the implementation of M_{PR} because of a "failing" property P. Here is the data from a chip design project that we supported: 75% of the logic bugs discovered by post-reboot simulation, after the equivalence verification was completed, were caused by the initialization issues.

For an FSM M, adopting the alignability semantics implies the need to prove for a property P the derived property $s \in WS(M) \Rightarrow P(s)$ for any state s of M. On the other hand, adopting post-reboot equivalence for a Hardware Machine (M,R) implies instead proving property $s \in R \Rightarrow P(s)$. We have seen that relation R can be computed (and is computed in practice) as the relation R_D associated to a stable decomposition of M (or M×M), while WS(M) cannot be computed for industrial designs, and it is unclear how a property $WS \Rightarrow P$ can be verified in general for a weakly-synchronizable FSM.

Let P be a CTL* formula written using the common (i.e., mapped) variables of M_1 and M_2 as the atomic propositions; such a formula is observable in both FSMs. Note that when we build a PRB (ρ,R_D) based on a stable decomposition D of (M_1,M_2), the HMs (M_1,R_1) and (M_2,R_2) are equivalent, where R_1 and R_2 are the projections of R_D on $S(M_1)$ and $S(M_2)$, respectively. Further, P is valid in (M_1,R_1) iff it is valid in (M_2,R_2) (cf. [Ch. 11, CGP99]). However, based on $M_1 \approx_{aln} M_2$ alone, it is no longer possible to infer the validity of P on the ws-states of M_2 from its validity on ws-states of M_1. Indeed, let M'_{PR} be the same FSM as the FSM M_{PR} in Figure 3, except now on input 0 the states A and B transition to D (rather than its equivalent state A). Then $M'_{PR} \approx_{aln} M_{PR}$ and {D, E, F} is the set of ws-states of M'_{PR} and, therefore, the property P (from the same example) is valid in all ws-states of M'_{PR} while it is not valid in the ws-state C of M_{PR}. That is, alignability equivalence for FSMs is not informative enough to allow inferring common observable properties from one model to its equivalent model. Note that considering the boundary latches as outputs and thereby strengthening alignability equivalence (by ensuring that the atomic propositions have the same values in M_1 and M_2) does not help us resolve the validity preservation problem with alignability, because the boundary properties are also used in proving the common observable properties.

In Tables 1 and 2, we present information on the verification of 5 assertions on the specification model. In both experiments, the boundaries of the cones on which the properties are checked are built using mapped latches (at this point, the equivalence of corresponding components of specification and implementation is already proven using the verification properties). If verification fails because of a spurious counter-example, the cone is expanded and verification is rerun. The use of the (boundary) verification properties as assumptions is allowed in the experiment in Table 1 and not allowed in the experiment in Table 2. In both cases, a SAT-based initialization algorithm is used to weakly synchronize the cones [RH02]. Thus, the first experiment closely corresponds to post-reboot equivalence verification with respect to the post-reboot bisimulation defined by the stable decomposition employed in the equivalence verification. Since we weakly synchronize the cones before verifying the properties, the second experiment is only an approximation of proving $WS \Rightarrow P$, because the computed synchronization sequence may reset the cone into a proper subset of the WS states. (We do not know how to prove $WS \Rightarrow P$ for large FSMs.)

In the tables, we present the highest level of expansion iterations (EI), the size of the cones, the number of boundary properties (BP) used in assertion verification, and the runtimes. All properties in Table 1 are verified using a SAT-based model checker, whereas the same properties in Table 2 cannot be verified. Some of them cannot be verified because of failures to weakly synchronize the cones, and some because of the resulting spurious counter-examples. As expected, the use of boundary properties as assumptions helps confine verification to smaller cones.

assertions	EI	inputs	latches	gates	BP	cpu (sec)
assert 1	2	114	26	1674	17	154
assert 2	1	24	20	221	1	155
assert 3	0	18	3	1012	9	209
assert 4	0	835	4	6986	64	209
assert 5	0	19	4	873	2	312

Table 1: assertion verification using boundary properties

assertions	EI	inputs	latches	gates	cpu (sec)
assert 1	10	125	526	7871	256
assert 2	10	369	1133	17690	262
assert 3	9	69	114	954	211
assert 4	2	858	37424	250745	596
assert 5	1	675	18	2585	1325

Table 2: assertion verification without boundary properties

Now let us turn to the question of verifying whether or not a sequence π is a ws-sequence for an FSM M. Since it is impossible to symbolically simulate [Jon02] a full-chip design, an over-approximation of states into which a reboot sequence brings a design is computed using 3-valued simulation, often also called X-simulation [HC98]. At the beginning of simulation, all latch values are set to the X value, and all inputs except a few reset signals are simulated with X. Propagation of reset signal values forces assignment of binary values to most of the latches, and the first part of the reboot sequence ends as soon as a fix-point of simulation is reached. Reboot sequence then continues to initialize counters, memory addresses, and other latches to specific values. To the best of our knowledge, no *practical, formal* methods exist for checking whether the over-approximation set of states of a combined full-chip design

$M_1 \times M_2$, built, as above, using 3-valued simulation, is indeed a subset of ws-states of $M_1 \times M_2$. Such a method would involve model checking on a huge state space.

The above question is actually irrelevant for achieving compositional post-reboot equivalence verification. Suppose a stable decomposition D of $M_1 \times M_2$ has been built. Let S be the 3-valued state obtained by simulating $M_1 \times M_2$ with π, starting from state X, let OS be the set of (binary) states of $M_1 \times M_2$ induced by S (latches with X values are assigned all possible binary value combinations), and let OS^* denote the closure of OS under the transition relation. Then it is enough to prove that (π, OS^*) is a PRB. The latter is a model-checking problem *on the components* of the decomposition (which are within the capacity of the model checker): for each component pair (A_1,A_2) of $M_1 \times M_2$, the state pair (t_1,t_2) induced by any state of OS must be an equivalence state of $A_1 \times A_2$, and the verification properties on the outputs of (A_1,A_2) must be valid for (t_1,t_2) and any state pair reachable from it by any input sequence of (A_1,A_2) satisfying its input properties.

6. Conclusions

We have proposed a new, finer formalism for modeling hardware – Hardware Machines, where the set of post-reboot operation states is a key component of the definition. This led us to introduce post-reboot equivalence, where, unlike alignability equivalence, the operation states play an important role in the semantics. Indeed, we could refine the alignability equivalence into a complete lattice of post-reboot bisimulations, and refine the homogeneous class of ws-sequences into an upper semi-lattice of reboot sequences. This new view of hardware also led us to a revision of the existing widespread equivalence concepts and the way they are employed in practice. We gain a new insight into compositional hardware verification, where the construction of a set of operation states is a by-product of building a stable decomposition of specification and implementation models. As a result of this revision, we were able to point to verification gaps in the existing methods, and propose a unified theory that bridges the verification practice to the Hardware Machine formalism.

We have briefly touched on the subject of assertion verification for Hardware Machines, demonstrating that the shift from FSMs to Hardware Machines implies important differences in the semantics of temporal logic assertions. We presented experimental evidence on how such a change in assertion semantics affects assertion verification in practice. We leave it to future work to come up with a comprehensive assertion verification theory and methodology that will be fully aligned with compositional equivalence verification and reboot sequence verification.

For non-state-matching designs, there are too many options to decompose the design into sub-circuits, and, at present, building stable decompositions is semi-automatic: heuristic latch mapping algorithms are used to define decomposition to start with, and then abstraction refinement methods [CGP99] are used to adjust the sub-circuit boundaries and add properties. Defining a fully automatic algorithm for building stable decompositions is a challenging direction for future work.

Acknowledgements: Without understanding a complete picture of design verification in practice, this work would have been impossible. We thank S. Goldenberg, R. Kaivola, M. Mneimneh, and A. Jas for the numerous discussions that oriented us in our search for a "right equivalence concept" for full-chip verification. We would also like to thank N. Piterman, A. Pnueli and M. Vardi for their valuable feedback that helped us sharpen the concepts.

References

[AGM01] Ashar P., A. Gupta, S. Malik, Using complete-1-destinguishability for FSM equivalence checking, ACM Trans. on Design Automation of Electronic Sys, vol. 6, no. 4, 2001.

[CGP99] Clarke E.M., O. Grumberg, D.A. Peled, Model Checking, MIT Press, 1999.

[DP90] Davey, B.A., Priestley H.A., *Introduction to Lattices and Order*, Cambridge University Press, 1990.

[HC98] Huang, S.-Y., K.-T., Cheng, *Formal Equivalence Checking and Design Debugging*, Kluwer, 1998.

[HS96] Hachtel G.D., F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer, 1996.

[Jon02] R.B. Jones, R.B. Symbolic Simulation Methods for Industrial Formal Verification, Kluwer, 2002.

[KH02] Khasidashvili Z, Z. Hanna, TRANS: Efficient sequential verification of loop-free circuits, HLDVT02.

[HH03] Khasidashvili, Z., Z. Hanna, SAT-Based methods for sequential hardware equivalence verification without synchronization, BMC'03, ENTCS 89 (4), 2003

[KSKH04] Khasidashvili Z., M. Skaba, D. Kaiss, Z. Hanna, Theoretical framework for compositional sequential hardware equivalence verification in presence of design constraints, ICCAD04, 2004.

[KvE04] Kuehlmann A., C.A.J van Eijk, Combinational and sequential equivalence checking, in: Logic Synthesis and Verification, Kluwer Academic Publishers, 2004.

[Koh78] Kohavi, Z., *Switching and Finite Automata Theory*, McGraw-Hill, 1978.

[Mil89] Milner, A.J.R.G. *Communication and Concurrency*, Prentice Hall, 1989.

[LS91] Leiserson C. E., J. B. Saxe. *Retiming synchronous circuitry*. Algorithmica, 6(1), 1991.

[Par81] Park, D. Concurrency and automata on infinite sequences, 5th GI-Conf. on TCS, LNCS 104, 1981.

[Pix92] Pixley, C. A theory and implementation of sequential hardware equivalence, IEEE trans. CAD, 1992.

[PR96] Pomeranz, I., S.M. Reddy, On removing redundancies from synchronous sequential circuits with synchronizing sequences, IEEE Trans. Computers, 1996.

[RSSB99] Ranjan R.K., V. Singhal, F. Somenzi, R.K. Brayton, Combinational verification for sequential circuits, DATE 1999.

[RH02] Rosenmann, A., Z. Hanna, Alignability equivalence of synchronous sequential circuits, HLDVT '02.

[SPAB01] Singhal, V., C. Pixley, A. Aziz, and R.K. Brayton, Theory of safe replacement for sequential circuits, IEEE Trans. on CAD of integrated circuits and systems, vol. 20, n.2, 2001.