Theoretical Framework for Compositional Sequential Hardware Equivalence Verification in Presence of Design Constraints

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

Abstract

We are interested in sequential hardware equivalence *(or* alignability equivalence*) verification of synchronous sequential circuits [Pix92]. To cope with large industrial designs, the circuits must be divided into smaller subcircuits and verified separately. Furthermore, in order to succeed in verifying the subcircuits, design constraints must be added to the subcircuits. These constraints mimic "essential" behavior of the subcircuit environment. In this work, we extend the classical alignability theory in the presence of design constraints, and prove a compositionality result allowing inferring alignability of the circuits from alignability of the subcircuits. As a result, we build a divide and conquer framework for alignability verification. This framework is successfully used on Intel designs.*

1. Introduction

We are interested in *sequential hardware equivalence* verification of circuits, also called *alignability* verification, introduced by Pixley [Pix92]. A very similar concept was studied in the ATPG context of *fault detection* by Pomeranz and Reddy [PR96]. We want to compare two gate level versions of a full chip design: the gate level models can be originated from RTL description, or extracted from a schematic net list.

Because formal equivalence verification tools normally do not scale to full chip verification, the gate level models are usually split into smaller subcircuits, and corresponding subcircuits are then verified. A question thus arises, whether alignability of subcircuits implies the alignability of the full chip models, and under what conditions. In other words, we want to answer the question: under what conditions is the alignability verification *compositional*?

The compositionality question is central to any concept of sequential equivalence, as one wants to know whether substituting a piece of a circuit by an equivalent (in some sense) piece yields an equivalent (in the same sense) circuit. Singhal et al. [SPAB01] pointed out that alignability is not compositional, and this observation led them actually to abandon alignability and to define a more restricted and complex concept of equivalence – *(delay) safe replaceability*, which is compositional: replacing, in a design D, a piece C with its (delay) safe replacement C_1 results in a design D_1 which is (delay) safe replacement of D. The idea of safe replaceability is that two equivalent circuits behave similarly *under any environment*.

Safe replaceability is safe to use (when it can be used!). However, we will see below (Section 3.1) that safe replaceability is not sufficient in the context of *divide and conquer framework* that we use in order to verify large circuits in small pieces. This is because, in our verification framework, we add *properties* to the subcircuits in order to *constrain* their behavior. The constraints mimic the "essential" behavior of the environment. We call these constraints *verification properties*, because they are introduced simply for the purpose of verification – in order to eliminate spurious counter-examples arising during verification of subcircuits (i.e., verification of subcircuits *without* the environment). Note that the criticism of alignability equivalence concept in [SPAB01] is based on the fact that alignable circuits do not behave in the same manner in an *arbitrary* environment. But in our approach, it is enough for corresponding subcircuits in the two circuits to behave same *under the imposed constraints*. Therefore, the criticism of alignability equivalence concept in [SPAB01] becomes *vacuous* for our verification framework.

Huang et al. [HCC01] point out that safe replaceability "…is more stringent (than alignability), and thus allows less flexibility for logic optimization. Furthermore, checking safe replaceability is difficult because every state of the transformed circuit needs to be examined. If the BDD representation can be constructed, this may be feasible. But for circuits beyond the capability of BDDs, this definition cannot be checked efficiently." Therefore [HCC01] introduced *3-valued safe replaceability* which is still stronger than alignability (for initializable circuits) but is easier to check, and furthermore it has excellent compositionality properties: (1) If in a large circuit every subcircuit is replaced by one of its respective 3-valued safe replacements, then the resulting circuit is 3-valued safe replacement of the original one; (2) Any initializing sequence of a circuit C initializes any of its 3-valued safe replacement circuits; and (3) Any 3-valued safe replacement of an initializable circuit C is alignable with C.

Still, just like safe replaceability, 3-valued safe replaceability is restrictive in our divide and conquer verification framework, because of the same reason (see Section 3.1).

In this work we will show that it is possible to recover the compositionality property of alignability under a certain reasonable condition, without resorting to a stronger concept of equivalence. The matter is complicated further by the fact that, under the presence of constraints, the entire alignability theory actually breaks down! Still, we show how to recover the alignability theory in the presence of verification properties, and we can derive alignability of full chip models from the alignability of the subcircuits under the assumption that the models are (weakly) synchronizable.

The paper is organized as follows. In the next section, we introduce main concepts of the sequential equivalence theory. While abstraction based compositional model checking methods are widely studied [CGP99], to the best of our knowledge, treatment of verification properties has not been addressed in the literature in the context of sequential equivalence verification. Therefore in Section 3, we provide a lengthy informal (but still pretty precise and detailed) discussion of the problems arising in the alignability theory in the presence of constraints. The problems and a solution proposed in Section 3 are formalized in Section 4. In Section 5 we prove our compositionality results. Experimental data is provided in Section 6. We conclude in Section 7.

2. Preliminaries

In order to be able to give an intuitive but precise description of the problems that we are going to solve, we start by introducing the basic concepts of sequential equivalence theory used in this work.

Definition [HS98]: A *Finite State Machine* (FSM) *M* is a 6-tuple $(S,\Sigma,\Gamma,\delta,\lambda,S_0)$ where

- S is a finite set of states (ranged over by s, s_1, s_2, …);
- Σ is a finite input alphabet (ranged over by a, …);
- Γ is a finite output alphabet (ranged over by e, …);
- δ: S x $\Sigma \rightarrow$ S is a state transition function;
- λ: S x $\Sigma \rightarrow \Gamma$ is an output function;
- $S_0 \subseteq S$ is the set of initial (i.e., start) states.

The FSMs that we will consider originate from synchronous gate-level sequential circuits, built from logic gates and state elements. For simplicity, we assume that the only state elements in the circuits are edge-triggered flip-flops with global clock (without set/reset or enable pins), which we will call *latches*. Our theory extends to other kinds of state elements as well.

Convention: In any FSM $M = (S,\Sigma,\Gamma,\delta,\lambda,S_0)$ that we will consider, a state $s \in S$ is represented as a tuple $(l_1,\ldots,l_e)$ of latch variables $L=\{l_1,\ldots,l_e\}$. More precisely, a state is given as a Boolean assignment to latch variables. Similarly, an input $a \in \Sigma$ is a tuple $(i_1,\ldots,i_h)$ of *input variables* $I=\{i_1,\ldots,i_h\}$, and is specified as a Boolean assignment to variables in I. And an output is a tuple of output variables $O=\{o_1,\ldots,o_j\}$. Further, S_0 for us is the set of all power-up states, that is, $S=S_0$ and we will omit mention of S_0 altogether. Therefore we may write an FSM M as $(S, L, \Sigma, I, \Gamma, O, \delta, \lambda)$. Further, we assume that δ is a collection of next state functions (NSFs) δ_i for each latch $l_i \in L$, and likewise for λ. Finally, for any input variable $i \in I$, $X^k(i)$ denotes the value of input i at time k (while $i = X^0(i)$ represents the value of the input i at time 0), and likewise for $o \in O$ and $l \in L$. Note that the values $X^k(l)$ are constrained with δ, and values $X^k(o)$ are constrained with λ, while input values $X^k(i)$ are not constrained in general.

Notation:

- Without loss of generality, we assume that every circuit has exactly one output. We consider circuits C, C_1, C_2, C_{xnor} with outputs o, o_1, o_2, and o_{xnor}, respectively, where $o_{xnor} = o_1$ xnor o_2 denotes the output of the *product* circuit $C_{xnor} = C_1 \times C_2$ (circuits C_1 and C_2 are assumed to be *compatible* – to have the same set of inputs and outputs) [HS98].
- π will denote a *binary input vector sequence* for C. Further, $o(s,\pi)$ denotes the value of o after simulating C with π, where C was initially at state s; $C(s,\pi)$ denotes the state into which π brings C from state s; and $C(S',\pi)=\{C(s, \pi) \mid s \in S'; S' \subseteq States(C)\}$.
- u denotes the *undefined* value; and $\perp$ denotes the *undefined state* of C, where all state elements have u values; $C(\perp,\pi)$ denotes the state of C after its 3-valued simulation with π, when C is initially at state $\perp$; and $o(\perp,\pi)$ denotes the corresponding value of o – it can be 1, 0 or u (see e.g. [HCC01]).

Definition:

- [CA89] An *initializing sequence* π_i brings C from *state* $\perp$ to a state s_i whose state elements have binary values 1 or 0 (which stand for True (T) and False (F), respectively). The state s_i is called an *initial* state.
- [Koh78] A *reset* or *synchronization* sequence π_r brings C from *any binary state* to a unique state s_r, called a *reset* or *synchronization* state.
- [Koh78] States $s_1 \in States(C_1)$ and $s_2 \in States(C_2)$ are *equivalent*, written $s_1 \simeq s_2$, if $\forall\pi$: $o_1(s_1, \pi) = o_2(s_2, \pi)$. State (s_1,s_2) is then an *equivalent state* of C_{xnor} .
- [PR96] A *weak synchronizing sequence* (*ws-sequence* for short) of a circuit C is an input vector sequence that brings C from any binary state to a subset of equivalent states $\{s_1,\ldots,s_m\}$, called *ws-states* of C.

An initializing sequence for C is its synchronizing sequence, and a synchronizing sequence is a ws-sequence. The converse to any of these statements is not true.

Definition [Pix92]:

- A binary input sequence π is an *aligning sequence for* a (binary) state (s_1,s_2) of C_{xnor} if it brings C_{xnor} from state (s_1,s_2) to an equivalent state.
- Circuits C_1 and C_2 are *alignable*, written $C_1 \simeq_{aln} C_2$, if every state of C_{xnor} has an aligning sequence (equivalently, there is a sequence, called *universal aligning sequence,* that aligns every state of C_{xnor}).

Note that initialization and (weak) synchronization refer to (i.e., require) *one* circuit only, while alignability refers to *two* circuits. When the two circuits are the same, one speaks of self-alignability. In particular, an input vector sequence for C is a ws-sequence for C iff it is a universal aligning sequence for C×C.

The alignability verification method reported in [Pix92] is BDD based [Bry86] – a BDD representing the set of all equivalent states of two circuits C_1 and C_2 is built first, and then it is checked whether any state can be synchronized into one of these states (again by using BDDs). A SAT based [DLL62] method is proposed in [RH02, KRSH04], where the circuits C_1 and C_2 are *first* weakly synchronized and *then* the resulting ws-states s_1 and s_2 of C_1 and C_2 are checked for state-equivalence. Following this method, below we distinguish between *weak-synchronization* and *state-equivalence* checking stages of alignability verification. This method is based on the following theorem:

Alignment Theorem [Pix92, HCC01, RH02]: Circuits C_1 and C_2 are alignable if and only if each circuit is weakly synchronizable and there is an equivalent pair $s_1 \simeq s_2$ of states in C_1 and C_2; the concatenation $\pi_1\pi_2$ of a ws-sequence π_1 of C_1 and a ws-sequence π_2 of C_2 is a ws-sequence for both C_1 and C_2, and is a universal aligning sequence for C_1 and C_2 (when the latter are alignable).

3. Alignability equivalence and constraints

As already mentioned in the introduction, in order to verify equivalence of large complex circuits, they are partitioned into smaller *subcircuits*. Development of sequential ATPG [HCC01] and SAT [BCC99] based verification methods made it possible to allow subcircuits with *sequential* logic – the subcircuits may contain internal latches. This however adds burden to equivalence verification check, not only in terms of complexity, but also in terms of semantic *correctness*. We explain the arising correctness issue on an example in Figure 1:

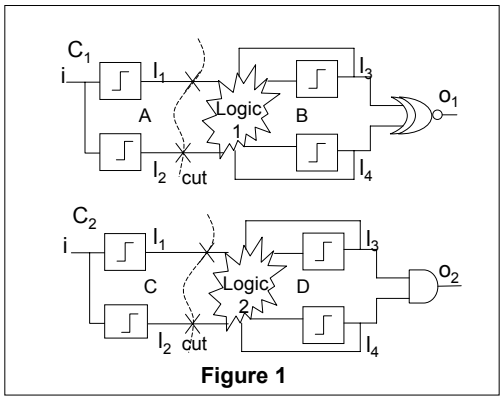

Figure 1

Suppose we want to split the circuits C_1 (*specification*, or spec) and C_2 (*implementation*, or imp) at latches l_1 and l_2. Call the subcircuits A, B, C and D, as shown on the figure. And assume B and D both represent the FSM in Figure 2 (we do not show the output values on the FSM – we assume that they differ only at state (l_3=0, l_4=0) – the only state where outputs o_1 and o_2 on Figure 1 differ):

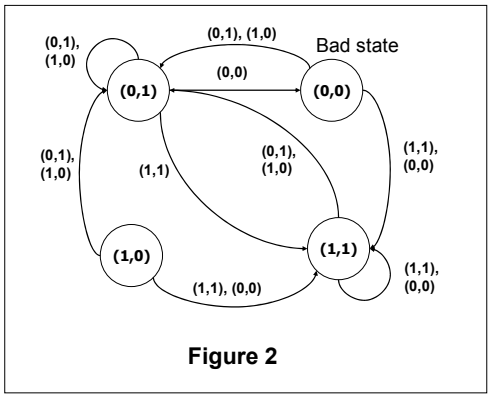

Figure 2

From the structure of subcircuits A and C, it is clear that, at any ws-state of C_1 and C_2, latches l_1 and l_2 have equal values. Furthermore, without the constraint $l_1 = l_2$ on the inputs of B and D, these subcircuits are synchronizable (e.g., input vector (l_1 =1,l_2 =1) synchronizes both B and D into state (l_3=1,l_4=1), vectors (1,0) and (0,1) synchronize B and D into state (0,1)), but B and D are *not alignable* (because the product B×D has a non-equivalent synchronization state ((0,0),(0,0)), and hence all its synchronization states are non-equivalent). On the other hand, C_1 and C_2 are synchronizable and *alignable* (any input sequence of length 2 or more aligns every state of $C_1 \times C_2$). That is, we must impose the constraint $l_1 = l_2$ on the inputs of B and D in order to be able to reduce alignability of C_1 and C_2 to alignability of the subcircuits.

3.1. Limitations of safe replaceability concepts in a divide and conquer paradigm

Recall that circuit C_2 is a *safe replacement* of C_1 if $\forall s_2 \in States(C_2)$ and $\forall \pi$, there is $s_1 \in States(C_1)$ such that,

from s_1 and s_2, C_1 and C_2 produce equal outputs *along* π [SPAB01]. And C_2 is a *3-valued safe replacement* of C_1 if $o_1(\perp,\pi) = o_2(\perp,\pi)$ when $o_1(\perp,\pi)$ is binary [HCC01].

Any synchronizable FSM has an initializable state encoding [CA89]. The shown state encoding of the FSM in Figure 2 is such that input (l_1=0, l_2=1) initializes B and D. Further, for weakly synchronizable circuits, safe replaceability implies alignability [SPAB01]. And for initializable circuits, 3-valued safe replaceability implies alignability [HCC01]. Thus D is not a safe replacement or a 3-valued safe replacement of B because B and D are not alignable. Thus we cannot use (3-valued) safe replaceability to reduce alignability verification C_1 and C_2 to alignability verification of their subcircuits.

Actually, even if we are interested in safe replaceability equivalence or 3-valued safe replaceability, we cannot split the circuits at latches l_1 and l_2, if we want to verify them in a divide and conquer fashion: C_2 is (3-valued) safe replacement of C_1, say when Logic 1 = Logic 2 in B and D, while D is not (3-valued) safe replacement of B. (Actually, the same is true for the *delay replaceability* equivalence [SPAB01]). To prove that C_2 is a safe replacement of C_1, note that for any input sequence π, at time 0 the outputs o_1 and o_2 are determined by values of latches l_3 and l_4 ($o_1 = l_3$ xnor l_4, $o_2 = l_3$ & l_4); at time 1, the outputs are determined by values of l_1 and l_2, and for the later times 2,3,…, the outputs are determined by π. Thus for any state s_2 of C_2 we can easily find state s_1 of C_1 such that o_1=o_2 at times 0 and 1. And for later times, o_1=o_2 follows from the fact that neither C_1 nor C_2 can be in the bad state (0,0). Finally, to prove C_2 is a 3-valued safe replacement of C_1, it is enough to note that o_1 and o_2 become binary at the same time (no matter what binary inputs are injected, starting from state $\perp$), and by a previous argument, o_1=o_2 at all initial states of C_1 and C_2.

We conclude that these replaceability concepts do not permit *convenient* partitioning of circuits, and *usage of verification properties is often inevitable* (this is supported by experimental data in Section 6). Taking into account subcircuit constraints in a safe replaceability check is against the nature of safe-replaceability (it was invented to make an equivalence concept *independent* of the environment). And it is not clear how 3-valued safe replaceability can be generalized in the presence of constraints, and whether its compositionality properties will be preserved. In the next subsection, we generalize alignability theory in the presence of constraints.

3.2. Methodology for handling properties in alignability equivalence

We call a constraint like $l_1 = l_2$ (in our example in Figure 1) a *verification property*, since it must be imposed on B and D in order to "pass" alignability verification on these subcircuits. Note that there may be other properties on the designs that are not related to the decomposition – we call them *correctness properties*. The correctness properties can be added to the circuit if the designer wants to verify the design correctness. Some of the correctness properties may be used as verification properties. In this paper we are not interested in verification of correctness properties, so we ignore correctness properties that are not needed or used as verification properties. Thus we assume we only have verification properties in the spec and or imp circuits.

Clearly, verification properties that will be used in proving alignability of the subcircuits must be verified as well – in the spirit of the *assume-guarantee paradigm* [Pnu85, CGP99]. Note that it is *not* correct to require their validity at all times or at all states: For example, the property $l_1 = l_2$ is valid at all *ws-states* of the circuits C_1 and C_2 and of subcircuits A and C (and at all post-ws times), but it is not valid at some legal *power-up states* (of the circuits or the subcircuits).

There are a number of possible intuitive semantics to verification properties (in the context of alignability verification), and here we give two of them that seemed most appealing to us:

1. Verification properties must be *valid at all post-ws times* (after a ws-sequence has been injected into the circuit, bringing the subcircuits into ws-states). Accordingly, during alignability verification of a subcircuit pair, the relevant verification properties on the subcircuits must be imposed *only after weak synchronization of the subcircuit*: that is, only at state-equivalence stage of alignability verification.
2. Verification properties must be *valid at all ws-states* of the subcircuits (no relation with *time* at all). Accordingly, during alignability verification of a subcircuit pair, the relevant verification properties must be imposed *at all ws-states of the subcircuit*: this includes both weak synchronization and state-equivalence stages of alignability verification.

We will now demonstrate that the first approach to the semantics of verification properties is problematic. If we allow inputs violating the assumption $l_1 = l_2$ *during weak synchronization* of the subcircuits B and D, we can arrive to any of the three *potential ws-states* of the subcircuits; under a potential ws-state of say B we mean all ws-states of B when there are *no constraints imposed* on it. After the weak synchronization ends, the FSM corresponding to B changes: for example, it is not possible to exit state (l_3=1, l_4=1) and arrive to the bad state (l_3=0, l_4=0) any longer because the constraint l_1=l_2 is "activated" *at post-ws times*. Thus if our ws-sequence brings the subcircuits B and D to state (l_3=1,l_4=1), the (state-equivalence part of the) alignability check will report "equal"; while if a ws-sequence brings B and D to state (l_3=0, l_4=0), the alignability check will report "differ"!

Recall from [PR96] that states s and s' in a circuit C are called *strongly connected* if there exist transitions from s to s' and from s' to s. Strong connectivity is an equivalence relation, and it splits the states of C into equivalence classes, called *strongly connected components* (SCCs). An SCC that is closed under state transition is called a *terminal* SCC [SPAB01]. The alignability theory is based on a fact that the set of all synchronization states forms a terminal SCC [PR96]. A similar statement is true, up to state-equivalence, for ws-states as well – all terminal SCCs containing a ws-state are equivalent sets of states [PR96]. The problem with the alignability theory that we are facing in the presence of verification properties is that, under the first approach, the above strong connectivity property fails for sets of potential synchronization or ws-states, as the underlying FSMs *during* and *after* the ws-stage are different.

Note that with the second approach to the semantics of verification properties, at the potential ws-states of the subcircuits B and D, we must restrict to input vectors $\{l_1=l_2=0, l_1=l_2=1\}$, therefore the subcircuits can be weakly synchronized into only one of the three potential ws-states – the state (1,1). For example, sequence $l_1=l_2=1$ is a ws-sequence for both B and D, bringing every state of each subcircuit into state (1,1). From this state, B and D will remain in equivalent states. Thus the alignability check will *always* report "equal".

While the second approach may seem to resolve the problem of the first one – the dependency of alignability verification outcome on a computed ws-state, it is not satisfactory for the following reason. A non-empty ws-sequence of a subcircuit – well, at least a (positive) number of its first transitions – should be applicable to *any* state of the subcircuit. This is different from what is stated in the second approach: in our example, the second approach permits a transition according to input vector ($l_1=1$, $l_2=0$) from state (0,0) but disallows transitions from the remaining states (with the same input vector).

To avoid this situation, we will *impose* verification properties on subcircuits at all states (and all times: including both the weak synchronization and the state-equivalence stages of alignability verification). The intuition justifying imposing the verification properties at all states of a subcircuit is as follows: suppose we are injecting an input vector sequence into the circuits C_1 and C_2 in Figure 1. *By the time* when these input values reach subcircuits B and D, the subcircuits A and C are synchronized, and the property $l_1=l_2$ is valid. And till that time, the input values at B and D can be ignored because they are generated by *random* latch values in A and C.

It is often convenient to impose needed constraints on internal latches rather than on inputs of the subcircuits. Since the verification properties replace environment behavior, *without loss of generality* we will assume that all latch properties have equivalent input properties. Therefore in the compositionality proofs in section 5, we restrict ourselves to input properties only.

There is also another reason that may cause the "instability" of the FSM corresponding to a subcircuit: in the presence of *sequential* properties, some of the arcs in the FSM graph may "disappear" and "re-appear" at different time phases. A simple sequential property causing instability of an FSM is a property expressing that the next value of an input i coincides its negation.

4. Conditional FSMs

We now formalize our observations by introducing *conditional* FSMs, and show that under our property treatment methodology and some additional conditions, the conditional FSMs induced by verification properties remain FSMs, thus alignability theory is valid for them.

Definition:

- A *(state transition) path* of an FSM $M = (S, \Sigma, \Gamma, \delta, \lambda)$ is a sequence p: $(s_0,a_0) \rightarrow \ldots \rightarrow (s_{n-1},a_{n-1}) \rightarrow s_n$, where for each $0<i<n+1$: $s_i = \delta(s_{i-1},a_{i-1})$. (When the length n = 0, p coincides with s_0.)
- A *conditional FSM* is a pair (M, Π), also written as a tuple $(S, \Sigma, \Gamma, \delta, \lambda, \Pi)$, where $M=(S, \Sigma, \Gamma, \delta, \lambda)$ is an FSM and Π is a subset of paths in M, called *admissible paths* of (M,Π),

Equivalently, a conditional FSM (M, Π) can be defined as tuple $(S, \Sigma, \Gamma, \delta^*, \lambda)$, where S, Σ, Γ and λ are as in an FSM, and δ^* is a total function δ^*: $\Pi x \Sigma \rightarrow S$, where Π is a subset of paths satisfying the following:

- If p: $(s_0,a_0) \rightarrow \ldots \rightarrow s_n \in \Pi$ and $a_n \in \Sigma$ is such that δ^* is defined on (p, a_n), then $(s_0,a_0) \rightarrow \ldots \rightarrow (s_n, a_n) \rightarrow s_{n+1} \in \Pi$, where $s_{n+1} = \delta^*(p, a_n)$.

Recall from the convention in Section 2 that an input $a \in \Sigma$ is a tuple of circuit inputs $I=\{i_1,\ldots,i_h\}$, represented as an assignment of 0s and 1s to $i_1,\ldots,i_h$; and an input $a_k \in \Sigma$ at time k is represented as an assignment to the inputs at time k, which are denoted by $X^k(i_1),\ldots,X^k(i_h)$. Similarly, a state $s \in S$ at time 0 is represented as an assignment to latches $L=\{l_1,\ldots,l_e\}$, and a state $s^k \in S$ at time k is represented by an assignment to variables $X^k(l_1),\ldots, X^k(l_e)$.

Definition: The assignment σ *corresponding* to path p: $(s_0,a_0) \rightarrow \ldots \rightarrow (s_{n-1},a_{n-1}) \rightarrow s_n$ in an FSM $M=(S, L, \Sigma, I, \Gamma, O, \delta, \lambda)$ is an assignment of 1 or 0 to variables $X^k(i_m)$ and $X^r(l_q)$, such that $\sigma(X^k(i_m))$ is the value of input $i_m \in I$ in a_k and $\sigma(X^r(l_q))$ is the value of latch $l_q \in L$ in s_r (k=0,…,n-1; r=0,…,n).

We need assignments to define whether a path satisfies a sequential property (defined below); the latter may refer to input and latch values at a number of time stages. For example, the transition (s,a) →s' where s = (l_1=0, l_2=1), a is an input (vector) i_1=1, and s' = (l_1=1, l_2=1), defines the assignment σ={l_1=0, l_2=1, i_1=1, X(l_1)=1, X(l_2)=1}. This transition satisfies property l_2=X(l_2), for example.

Definition: Let M =(S, L,Σ, I, Γ,O, δ, λ) be an FSM.

- A *property* P on M is a formula written using Boolean variables in I and L, the operator X, and Boolean connectives (& (and), + (or), ! (not), etc).
- P is a *combinational* property if it does not contain occurrences of X; else it is *sequential*. P is an *input* property if it does not contains variables in L.
- A path p: (s_0,a_0) → …→ (s_{k-1},a_{k-1}) → s_k in M *satisfies* P if for any tail p_j: (s_j,a_j) → … → s_k of p and the corresponding assignment σ_{pj} of p_j, the substitution instance σ_{pj}(P) of P is satisfyable. In this case we say that the path p is *admissible* for P in M, or simply, that path p is *P-admissible* in M.
- A property P on M *induces* a conditional FSM M_P=(S,Σ_P,Γ,δ, λ, Π) as follows:
  - The set of admissible paths Π of M_p coincides with the set of P-admissible paths of M.
  - $\Sigma_P \subseteq \Sigma$ is the set of all inputs in M that participate in at least one admissible path in M_P.

For example, the sequential property expressing that the next value of an input i is its negation, is written as X(i) = !i. To see whether a path p satisfies it, we need to check each sub-path of p of length 2, which is equivalent to checking validity of the property on each tail of p. Note that the state sets of M and M_P are the same – a property may rule out paths but not states, as an FSM can power up at any state of M. Power-up states may become "isolated" in M_P because of the constraints on transitions.

Definition: Let M =(S, Σ, Γ, δ, λ) be an FSM. An FSM M'=(S', Σ', Γ', δ', λ') is called a sub-FSM of M if S'⊆ S, Σ'⊆ Σ, Γ'⊆ Γ, and δ' and λ' are restrictions of δ and λ on S' × Σ'.

Theorem: Let M=(S, Σ, Γ, δ, λ) be an FSM and let P be a (satisfyable) combinational input property on M. Then there is a sub-FSM M' of M whose set of all paths coincides with the set of admissible paths of the conditional FSM M_P induced by M and P.

Proof: Let Σ'⊆ Σ be the set of all inputs satisfying P. Let M'=(S, Σ', Γ, δ', λ') be a sub-FSM of M. Then (s,a') → s' in M' iff (s,a') → s' is P-admissible in M_P. Thus the set of all paths of M' coincides with the set of P-admissible paths of M_P (since for a combinational input property P, P-admissibility of a path is determined by P-admissibility of its one-step transitions).

Property Convention: We restrict ourselves to verification properties P on an FSM M such that the induced conditional FSM M_P can be associated with a sub-FSM of M whose set of paths coincides with the admissible paths of M_P. We call such properties *stable*.

Stable properties exist by the above theorem. An example of a stable *sequential* property is i=X(i), on an input variable i∈I of a FSM M. Such a property causes "case splitting" on M, since i is either always 1 or always 0.

5. Decomposition and Compositionality

In this section, we establish our compositionality results. We start by defining our divide and conquer framework. Without loss of generality, we assume all subcircuits are sequential.

Definition (*stable decomposition*): Assume a circuit C is given together with a set of *cut-points* (including the outputs), and a set $\mathcal{VP}$ of verification properties, such that C can be partitioned into subcircuits A_i as follows:

1. For every cut-point there is exactly one subcircuit in C. And every subcircuit corresponds to one or more cut-points, which are the outputs of that subcircuit.
2. The subcircuits are non-overlapping, and expand maximally from the subcircuit outputs to the nearest cut points or primary inputs.
3. There is at most one property $P_i \in \mathcal{VP}$ on the inputs of a subcircuit A_i. The variables in P_i may be primary inputs. Else all of them are output nodes of another subcircuit, A_j, called a *supporting* subcircuit for A_i.

Then we call such a decomposition of C *stable* if every P_i is *stable* on (the FSM corresponding to) A_i; and P_i is valid *at all ws-states* of the relevant supporting subcircuit A_j, under the input constraint on A_j (if any).

The concept of stable decomposition can be relaxed by allowing more than one supporting subcircuits for any subcircuit; by allowing overlaps between the subcircuits under certain conditions, etc. This is necessary for the applicability of the framework in the practice.

Given a stable decomposition of a circuit C into subcircuits, by *compositionality of synchronization* we mean that synchronization of the subcircuits implies that of C; *compositionality of weak synchronization* and *initialization* are defined similarly.

It is clear from discussions in [SPAB01] that weak synchronization, synchronization, and initialization are not compositional. To demonstrate this, we build a non-weakly-synchronizable circuit and partition it into two initializable subcircuits. A simple FSM with two disjoint SCCs and two outputs l_1 and l_2 is given on Figure 3. Its

corresponding circuit is not weakly synchronizable but can be split into initializable subcircuits, as shown.

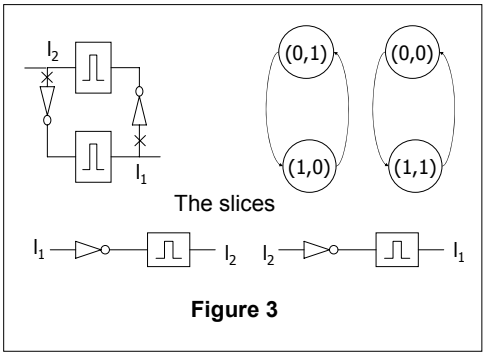

Figure 3

When we speak about a *decomposition* of two circuits C_1 and C_2 into subcircuits, we assume that for every cut point in C_1 there is at least one *corresponding*, also called *mapped*, cut point in C_2, and vice versa.

Definition: Suppose we are given a stable decomposition of C_1 and C_2 into subcircuits.

- By *compositionality of alignability* we mean that alignability of corresponding subcircuit pairs implies alignability of C_1 and C_2.
- By *weak compositionality of alignability* we mean that alignability of corresponding subcircuit pairs implies that of C_1 and C_2 under the assumption that C_1 and C_2 are weakly synchronizable.

To see that alignability is not compositional [SPAB01], assume $C_1 = C_2$ and let they coincide with the circuit in Figure 3. Then the subcircuits corresponding to l_1 and l_2 in C_1 and C_2 are alignable, while the circuits are not: for example, states (0,1) of C_1 and (1,1) of C_2 are not alignable. Therefore, C_1 and C_2 are not alignable.

Definition: States s_1 and s_2 of C_1 and C_2 are *consistent* with respect to a given stable decomposition of C_1 and C_2 if any corresponding subcircuit pair (A_1,A_2) in C_1 and C_2 is in an *equivalent ws-state* (s'_1,s'_2) (verification properties are used as constraints when computing the ws-states).

We will show that, for a stable decomposition of C_1 and C_2, consistency is a *transition invariant* implying state equivalence, and that consistent states exist if the subcircuits are alignable (that is, the consistency relation is a non-empty bisimulation [Par81] on the states of C_1 and C_2). Note that under the stability and consistency assumptions, inputs received by any subcircuit satisfy the input property on that subcircuit, thus any subcircuit behaves as the sub-FSM induced by the subcircuit FSM and the property on it. Therefore, in the proofs below, we will use term "constrained subcircuit" to refer to the constrained behavior of the subcircuit. Note also that the alignability theory is sound for constrained subcircuits since they are (sub) FSMs, thanks to the property convention.

Lemma 1: Assume we are given a stable decomposition of C_1 and C_2 such that the corresponding constrained subcircuits are alignable. Then the set of consistent states of C_1 and C_2 is non-empty and is closed under state transition.
Proof: We build consistent states s_1^* and s_2^* as follows: Assign such values to the latches in C_1 and C_2 that all corresponding constrained subcircuits will be set into *equivalent ws-states*. This is possible by the assumption that subcircuits are non-overlapping, and corresponding constrained subcircuits in C_1 and C_2 are alignable. The properties on subcircuit outputs are valid by the stability assumption. Now let s_1' and s_2' be any consistent states of C_1 and C_2, and let a be an input vector. Since corresponding subcircuits are in equivalent (ws-) states, their corresponding outputs have equal values (these outputs are mapped). Thus all subcircuits receive equal values at mapped inputs. Therefore, all corresponding subcircuits will remain in equivalent (ws-) states at the next time phase, and the properties on subcircuit outputs will remain valid. Thus, the states $C_1(s_1',a)$ and $C_2(s_2',a)$ remain consistent.

Lemma 2: Let s_1 and s_2 be states consistent with respect to a stable decomposition of C_1 and C_2. Then $s_1 \simeq s_2$.
Proof: By Lemma 1, $s'_1 = C_1(s_1,\pi)$ and $s'_2 = C(s_1,\pi)$ are consistent for any π. And the outputs of corresponding subcircuits, and in particular the outputs of C_1 and C_2, have the same values in s'_1 and s'_2. Hence $s_1 \simeq s_2$.

Theorem: Alignability is weakly compositional.
Proof: Assume we are given a stable decomposition of C_1 and C_2, and let corresponding constrained subcircuits be alignable. Let π be a ws-sequence for both C_1 and C_2. Let s_1^* and s_2^* be consistent states of C_1 and C_2, constructed as in the Lemma 1, and let $s_1=C_1(s_1^*,\pi)$ and $s_2= C(s_1^*,\pi)$. By Lemma 1, s_1 and s_2 are consistent, and by Lemma 2, $s_1 \simeq s_2$. Hence, by the Alignment Theorem, $C_1 \simeq_{aln} C_2$.

6. Experimental data

We have verified a large number of Intel designs using this verification framework, which is implemented in our sequential equivalence verification tool – *Seqver*. Hundreds of bugs were found, ranged from missing invertors to wrong coding, before a reboot sequence was provided by the designers. The two tables below contain data on the size of 3 verified circuits, as well as average size of the subcircuits. In some cases, subcircuits have substantial overlap (including common latches). The reported run times were measured on a 2.4GHz Linux machine with 2GB memory. We use SAT-based algorithms similar to those reported in [RH02, KH03, KRSH04] in the alignability and 3-valued safe replaceability checks. We do not have a safe replaceability verification algorithm in Seqver, and cannot report corresponding experimental data.

On design M_3, 129 subcircuits (with 1981 verification pairs) needed verification properties to pass alignability verification of decomposed spec / imp circuits; only 101 remaining pairs (which form two subcircuits) are 3-valued safe replaceable. On another spec / imp pair M_2, 2 chunks (16 verification pairs) needed verification properties to prove alignability, and 2666 pairs could be verified without any properties (both using alignability and 3-valued safe replaceability notions). On yet another spec / imp pair M_1, 207 chunks (1318 verification pairs) needed verification properties, and 1046 remaining pairs can be verified without using properties. All verification properties were found manually. Often same verification properties were used for several subcircuits.

Model	Gates	Latches	Outs	Inps	Cpu(sec)
M1	362604	12693	389	397	4700
M2	423070	23913	349	302	5722
M3	157110	22080	572	578	2478

Model	SubCkts	Avrg Inps	Avrg Gates	Avrg Latches
M1	524	650	11171	804
M2	402	102	824	33
M3	134	233	2874	187

7. Conclusions

We have developed a divide and conquer framework for verifying alignability equivalence of large circuits by dividing them into smaller subcircuits and verifying each subcircuit separately using properties (constraints) that abstract behavior of the subcircuit environment. And we have proven a suitable compositionality result allowing inferring alignability of the circuits from alignability of the constrained subcircuits. To enable this, we have extended the classical alignability theory in the presence of design constraints.

Note that our compositionality result ensures that, under the condition that the specification and implementation circuits are weakly synchronizable, there is no need to worry about cyclic dependences in the assume-guarantee reasoning [Pnu85, CGP99] – it is enough we prove output properties on subcircuits by using their input properties. Note also that weak synchronization is a reasonable requirement from any reboot sequence, as otherwise the post-reboot behavior of the circuit will not be deterministic – it will depend on the specific (random) power-up state.

Acknowledgements: We thank S. Goldenberg and A. Jas, who were the first to encounter the inconsistency of alignability verification in Seqver, in the presence of design constraints. And we thank the referees for constructive feedback.

References

[BCC99] Biere, A., A. Cimatti, and E. Clarke, Symbolic model checking without BDDs, Tools and Algorithms for the Construction and Analysis of Systems, 1999.

[Bry86] Bryant R.E., Graph-based algorithms for Boolean function manipulation, IEEE Trans. Computers, C-35(8), 1986.

[CA89] Cheng K.-T. and D. Agrawal, State assignment for initializable synthesis, ICCAD'89, 1989.

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

[DLL62] Davis, M., G. Logemann and D. Loveland, A machine program for theorem-proving, Communications of ACM 5(7), 1962.

[HS98] Hachtel G.D. and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publishers, 1998.

[HCC01] Huang, S.-Y., K.-T., Cheng, and K.-C. Chen, Verifying sequential equivalence using ATPG techniques, ACM Trans. on Design Automation of Electronic Systems, 2001.

[KRSH04] Kaiss, D., A. Rosenmann, M. Skaba and Z. Hanna, A formal method and apparatus for an automatic synchronization of finite state machines for sequential verification of chip design, US Patent application, 2004.

[KH03] Khasidashvili, Z., Z. Hanna, SAT-Based methods for sequential hardware equivalence verification without synchronization, BMC'03, ENTCS 89 (4), 2003.

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

[Par81] Park, D. Concurrency and automata on infinite sequences, 5th GI-Conference on Theoretical Computer Science, Springer LNCS, vol. 104, 1981.

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

[Pnu85] Pnueli, A., In transition from global to modular temporal reasoning about programs, In: Logics and Models of Concurrent Systems, Springer LNCS, vol. F-13 of NATO ASI series, 1985.

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

[RH02] Rosenmann, A. and Z. Hanna, Alignability equivalence of synchronous sequential circuits, HLDVT'02, 2002.

[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.