Solving the Latch Mapping Problem in an Industrial Setting

Kelvin Ng
Dept. of Computer Science
University of British Columbia
Vancouver, BC
kng@cs.ubc.ca

Mukul R Prasad   Rajarshi Mukherjee   Jawahar Jain
Advanced CAD Research
Fujitsu Laboratories of America
Sunnyvale, CA
{mukul, rmukherj, jawahar}@fla.fujitsu.com

ABSTRACT

We describe a complete method for the latch mapping problem that is based on the efficient integration of previously proposed techniques for latch mapping as well as novel optimizations for further improvement. The highlights of the proposed approach include a new method of integrating complete methods and incomplete methods for latch mapping, the use of incremental reasoning to optimize the overall algorithm and the use of a conventional combinational equivalence checking tool as the core engine. Experiments confirm that the proposed method retains much of the efficiency and capacity of incomplete methods while providing the completeness of complete methods and derives significant performance improvements from the proposed optimizations.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: formal verification

General Terms

Algorithms, Verification

Keywords

Latch mapping, combinational equivalence checking

1. INTRODUCTION

Combinational equivalence checking (CEC) is a mature and practical technology [2, 7, 8, 9, 11] that is commonly used in current verification methodology. *Latch mapping* is used to transform the problem of checking sequential equivalence into a combinational equivalence checking problem. Recent work on latch mapping [1, 3, 4, 5, 14] has offered some promising solutions. However, as combinational equivalence checking technology becomes more pervasive in commercial verification flows, there is a need for more powerful and efficient latch mappers to complement them.

Methods for latch mapping can be divided into incomplete methods and complete methods. Incomplete methods use heuristics to group promising matches without providing any guarantees on the correctness or completeness of the matching. They can be function-based or non-function based. Non-function based incomplete methods [4] use name or structural comparisons to group latches. Function-based methods such as those proposed in [1, 4] use random simulation [4] or ATPG-based search [1] to generate inequivalence information, which is used to group latches. Complete methods, on the other hand, are guaranteed to produce a latch mapping if one exists, given sufficient computational resources. Almost all complete methods for latch mapping, proposed in the literature [3, 5, 14], employ a functional fixed-point iteration to refine the universe of all latches into a provably correct and complete grouping. Also, related work on general sequential equivalence checking [6, 12, 13] uses techniques that may be applicable to latch mapping.

In this paper we describe a methodology for the latch mapping problem that is based on the efficient integration of previously proposed techniques as well as novel optimizations to further improve these techniques. This method has been deployed and extensively tested in an industrial setting and has proved to be quite successful. Our methodology for latch mapping is premised on the following observations, based in part on the experiences of other researchers who have worked on this topic.

Incomplete methods can usually match a large percentage of the latches in practical designs, efficiently and correctly [1]. Therefore, any efficient methodology for latch mapping needs to make use of such methods. However, complete methods are the only option for difficult instances of latch mapping, such as those produced by heavy optimization of a design by a tool which does not preserve name correspondences [3] or cases where one of the circuits under comparison has significant redundancies and/or replicated portions of logic which the second circuit does not have. This could lead to matchings involving groups of more than two latches or complex intra-circuit groupings. Incomplete methods are usually designed to detect simple, pairwise inter-circuit matchings. Thus, an effective method for latch mapping should be one which combines incomplete and complete methods together in a symbiotic manner. Another benefit of using a complete latch mapping method is that if the given problem is *not* a latch mapping problem (and hence not solvable through CEC) the complete method could return a partial set of *true* latch equivalences which can subsequently be utilized by a more general sequential equivalence verifier.

In the light of the above, the major contributions of this paper can be summarized as follows:

- We formulate and discuss the problem of combining incomplete and complete methods for latch mapping and propose a comprehensive and efficient solution to this problem.
- We use an industrial-strength, mixed-engine combinational equivalence checker [9], to implement a complete algorithm for latch

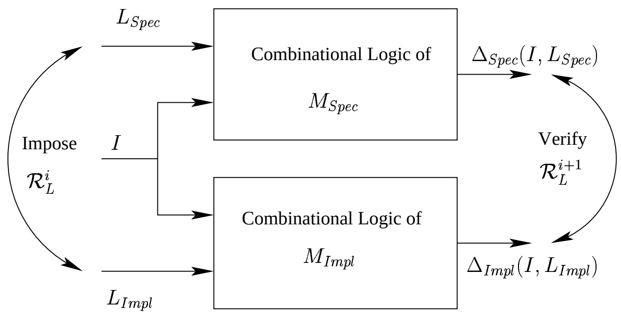

Figure 1: Circuit Model $\mathcal{P}$ for Refinement of $\mathcal{R}_L^i$

mapping, similar to the fixed-point iteration reported in [3, 14]. We note that previous approaches have either reported results using a single engine (BDDs in [14] and ATPG in [1]) or not given details of the engine used [3]. Thus, this is the first reported application and experimental evaluation of a proven combinational equivalence checking technology for latch mapping.

• We propose two novel techniques to substantially improve the performance of the above complete method. The key idea is to optimize the use of the CEC tool within the latch mapping framework, rather than using it as a black box.

The rest of the paper is organized as follows. The next section reviews some notation and basic algorithms used in the rest of the exposition. In Section 3 we describe the details of our latch mapping methodology. Section 4 presents an experimental validation of our approach. We present our conclusions in Section 5.

2. PRELIMINARIES

Sequential circuits can be represented as finite state machines (FSMs). An FSM, M is a 6-tuple, $M = (I, O, L, S_0, \Delta, \lambda)$, where $I = (x_1, x_2, \ldots, x_m)$ is an ordered set of inputs, $O = (z_1, z_2, \ldots, z_p)$ is an ordered set of outputs, L is an ordered set of state variables (denoting latches), $S_0 \subseteq \mathcal{B}^{|L|}$ is a non-empty set of initial states, $\Delta : \mathcal{B}^{|L|} \times \mathcal{B}^m \rightarrow \mathcal{B}^{|L|}$ is the next-state function and $\lambda : \mathcal{B}^{|L|} \times \mathcal{B}^m \rightarrow \mathcal{B}^p$ is the output function. A state S of M is a Boolean valuation to the state variables L. In the sequel, the present state and next state variables corresponding to a latch l are denoted l and δ_l respectively.

2.1 The Latch Mapping Problem

Let the two sequential circuits being checked for equivalence be represented by FSMs M_{Spec} (specification FSM) and M_{Impl} (implementation FSM). Further, to simplify the exposition we assume the circuits have a single clock, the same inputs and outputs, and exactly one initial state, denoted $S_{0,Spec}$ and $S_{0,Impl}$ respectively. We note that the methods presented in this paper can be extended for the case of multiple initial states using the treatment in [3]. Thus, $M_{Spec} = (I, O, L_{Spec}, S_{0,Spec}, \Delta_{Spec}, \lambda_{Spec})$ and $M_{Impl} = (I, O, L_{Impl}, S_{0,Impl}, \Delta_{Impl}, \lambda_{Impl})$. Let $L = L_{Spec} \cup L_{Impl}$ denote the combined state variables of M_{Spec} and M_{Impl}. Further if S_{Spec} and S_{Impl} are states in the state-spaces of M_{Spec} and M_{Impl} respectively, *i.e.* $S_{Spec} \in \mathcal{B}^{|L_{Spec}|}$ and $S_{Impl} \in \mathcal{B}^{|L_{Impl}|}$ we use $S = S_{Spec} \cup S_{Impl}$ to denote the combined state. Similarly, the combined transition function Δ is obtained by combining Δ_{Spec} and Δ_{Impl} and the combined initial state $S_0 = S_{0,Spec} \cup S_{0,Impl}$.

The latch mapping problem is posed on the combined set of latch variables $L = L_{Spec} \cup L_{Impl}$ and on the combined states in the state-space of these variables. In the sequel, we will drop the *combined* prefix when referring to these latch variables and states, stating otherwise, explicitly, whenever required. A latch mapping is denoted by a *latch correspondence relation*, $\mathcal{R}_L$ which is an equivalence relation on the latches, L. Thus, $\mathcal{R}_L : L \times L \rightarrow \mathcal{B}$. Further, the *variable correspondence condition* [14], $\mathcal{F}_L : \mathcal{B}^{|L|} \rightarrow \mathcal{B}$ is a predicate that defines whether a state S conforms to $\mathcal{R}_L$, *i.e.*, whether equivalent latch variables assume identical values in S:

$$\mathcal{F}_L(S) \Leftrightarrow \forall l_1, l_2(\mathcal{R}_L(l_1, l_2) \Rightarrow S(l_1) = S(l_2)) \qquad (1)$$

The relation $\mathcal{R}_L$ is designed to group together latches that are equivalent, under some notion of sequential equivalence.

2.2 Van Eijk's Algorithm for Latch Mapping

Van Eijk [14] proposed the following definition of $\mathcal{R}_L$, based on a sufficient (but not necessary) condition for latch equivalence.

DEFINITION 2.1 (LATCH CORRESPONDENCE RELATION). *A latch correspondence relation is an equivalence relation $\mathcal{R}_L : L \times L \rightarrow \mathcal{B}$ which satisfies the following conditions:*

- *It is true in the initial state, S_0 of the combined machine: $\mathcal{F}_L(S_0) = 1$,*
- *It is invariant under the next state function: $\forall S \in \mathcal{B}^{|L|}, X \in \mathcal{B}^m : \mathcal{R}_L(S) \Rightarrow \mathcal{R}_L(\Delta(S, X))$*

Further, van Eijk noted that there may exist several correct latch correspondence relations for a given FSM and there exists a *unique maximum latch correspondence relation* $\mathcal{R}_L^{max}$, which is the *fixed-point* computed by the following iterative procedure [14].

Algorithm 1 van Eijk's Algorithm

```
1: Compute the first approximation $\mathcal{R}_L^0$ of $\mathcal{R}_L$ as: $\mathcal{R}_L^0(l_i, l_j) \Leftrightarrow (S_0(l_i) = S_0(l_j))$
2: Given $\mathcal{R}_L^i$, refine it to compute $\mathcal{R}_L^{i+1}$ as: $\mathcal{R}_L^{i+1}(l_i, l_j) \Leftrightarrow \mathcal{R}_L^i(l_i, l_j) \wedge \forall S \in \mathcal{B}^{|L|}, X \in \mathcal{B}^m : \mathcal{F}_L^i(S) \Rightarrow \mathcal{F}_L^{i+1}(\Delta(S, X))$
3: If $\mathcal{R}_L^{i+1} = \mathcal{R}_L^i$ stop. $\mathcal{R}_L^{max} = \mathcal{R}_L^i$.
```

Since this is a complete algorithm, the maximum latch correspondence relation $\mathcal{R}_L^{max}$ is a *complete and correct solution* to the latch mapping problem on M_{Spec} and M_{Impl}.

In practise [14] this algorithm is implemented on the circuit model $\mathcal{P}$ shown in Figure 1. The relation $\mathcal{R}_L^i$ is represented as a set of *equivalence classes* over the variables L, which are refined in each iteration, using Step 2 of the algorithm. The refined equivalence relation $\mathcal{R}_L^{i+1}$ is computed through a series of equivalence checks on $\mathcal{P}$. In iteration $i + 1$ equivalences in $\mathcal{R}_L^i$ are imposed on the present-state latch variables L and the same equivalences then verified on the next-state latch variables Δ, under the above constraints. The equivalences that hold, form the refined relation $\mathcal{R}_L^{i+1}$. Note that the primary outputs O of M_{Spec} and M_{Impl} are ignored when solving the latch mapping problem.

As mentioned in [3, 14], the overall efficiency of the procedure can be significantly improved by refining $\mathcal{R}_L^0$ through random simulation, before entering the fixed-point iteration.

3. PROPOSED METHOD

We believe that a latch mapping methodology needs to employ a combination of incomplete and complete methods in order to provide a comprehensive but efficient solution to industrial latch mapping problems. Our method is based on such a combination. Our

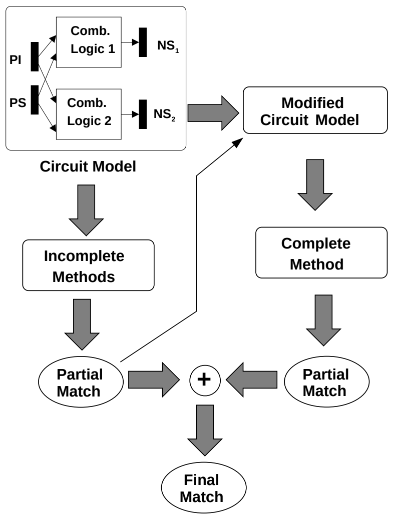

Figure 2: A Hybrid Flow for Latch Mapping

tool includes a set of efficient structural and functional, incomplete methods for latch mapping. We believe these methods are competitive with the state-of-the-art in incomplete methods. However, these methods are not the contribution of this paper and hence will not be discussed here. Our contributions are 1.) a novel method of combining incomplete and complete methods for latch mapping (discussed in Section 3.1) and 2.) an optimized complete method for latch mapping (described in Section 3.2). The complete method is a variant of van Eijk's algorithm [14], implemented using powerful engines and further enhanced with novel pruning techniques.

3.1 A Hybrid Method for Latch Mapping

Previous works on latch mapping have proposed either complete methods [3, 5, 14] or incomplete methods [1, 4] for the problem. In the following we formulate and discuss the problem of combining these two kinds of approaches and propose a comprehensive and efficient solution. To the best of our knowledge, this is the first reported work to explicitly address this aspect of latch mapping.

Figure 2 shows a possible flow in which incomplete and complete methods could be used together. If $\mathcal{R}_1$ and $\mathcal{R}_2$ are the *partial matches* produced by incomplete and complete methods, respectively, the final match can be computed as $\langle \mathcal{R}_1 \vee \mathcal{R}_2 \rangle$, where $\langle \mathcal{R} \rangle$ denotes the transitive closure of relation $\mathcal{R}$. The key element is in generating the *modified circuit model* (denoted $\mathcal{P}^*$) to be supplied as input to the complete method. More concretely, this problem can be formulated as follows. Suppose $\mathcal{R}_{L_{sub}}$ is a partial latch mapping given to us, *i.e.* $\mathcal{R}_{L_{sub}}$ is an equivalence relation over the set of latches $L_{sub} \subseteq L$. In practise, the partial match may be generated by a combination of incomplete methods and constraints specified by the designer. Further, we assume that $\mathcal{R}_{L_{sub}}$ is a *true* relation, *i.e.* $\mathcal{R}_{L_{sub}} \subseteq \mathcal{R}_L^{max}$. In other words, the equivalences specified in $\mathcal{R}_{L_{sub}}$ are true in any correct and complete solution to the given latch mapping problem. Our objective is to use a complete method to find a *complete* latch correspondence, taking advantage of the information provided by $\mathcal{R}_{L_{sub}}$.

A simple method of producing the modified circuit model, $\mathcal{P}^*$ from $\mathcal{P}$ and $\mathcal{R}_{L_{sub}}$ is the following. The equivalences represented by $\mathcal{R}_{L_{sub}}$ are imposed on the present-state variables in $\mathcal{P}$ by merging all specified equivalent latches. Further, the next-state functions of the latches L_{sub} and the cones of logic exclusively feeding them are removed from $\mathcal{P}$. The following example illustrates the problem with this approach.

EXAMPLE 3.1. *Suppose the exact latch correspondence solution has latches l_i and l_j from M_{Spec} mapped together with latch l_k from M_{Impl}. Further, suppose that the supplied partial match $\mathcal{R}_{L_{sub}}$ is such that $(l_i, l_k) \in \mathcal{R}_{L_{sub}}$ but $l_j \notin L_{sub}$.*

Then, the $\mathcal{P}^*$ produced as above will not have latches l_i and l_k. Hence the complete method will not be able to match latch l_j to anything. Consequently, the overall solution (final match) will be missing part of the equivalence $l_1 \equiv l_j \equiv l_k$ and therefore be incomplete.

Our method is a modification of the above solution. The constraints $\mathcal{R}_{L_{sub}}$ are imposed on the present-state variables in $\mathcal{P}$, as above. Let $\mathcal{R}_{L_{sub}}$ be comprised of q equivalence classes $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_q$. For each equivalence class $\mathcal{C}_i$, one of the constituent latches l^i_{rep} is chosen as its representative. The next state functions of all latches in the set $L_{sub} - \bigcup_{i=1}^{q} l^i_{rep}$ as well as cones of logic exclusively feeding them are removed from $\mathcal{P}$. This gives us the reduced model $\mathcal{P}^*$. Let $\mathcal{R}^{max}_{rep}$ be the final relation computed by the complete procedure run on $\mathcal{P}^*$. Then our claim is that $\langle \mathcal{R}_{L_{sub}} \vee \mathcal{R}^{max}_{rep} \rangle$ is precisely the maximum latch correspondence $\mathcal{R}_L^{max}$ that would be computed by van Eijk's algorithm executed on the original problem, $\mathcal{P}$.

THEOREM 3.1. *Given a latch mapping problem on a circuit model $\mathcal{P}$ and a partial, true latch equivalence relation $\mathcal{R}_{L_{sub}}$, if $\mathcal{R}^{max}_{rep}$ is the latch equivalence relation computed using the above construction, then $\langle \mathcal{R}_{L_{sub}} \vee \mathcal{R}^{max}_{rep} \rangle = \mathcal{R}_L^{max}$.*

Thus, the above combined method is provably a complete method for latch mapping. The proposed construction allows van Eijk's algorithm to efficiently and correctly use the information contained in $\mathcal{R}_{L_{sub}}$. The problem solved by van Eijk's algorithm is significantly smaller than the initial problem. Further, *none* of the relationships specified in $\mathcal{R}_{L_{sub}}$ are re-verified during the run of van Eijk's algorithm.

3.2 Efficient Implementation of Van Eijk's Algorithm

As described in Section 2.2, van Eijk's algorithm involves a fixed-point iteration. In each iteration latch equivalences of the current latch mapping relation are imposed on the inputs of the circuit model $\mathcal{P}$ and the same equivalences verified at the outputs. This computation can be easily reduced to a combinational equivalence checking problem. Additionally, the two circuits M_{Spec} and M_{Impl} often exhibit a lot of structural similarity. Therefore, we have chosen to implement this computation by a state-of-the-art, industrial strength combinational equivalence checking (CEC) tool [9]. This tool implements a well-tuned combination of random simulation, BDDs, ATPG and structural pruning techniques and compares very favorably to the state-of-the-art in combinational equivalence checking.

Like most industrial combinational equivalence checkers today, our CEC tool exploits the structural similarity between the two circuits under comparison. The general approach is to partition the overall equivalence check into a set of smaller equivalence checks. A set of candidate equivalences (referred to as *potentially equivalent nodes (PENs)* in the sequel) is initially composed from internal nodes of the two circuits. Then the algorithm sweeps from the primary inputs to outputs successively resolving these candidate equivalences, taking advantage of internal equivalences proved thus far, till the output equivalences are resolved.

We note that previous implementations of van Eijk's algorithm and related work have been reported using a single engine [1, 14] or a dedicated combination of engines [3, 6]. By using an off-the-shelf CEC tool we hope to seamlessly leverage the powerful technology

Algorithm 2 Update changed flag and affected pointer in Iteration $i, i \geq 1$

```
for all $w \in W$ and $l \in L$ do
    Reset changed flag
    Clear affected pointer
end for
for all $l \in L : Rep(l^{i-1}) = l$ do
    if $i = 1$ then
        affected$(l) \leftarrow l$
    else if Equiv_Class$(\delta_l, \mathcal{R}_L^{i-1}) \neq$ Equiv_Class$(\delta_l, \mathcal{R}_L^{i-2})$ then
        affected$(l) \leftarrow l$
    end if
end for
for all $l \in L : (Rep(l^{i-1}) = l) \wedge (affected(l) \neq$ NULL$)$ do
    for all $w \in$ TFO$(l)$ do
        if affected$(w) \neq$ NULL then
            changed$(w) \leftarrow 1$
        else
            affected$(w) \leftarrow l$
        end if
    end for
end for
```

built into current CEC tools and also bypass the development effort required to build a dedicated tool of comparable performance.

In our CEC tool PENs are maintained and manipulated as follows. Initially, PENs are composed by simulating the circuit with random vectors. Internal nodes with the same simulation signature are grouped together. Each such group is a PEN set. The interpretation here is that nodes in the same group are *potentially equivalent*, hence PENs. Nodes falling in different groups cannot be combinationally equivalent. These classes are refined by successively validating PEN pairs from each group, till finally all PENs have been resolved (*i.e.* either proved equivalent or proved inequivalent and split apart by a witness vector). When used in van Eijk's algorithm, the PEN sets are populated by the internal nodes $W = \{w_1, w_2, \ldots, w_n\}$ of the circuit model $\mathcal{P}$. In each iteration i, the latch inputs L are constrained by the corresponding relation $\mathcal{R}_L^{i-1}$. We will denote this instance of $\mathcal{P}$ as $\mathcal{P}^i$. Further, if w is an arbitrary internal node in $\mathcal{P}$, w^i denotes its instance in $\mathcal{P}^i$. Given an internal node w in $\mathcal{P}$, TFI(w) and TFO(w) denote the set of nodes in the transitive fanin and transitive fanout of w respectively. In practise, the application of the constraints $\mathcal{R}_L^i$ on the present-state latch inputs L is implemented as follows. As mentioned in Section 2.1, $\mathcal{R}_L^i$ is represented as a set of equivalence classes. For each such class $\mathcal{C}_j$, one latch l_{rep}^j is chosen as the representative and all fanouts of other latches in $\mathcal{C}_j$ are re-routed from l_{rep}^j.

In the sequel we describe two novel optimizations to substantially improve the performance of the CEC tool, for latch mapping. Although, these optimizations are described in the context of our CEC tool, they are equally applicable to other CEC tools.

3.2.1 Incremental Reasoning

As per the above description, in each iteration of van Eijk's algorithm, the CEC tool creates and verifies a number of internal node PENs. The notion of incremental reasoning seeks to reduce some of this effort. It is motivated by the following observations we made while using van Eijk's algorithm in an industrial setting:

• For complex, industrial circuits, van Eijk's algorithm usually takes several iterations to converge (typically $2 - 5$).

• Usually, the relation $\mathcal{R}_L^i$ changes only gradually from iteration i to $i + 1$.

From the above it is plausible that a large fraction of the PEN equivalences (and inequivalences) that were true in iteration i would also hold in iteration $i + 1$. Therefore, our proposal is to efficiently extract PEN equivalences (and inequivalences) that remain *invariant* under the refinement of $\mathcal{R}_L^{i-1}$ to $\mathcal{R}_L^i$, performed in iteration i. This information is then supplied as axioms to the CEC tool in iteration $i + 1$, which can bypass the verification of these PENs in this iteration. In this manner, the CEC tool only re-verifies information that has potentially changed since the last iteration.

Our method has two aspects, 1.) Using PEN inequivalence information, 2.) Using PEN equivalence information.

THEOREM 3.2. *Given internal nodes w_1 and w_2 in model $\mathcal{P}$ and some $i \geq 0$, $w_1^i \neq w_2^i \Rightarrow w_1^j \neq w_2^j$ for any $j \geq i$.*

Using PEN Inequivalence Information: Using Theorem 3.2, in our implementation, the internal node PEN sets (for the CEC tool) are not built from scratch but rather the final internal PEN sets at the end of iteration i are used as the starting point for iteration $i+1$. This simple observation allows the computation of iteration $i + 1$ to seamlessly leverage the entire effort spent in iterations 1 through i in detecting *inequivalences* between internal PENs.

LEMMA 3.1. *Given internal nodes w_1, w_2 in $\mathcal{P}$, if for some iteration i, $w_1^i = w_2^i$ and $\forall l_1, l_2 \in (L \cap TFI(w_1))\ R_L^{i+1}(l_1, l_2) \Leftrightarrow R_L^i(l_1, l_2)$ and $\forall l_1, l_2 \in (L \cap TFI(w_2))\ R_L^{i+1}(l_1, l_2) \Leftrightarrow R_L^i(l_1, l_2)$ then $w_1^{i+1} = w_2^{i+1}$.*

Using PEN Equivalence Information: Lemma 3.1 provides a way of leveraging equivalences proved in previous iterations to bypass some such checks in the current iteration. However, the conditions stated in Lemma 3.1 are computationally difficult to check. Therefore, we have implemented a safe approximation of them which can be represented, maintained and checked by simple structural operations on the circuit. The implementation requires two data entities to be maintained for each node w in $\mathcal{P}$, namely a single-bit Boolean flag, *changed*(w) and a node pointer *affected*(w). At the beginning of each iteration i ($i \geq 1$) of van Eijk's algorithm, these are updated as per Algorithm 2.

THEOREM 3.3. *Given internal nodes w_1 and w_2 in $\mathcal{P}$, given some iteration i, if $w_1^{i-1} = w_2^{i-1}$ and after updating changed and affected data as per Algorithm 2 if:*

1. *affected$(w_1^i) = NULL \quad \wedge \quad$ changed$(w_2^i) = 0 \quad$ or*
2. *affected$(w_2^i) = NULL \quad \wedge \quad$ changed$(w_1^i) = 0 \quad$ or*
3. *changed$(w_1^i) = 0 \quad \wedge \quad$ changed$(w_2^i) = 0 \quad \wedge$ (affected$(w_1^i) =$ affected$(w_2^i) \neq NULL$)*

then $w_1^i = w_2^i$.

Theorem 3.3 and Algorithm 2 form the basis for our method to bypass certain PEN equivalence checks during a run of our CEC tool in van Eijk's algorithm. Algorithm 3 describes our incremental version of van Eijk's algorithm.

3.2.2 PEN Rejection during CEC

We have developed the following simple, but useful optimization to further enhance the performance of the CEC tool during van Eijk's algorithm. This is applicable when van Eijk's algorithm is used in a hybrid flow as in Section 3.1.

As described in Section 3.1 the latches (L_{sub}) comprising the partial match $\mathcal{R}_{L_{sub}}$ are condensed into a set of *representatives* $L_{rep} = \{l_{rep}^1, l_{rep}^2, \ldots, l_{rep}^q\}$ and supplied along with the latches $L - L_{sub}$ to van Eijk's method. For a given iteration i of van

Algorithm 3 Incremental Version of van Eijk's Algorithm

```
1. Compute $\mathcal{R}_L^0$ using Definition 2.1 as: $\mathcal{R}_L^0(l_i, l_j) \Leftrightarrow (S_0(l_i) = S_0(l_j))$. Refine this with random vector simulation on circuit model $\mathcal{P}$.
2. For each iteration $i$, (initialize $i$ to 1):
    i. Construct circuit model $\mathcal{P}^i$ by applying conditions $\mathcal{R}_L^{i-1}$ on PS latch variables.
    ii. If $i = 1$, construct initial internal PEN sets $\mathcal{Q}_{init}^1$, through random vector simulation on $\mathcal{P}^1$ else $\mathcal{Q}_{init}^i = \mathcal{Q}_{final}^{i-1}$
    iii. Update affected and changed fields using Algorithm 2.
    iv. Run CEC tool on model $\mathcal{P}^i$ to resolve internal PEN sets $\mathcal{Q}_{init}^i$ and refine $\mathcal{R}_L^{i-1}$.
    v. For each internal PEN pair $w_1, w_2 \in \mathcal{Q}^i$, if $w_1$ and $w_2$ satisfy the conditions of Theorem 3.3 infer $w_1 = w_2$ else check $w_1 \overset{?}{=} w_2$. If $w_1 \neq w_2$ simulate $\mathcal{P}^i$ with the obtained distinguishing vector (to refine $\mathcal{Q}^i$ and $\mathcal{R}_L^{i-1}$). The final PEN sets obtained after this step constitute the relation $\mathcal{Q}_{final}^i$
    vi. Verify $\mathcal{R}_L^{i-1}$ at the NS latch variables, checking equivalences pair by pair, as with internal PENs in the last step. The relation obtained after these checks is $\mathcal{R}_L^i$.
3. If $\mathcal{R}_L^i = \mathcal{R}_L^{i-1}$ stop and return $\mathcal{R}_L^{max} = \mathcal{R}_L^i$ else iterate Step 2 with $i \leftarrow i + 1$.
```

Eijk's algorithm, and an arbitrary equivalence class, $\mathcal{C}$ of $\mathcal{R}_{Rep}^i$, the optimization is as follows:
Optimization: *If $\mathcal{C} \subseteq L_{rep}$, i.e. all the latches of this class are representative latches, discard $\mathcal{C}$ remove from $\mathcal{P}$ all internal nodes that fan-out only to latches in $\mathcal{C}$.*

This optimization simplifies the problem being solved by the CEC tool. It is derived from our practical experience with the typical anatomy of partial matches $\mathcal{R}_{L_{sub}}$. Our experience is that partial matches specified in $\mathcal{R}_{L_{sub}}$ are either complete to begin with or they are to be matched with latches that are unmatched so far (*i.e.* not part of L_{sub}). Matchings between latches in L_{sub} are either not observed (other than the ones stated in $\mathcal{R}_{L_{sub}}$) or not important from the point of view of the latch mapping problem.

Note that while in theory this optimization makes the overall method incomplete, in practise, in extensive experiments we have *never* found this to be the case. On the contrary this simple optimization provides significant runtime improvements in many (but not all) cases. In any case, this is an innocuous optimization which can easily be omitted from the method, for a particular problem, if there are any concerns of completeness, without impacting the correctness of the rest of the algorithm.

4. EXPERIMENTAL RESULTS

In the following we present experimental results on some representative circuits to validate the following two claims regarding the techniques proposed in this paper.

1. The hybrid method of Section 3.1 is an improvement over individual incomplete or complete methods.
2. The optimizations proposed in Sections 3.2.1 and 3.2.2 substantially improve the complete method.

Our benchmark suite consists of several large industrial circuits (few hundreds to few thousand latches) as well as the largest of the ISCAS89 sequential circuits. In the case of the ISCAS89 examples the two circuits (to be verified) were obtained by optimizing the original benchmark with the `script.rugged` combinational optimization script in **SIS** [10]. The industrial examples are obtained from a wide variety of design scenarios including 1.) combinational optimization of one gate-level design by hand and/or by a CAD tool, and 2.) two designs generated from different synthesis paths from RTL (*e.g.* one by hand and the other by a CAD tool). All experiments are reported on a 450MHz Sun UltraSparc-III machine, using implementations of the algorithms and techniques described in Section 3.

It is noteworthy that the time taken to solve a given latch mapping problem depends not only on the size (in terms of number of latches) of the problem but also on the difficulty of the problem. As in combinational equivalence checking this depends on how different the two sequential designs being compared are. In our experience almost 50-60% of typical latch mapping problems are relatively simple because the two designs have the same number of latches and are structurally very similar. In such cases the latch mapping solution consists of a pair-wise matching of latches from the two circuits, which can be efficiently and correctly obtained by incomplete methods alone [1, 4]. Our contributions are aimed at the remaining 40-50% of latch mapping problems.

Figure 3(a) presents a comparison of the latch mapping results between incomplete methods (a set of proprietary, in-house incomplete methods, as mentioned in Section 3), our implementation of a complete method (see Section 3.2) and the combination of these two as explained in Section 3.1. Note that in almost all the examples the two designs have different number of latches. Columns 2 and 3 show the number of latches in the 2 designs being matched. Columns $4, 5$ and 6 show the match times for the three methods. The number of matches have been noted in parentheses, whenever the method was unsuccessful in obtaining a complete match. In all but 2 of the examples the incomplete method was unsuccessful in finding a complete match. In all but one case, the complete method is able to find the complete match but is computationally much more expensive. Our proposed hybrid scheme, on the other hand, is able to find a complete match in all cases, and much faster than the individual complete method. In the two cases where the incomplete method finds a complete match, the combined scheme does not incur any additional cost. The example `Bench 1` provides an interesting case where the complete method cannot find a complete match whereas the combined method does. In this case some of the equivalence checks in the complete method aborted due to resource constraints. When using the combined method, these equivalence checks were circumvented by the partial match supplied by the incomplete methods.

The second set of results, presented in Figure 3(b), demonstrate the speed-up obtained by using the optimizations presented in Sections 3.2.1 and 3.2.2. The speedup number is a ratio of runtimes, of our implementation of the complete method, with and without the optimizations. The experiments cover two scenarios, 1.) where the complete method is run on the modified circuit model $\mathcal{P}^*$ (as it would be in the case of the combined method), 2.) when the complete method is run on the original problem $\mathcal{P}$ (these cases are marked as *(full)*. In all cases (except one where there is a slight slow-down) the optimizations provide a speedup of $1.5 - 2.5X$. The example `Bench 11` is a case where the two circuits were not equivalent. The results show that the proposed optimizations can be useful in a variety of contexts. The improvements obtained are significant considering the fact that the proposed optimizations carry virtually no computational overhead and provide a speed-up even when the complete algorithm has as few as $2 - 3$ iterations (this was the case for most benchmarks, with the combined method).

Circuit	# Latches		Match Time [# latches Matched: Spec./ Impl.]		
	Spec.	Impl.	Incomplete	Complete	Combined
Bench 1	181	181	62s [137/137]	1837s [179/179]	742s
Bench 2	478	478	19.5s	23.97s	19.5s
Bench 3	497	497	11.4s [471/471]	10.4s	16s
Bench 4	1256	1294	81s [1136/1162]	302s	247s
Bench 5	1592	1592	302s	1328s	302s
Bench 6	1607	2134	183s [1410/1410]	1261s	945s
Bench 7	2287	2494	160s [2151/2151]	899s	571s
Bench 8	3018	2543	231s [2053/2053]	1713s	1218s
Bench 9	3006	3070	531s [2828/2828]	7320s	3263s
s5378	164	163	2.5s [162/162]	2.8s	4.2s
s9234	211	135	5.5s [108/108]	11.3s	13.8s
s38417	1636	1464	57.5s [1278/1278]	150s	116.5s

(a) Proposed Hybrid Method Vs. Complete & Incomplete Methods

Circuit	Speed-up
Bench 1	2.32
Bench 6	2.50
Bench 7	1.43
Bench 9	0.98
Bench 10	1.76
Bench 11*	2.19
Bench 4 (full)	1.65
Bench 8 (full)	1.80
s38417 (full)	1.66

(b) Speed-up due to Optimizations of Sections 3.2.1 and 3.2.2

Figure 3: Experimental Results

5. CONCLUSIONS

Combinational equivalence checkers are widely deployed in industry today. However their application is often plagued by ineffective solutions to the latch mapping problem. We recognize that incomplete methods for latch mapping are inadequate for a large percentage of such problems arising in industry. Our experience indicates that complete methods, when applied alone to such problems, are unduly expensive. In this work we have proposed a complete and efficient methodology that seamlessly integrates an efficient and scalable incomplete method with a powerful complete method based on an in-house, state-of-the-art CEC tool. We further recognize that the application of the complete method as a black box is unable to utilize the information that the underlying problem is latch mapping. We have proposed several inexpensive, yet effective optimizations which further tune the complete method to latch mapping and extract significant performance gains.

We have proved the efficacy of our approach with results on large and complex industrial designs that have undergone extensive manual and automated optimization. In addition, we have presented results on several large public domain benchmarks. We believe that the methodology presented in this paper, along with previous complementary techniques [1, 4] can provide effective latch mapping support to CEC-based verification flows.

6. REFERENCES

[1] D. Anastasakis, R. Damiano, H.-K. T. Ma, and T. Stanion. A Practical and Efficient Method for Compare-Point Matching. In *Proceedings of the* 39th *Design Automation Conference*, pages 305–310, June 2002.

[2] D. Brand. Verification of Large Synthesized Designs. In *Proceedings of International Conference on Computer-Aided Design*, pages 534–537, Nov 1993.

[3] J. R. Burch and V. Singhal. Robust Latch Mapping for Combinational Equivalence Checking. In *Proceedings of International Conference on Computer-Aided Design*, pages 563–569, Nov. 1998.

[4] H. Cho and C. Pixley. Apparatus and method for deriving correspondences between storage elements of a first circuit model and storage elements of a second circuit model. U. S. Patent 5,638,381, June 1997.

[5] T. Filkorn. *Symbolische methoden für die Verifikation endlicher Zustandssysteme*. PhD thesis, Institut für Informatik der Technischen Universität München, Munich, Germany, 1992.

[6] S.-Y. Huang, K.-T. Cheng, K.-C. Chen, and F. Brewer. AQUILA: An Equivalence Checking System for Large Sequential Circuits. *IEEE Transactions on Computers*, 49(5), May 2000.

[7] A. Kuehlmann and F. Krohm. Equivalence Checking Using Cuts and Heaps. In *Proceedings of the* 34th *Design Automation Conference*, pages 263–268, June 1997.

[8] W. Kunz, S. Reddy, and D. Pradhan. Efficient Logic Verification in a Synthesis Environment. *IEEE Transactions on CAD*, 15(1):20–32, Jan. 1996.

[9] R. Mukherjee, J. Jain, K. Takayama, M. Fujita, J. Abraham, and D. Fussell. An Efficient Filter–based Approach for Combinational Verif ication. *IEEE Transactions on CAD*, 18:1542–1557, Nov. 1999.

[10] E. M. Sentovich et al. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, ERL, College of Engineering, University of California, Berkeley, May 1998.

[11] V. Singhal and J. Burch. Tight integration of Combinational Verification Methods. In *Proceedings of International Conference on Computer-Aided Design*, pages 570–576, Nov. 1998.

[12] D. Stoffel and W. Kunz. Record & Play: A Structural Fixed Point Iteration for Sequential Circuit Verification. In *Proceedings of International Conference on Computer-Aided Design*, pages 394–399, Nov. 1997.

[13] C. A. J. van Eijk. Sequential Equivalence Checking Based on Structural Similarities. *IEEE Transactions on CAD*, 19(7):814–819, July 2000.

[14] C. A. J. van Eijk and J. A. G. Jess. Detection of Equivalent State Variables in Finite State Machine Verification. In *Proceedings of International Workshop on Logic Synthesis*, May 1995.