Efficient Equivalence Checking of Multi-Phase Designs Using Retiming

† Gagan Hasteer
Ambit Design Systems
gagan@ambit.com

Anmol Mathur
Ambit Design Systems
amathur@ambit.com

Prithviraj Banerjee
Northwestern University
banerjee@ece.nwu.edu

Abstract

The use of multi-phase clocking scheme, aggressive pipelining and “sparse” encodings in high performance designs results in a tremendous increase in the state space. In this paper, we show that automatically transforming such designs to ones that have more “dense” encodings can result in significant benefits in using implicit BDD-based techniques for their verification. We formulate a relaxed retiming framework which is more powerful than traditional retiming in reducing the number of latches and show that it can be applied to the product machine model for checking sequential hardware equivalence (SHE) without altering the correctness of the SHE check. We combine retiming with phase abstraction [4] (a technique to transform multi-phase FSMs to single-phase FSMs for equivalence checking). The two transformations enable the SHE check to be performed on high performance controllers with large state space (more than 100 latches) from an industrial setting.

1 Introduction

Due to aggressive timing, many recent microprocessors use a multi-phase (most commonly two-phase with two non-overlapping complementary clocks) design methodology with level sensitive latches. Level sensitive latches avoid the timing costs associated with a synchronization boundary resulting from a register based design. They allow a more relaxed synchronization paradigm that allows for cycle borrowing between adjacent phases. However, such a design methodology increases the number of latches in the design by a factor of $k$, where $k$ is the number of distinct phases. A large class of multi-phase designs (including all multi-phase industrial designs that we encountered) use a very structured multi-phase methodology. For such designs it is possible to ignore all but one of the phases.

This research was supported in part by the National Science Foundation under grant MIP-9220854 and the Semiconductor Research Corporation under contract SRC 96-DF-100.

† This work was done when the author was working at MIPS Technologies Inc., Silicon Graphics. Current affiliation: Ambit Design Systems.

Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.

ICCAD ’98, San Jose, California

for functional verification. This transformation called phase abstraction allows us to perform equivalence checking on much larger FSMs than is possible otherwise [1]. Most recent microprocessor designs also use aggressive pipelining to allow faster clock frequencies. In addition, often very sparse encodings (such as one-hot encodings) are used for meeting the timing goals by avoiding the cost of decoding logic. This introduces additional states in the design that can be ignored for functional verification. We show that using min-latch retiming to reduce the number of state variables can significantly reduce the complexity of equivalence checking.

Researchers have used identical input-output behavior starting from a set of designated initial states (DIS) to check for equivalence. However, in many microprocessor designs, the DIS assumption cannot be made because the initial states are unknown since most latches do not have explicit reset. Sequential hardware equivalence (SHE) checks FSMs for equivalence in their steady states. An approach to check for SHE is to construct a product machine by merging corresponding inputs and XORing the corresponding outputs of the individual machines. The problem reduces to verifying that the outputs of the product machine are always 0 for all the steady states. An implicit algorithm for computing steady states and checking FSMs for SHE is presented in [2]. In this paper we present techniques to make this computation more tractable for multi-phase FSMs.

The usual bottleneck in the above approach is the computation of the steady states. The size of the BDD representing the transition relation and the steady states has support proportional to the number of memory elements and its size is determined by the nature of the transition relation (assuming a fixed variable ordering). Both phase abstraction and min-latch retiming reduce the support size for these BDDs by reducing the number of latches at the cost of a potential increase in the logic depth. Our results show that this is a beneficial tradeoff in the context of sequential equivalence checking. Another way to look at both these transformations is to say that they increase the encoding density of the FSMs under consideration. Consider an FSM with $n$ latches and $|S|$ steady states. We define the encoding density ($\rho$) to be the ratio of the number of steady states to the total number of possible states, i.e.

$$\rho = \frac{|S|}{2^n}.$$

Since both phase abstraction and retiming preserve steady state behavior of the FSM while reducing the number of memory elements ($n$), they increase the encoding density. Thus, increase in encoding density seems to positively impact steady state computation using BDDs. It should be noted that these techniques can be used for any other reachability related operations on
FSMs (such as in the context of model checking), although in this paper we focus only on using them to check SHE.

![Diagram of FSMs](image)

**Figure 1:** Motivation for phase abstraction and minimum area retiming.

**Example 1.** Figure 1 illustrates our point on a real microprocessor sub-design. A four bit value is generated by a combinational logic that is split across two phases. This value is then incremented by 0, 1, 2 or 3 depending on the increment control. Since clock cycles are at a premium, the new counter values for all the cases are computed before the increment control values are computed. The correct increment value is subsequently selected. Due to performance reasons the outputs of the four adders are latched (the adder for +0 is shown only conceptually). This yields a total of 32 latches (16 phi1 latches and 16 phi2 latches). Using phase abstraction we can remove all latches of one phase (say phi1) and ret ime the phi2 latches to 4 latches as shown. Since the outputs of the incrementers are correlated there can be only 16 different states of the phi2 latches. Thus, the encoding density of the original design is $16/2^{32} = 2^{-28}$ while that of the transformed design is 1. This example corresponds to a counter embedded in Ckt21 (see Section 5). Attempts to perform equivalence checking on identical copies of Ckt21 without retiming failed, but after retiming it was successful in under 20 seconds.

The main contributions of our paper are:

- We show that retiming is a valid SHE preserving transformation. We propose relaxed retiming which is more powerful than traditional min-area retiming in reducing the number of latches. Although relaxed retiming alters the behavior of an FSM, we show that it can be applied to the product machine model without altering the correctness of the SHE check.

- The combination of phase abstraction and retiming provides an interesting set of possibilities for reducing the state space. We give an efficient algorithm to combine phase abstraction and relaxed retiming for reducing the state space for equivalence checking. In particular, we address the issue of optimally retiming FSMs with multiple phases.

- We show that relaxed retiming combined with phase abstraction results in significant reduction in the complexity of equivalence checking of two-phase circuits from an industrial setting.

## 2 Preliminaries

An FSM $M$ is represented by a quintuple, $(Q, I, O, \lambda, \delta)$, where $Q$ is the set of states, $I$ is the set of input values, $O$ is the set of output values, $\lambda$ is the output function, and $\delta$ is the next state function. For simplicity, we assume that $M$ is fully specified i.e $\lambda : (Q \times I) \rightarrow O$ and $\delta : (Q \times I) \rightarrow Q$ are completely specified.

We also use $\lambda$ and $\delta$ to denote the output and next state functions on sequence of inputs. So, if $\pi = i_1, i_2, \ldots, i_k \in I^k$ is a sequence of $k$ inputs, then $\lambda(s, \pi) = \lambda(s, i_1) \cdot \lambda(s, i_1, \pi')$ and $\delta(s, \pi) = \delta(s, i_1) \cdot \delta(s, i_1, \pi')$ where $\pi' = i_2, \ldots, i_k$.

Two FSMs are compatible if they have the same number of inputs and outputs. All notions of FSM equivalence are meaningful only for pairs of compatible FSMs.

**Definition.** The product machine of two compatible machines $M_1, M_2$ is defined by machine $M(Q, I, O, \lambda, \delta) = M_1 \times M_2$, where $Q = Q_1 \times Q_2$, $I = I_1 = I_2$, $O = O_1 = O_2$ and the corresponding outputs are XORed together, $\lambda = \lambda_1 \cdot \lambda_2$ and $\delta = \delta_1 \cdot \delta_2$. If $s_1 \in Q_1$, $s_2 \in Q_2$ and $\pi$ is an input sequence then $\delta(s_1, s_2, \pi) = (\delta_1(s_1, \pi), \delta_2(s_2, \pi))$.

The notion of equivalence used in this paper is sequential hardware equivalence (SHE) which was proposed by Pixley in [4]. We will use the following characterization of SHE in terms of steady states of the product machine of the two FSMs being tested for SHE [2].

**Theorem 1** $M_1 \cong M_2$ if and only if the product machine $M = M_1 \times M_2 = (Q_M, I_M, O_M, \lambda_M, \delta_M)$ satisfies $\lambda_M(s, i) = 0$ for all states $s \in \Sigma(M)$, where $\Sigma(M) \subseteq Q_M$ is the steady state set of $M$ (union of the tSCCs of $M$).

## 3 Review : Phase Abstraction

In this section, we describe our multi-phase design methodology whose structure allows for phase abstraction, an efficient one-phase transformation which enables us to reduce the state space of the design.

For functional analysis, level sensitive latches can be modeled by equivalent edge sensitive registers. The clock edge on which a level sensitive latch closes can be considered as the edge that
equal number of latches of all phases. Further, these latches are phase aligned i.e. latches of phase $j$ feed latches of phase $j + 1$ (and no other phase).

**Property 3:** The values taken by the latches after the rising edge of clock $(j + 1)$ is fully determined as a function of phase $j$ latches and the primary inputs for all times in the future.

Phase abstraction refers to the removal of latches of all but one of the phases of an FSM. Based on the above properties the following theorem was proved in [1]:

**Theorem 2** Let $M'$ be the phase abstraction of a product machine $M = M_1 \times M_2$. Then all outputs of $M$ are 0 in its steady states if and only if all outputs of $M'$ are 0 in its steady states.

From Theorems 1 and 2, it suffices to consider the phase abstracted product machine for the SHE check. In the next section we discuss retiming as a transformation to further reduce the state space of the product machine. The techniques that we discuss in the sequel are also applicable to one-phase FSMs and we could have avoided the multi-phase discussion in this paper. However, the reason we address multi-phase FSMs is that multiple phases coupled with relaxed retiming provides an interesting set of possibilities for optimization (not all optimal). Besides, our benchmarks were all two-phase FSMs since this work was done in a two-phase industrial design environment.

## 4 Retiming Multi-phase FSMs

Retiming has been used as a transformation to reduce area and/or clock period. Leiserson and Saxe have given a polynomial time algorithm for exact minimum area retiming in [3]. Due to the DIS assumption made in research the issue of retiming the initial states is closely tied with retiming a machine [5]. However, SHE does not require a set of initial states. Thus we avoid the problem of retiming the initial states altogether. In this section, we show that retiming preserves SHE and give an efficient algorithm for retiming multi-phase FSMs. All proofs are omitted due to lack of space.

**Theorem 3** Let $M_1$ and $M_2$ be two FSMs and let $M_1'$ and $M_2'$ be their retimed versions. $M_1$ is SHE to $M_2$ if and only if $M_1'$ is SHE to $M_2'$.

Theorem 3 shows that we can retime the two machines to be compared for minimum area as a preprocessing step (to minimize the number of latches). This would reduce the state space complexity and make equivalence checking more efficient by increasing the encoding density. In practice, we construct the product machine first and then retime the product machine for minimum area. Since the primary inputs and outputs are the only interfaces where the individual machines interact in the product machine, this has the net effect of retiming the two machines separately. The benefit of this approach lies in the sharing of latches which get retimed to corresponding primary inputs and outputs in both the machines. Note that we can also obtain this sharing by a post processing of the inputs and outputs of the retimed individual machines. However, the retiming cost function for the individual machine does not know that a latch has the potential of being shared by another machine if retimed to the primary input. Hence, retiming the individual
machines may result in another optimal solution which does not push a latch to the primary input.

**Theorem 4** Let \( M = M_1 \times M_2 \) be a product machine and \( M' \) a retiming of \( M \), then \( M_1 \) is SHE to \( M_2 \) if and only if the outputs of \( M' \) are 0 in all its steady states.

Theorem 4 states that retiming the product machine is a valid transformation in deciding whether the given machines are SHE. Thus, in practice we retime the product machine itself enabling more aggressive reductions to take place.

### 4.1 Relaxed Retiming

We observe that the latches on primary input (output) lines can be interpreted to have a delay effect on that input (output). Thus, we can retime away the latches on the primary inputs and outputs of the product machine to the environment further reducing the state space complexity. In fact, we can show that it is possible to use relaxed retiming that allows latches to be borrowed from and discarded to the environment at primary inputs and outputs. Figure 5 illustrates the notion of borrowing/discard latches. Relaxed retiming allows more aggressive retiming of the product machine, thus resulting in even smaller number of latches in the min-latch retiming solution. The following results show that such relaxed retiming is valid on the product machine in the context of checking SHE.

Implementing relaxed retiming can be accomplished in a conventional retiming package by (1) Placing a large number of latches at all IOs to facilitate latch borrowing and (2) Forcing the cost of a latch at an IO to be zero. This means that latches at IOs do not contribute to the total cost of a retiming solution and this is equivalent to discarding latches at IOs to the environment.

### 4.2 Retiming Multiple Phases Optimally

The combination of phase abstraction with retiming gives us an interesting set of possibilities for retiming. Since retiming can be a time consuming operation, it is important to perform it optimally and efficiently. The main choice is whether to perform phase abstraction first and then do retiming, or first perform retiming on both the phases and then do phase abstraction. In case phase abstraction is done first, which phase should be abstracted so that the final solution after min-latch retiming has the smallest possible number of latches? We can pick the phase with lower number of latches in the product machine and perform min-latch retiming. However, the following example illustrates that this may not always yield the optimal solution, if we do not use relaxed retiming. We subsequently prove that in the relaxed retiming model, picking either phase for abstraction followed by min-latch retiming yields the same number of latches in the min-latch retiming solution. Thus, picking the phase with the smaller number of latches and then doing a relaxed min-latch retiming yields the provably optimum solution.

![Figure 5: Latch borrowing and discarding in relaxed retiming.](image)

**Lemma 1** Let \( M' \) be the FSM obtained by borrowing one latch at an input or output of FSM \( M \). Outputs 0 in all steady states if and only if \( M' \) outputs 0 in all steady states.

Note that discarding a latch at an input is just the inverse of borrowing, and since Lemma 1 defines an "if and only if" property for borrowing, it follows that the above lemma also holds for discarding a latch at an input or output.

**Theorem 5** Let \( M' \) be a retimed version of \( M \) assuming the relaxed retiming model. The outputs of \( M \) are 0 in all its steady states if and only if the outputs of \( M' \) are 0 in all its steady states.

**Corollary 1** Let \( M_1 \) and \( M_2 \) be FSMS and let \( M' \) be the FSM obtained by retiming \( M_1 \times M_2 \) using relaxed retiming. Then \( M_1 \) is SHE to \( M_2 \) if and only if the outputs of \( M' \) are 0 in all its steady states.

**Example 2** Figure 6a illustrates a two-phase machine with its combinational logic removed for ease of implementation.

![Figure 6: Retiming different phases of a machine – picking the phase with the smaller number of latches may not give the best solution using non-relaxed retiming.](image)
illustration. The original machine has three latches of phase 1 and four latches of phase 2. The greedy solution would be to pick phase 1 and retime the machine to get the machine of Figure 6b. However, this choice is not optimal. By picking phase 2 and subsequently retiming, we would get the machine of Figure 6c which has lesser number of latches. Notice that if we used relaxed retiming then we could discard 2 latches from the solution in Figure 6b and one latch from Figure 6c, and thus both the solutions would have 1 latch in the min-latch retiming solution.

In the subsequent discussion, we will assume that we are performing relaxed retiming unless stated otherwise. Let $M_{o_i}$ denote the phase abstracted machine $M$ which keeps only the latches of phase $i$. Further, let $r_{min}(M)$ denote the machine obtained by min-latch retiming of $M$, and let $|r_{min}(M)|$ denote the number of latches in $r_{min}(M)$. For simplicity, let us assume a two-phase design methodology. The following theorem shows that we can pick any of the two phases in phase abstraction and perform min-latch retiming to obtain the optimum retiming solution. Since the complexity of retiming is a function of the number of latches, it is better to choose the phase with the smaller number of latches.

**Theorem 6**

$$|r_{min}(M_{o_1})| = |r_{min}(M_{o_2})|.$$ 

Similar arguments can be used to generalize the above result to $k$-phase systems. All of our experiments were performed on two phase designs. Given two machines $M_1$ and $M_2$, the following steps can be performed to check whether they are SHE:

1. The product machine is constructed: $M = M_1 \times M_2$.
2. The smaller of the two phases is retained during phase abstraction to get $M_{o_i}$.
3. The phase abstracted design is subjected to min-latch retiming in the relaxed retiming model to obtain $M'_{o_i}$.
4. The outputs of $M'_{o_i}$ are checked in its steady states. If they are 0 in all the steady states, the machines $M_1$ and $M_2$ are SHE otherwise they are not.

5 **Results**

Phase abstraction and retiming have been implemented as a part of our prototype equivalence checking engine, Equichk. Tables 1 and 2 show the performance of the algorithm on a set of industrial controllers on SGI Power Challenge machines with 200MHz clock and 2048MB of main memory. The BDD variable ordering scheme is based on a depth first ordering of the primary inputs. The Colorado BDD package was used for symbolic manipulation. The present state and the next state variables of all latches are constrained to occur together. The pairs of latch support variables of the product machine are interleaved with the primary inputs. The interleaving is based on a greedy heuristic. A latch variable pair is inserted immediately after the last primary input in the support of the latch. Dynamic variable ordering was enabled with the “group sift” parameter. Time is reported in seconds for the entire application. Memory is the maximum BDD memory size in MegaBytes (MB). Timeout for our experiments is set as 12 hours. Blank entries correspond to experiments which could not complete because of their time and/or memory requirements. For experimentation, two identical copies of a set of two-phase controllers from industry were checked for SHE.

Table 1 compares the number of latches in the product machine before and after phase abstraction, retiming and relaxed retiming. It should be noted that our framework of relaxed retiming currently implements only the leading of latches to the environment. Due to the limitations of the underlying package we have not been able to implement latch borrowing. However, the results would become only better in the complete framework. The total number of latches in the first column is the sum of the latches of the two phases in the product machine (twice the number of latches in a single copy of the circuit). The number of latches after phase abstraction is less than half of the original number because the phase with lesser number of latches is retained.

As seen from the table, retiming is not effective for memory and queue controllers because there are many embedded pointers and counters which cannot be retimed. However, it is particularly effective for arithmetic and system interface controllers. These controllers are characterized by decoding logic with many latches put in due to timing constraints. Example 6.1 illustrates a part of the logic of Ckt121 where significant reduction is obtained by retiming such latches (90%). Cache controllers exhibit a mix of the two types of controllers described above. For arithmetic controllers latches are used to store incoming instruction streams before decoding. In the product machine, these latches get shared after retiming leading to very large reductions after relaxed retiming. In case of Ckt12 the number of latches reduces by as much as 98%.

The final results are impressive with phase abstraction showing 58% reduction in the number of latches, min-latch retiming
gaining around 13% and relaxed retiming adding another 7%. Our results also show that there is a class of machines characterized by the arithmetic controllers which are amenable to retiming. However, retiming does not buy much on controllers with embedded counters and pointers such as the queue controllers. Thus, a prior knowledge of the structure of the machine can help in deciding whether the retiming option should be used in *EppCheck* or not.

Table 2 demonstrates the effectiveness of relaxed retiming in reducing the state space for equivalence checking. The biggest win for the retiming approach is illustrated by *Ckt21*. This controller fails to complete without retiming. However, using retiming the state space is exponentially reduced. The resulting state space structure is so simple that the equivalence checking completes in less than five seconds. As expected significant gains are observed for all arithmetic and system interface controllers. As explained above these circuits are amenable to retiming as they do not have embedded counters. Some cache controllers also show improved results for the same reasons. A direct correspondence can be found between the run times of Table 2 and the latch reductions of Table 1.

### 6 Conclusions

We have presented techniques for automatically transforming multi-phase FSMs using phase abstraction and retiming to contain the complexity of FSM equivalence checking. We have demonstrated that increase in encoding density allows the analysis of larger FSMs using implicit BDD-based techniques. The validity of both phase abstraction and the relaxed retiming model is proven and efficient algorithms are presented to carry out these transformations. It is clear from our experiments with a large class of industrial designs that these techniques increase the size of FSMs that can be handled using formal methods significantly. It would be interesting to study further the relation between encoding density and the effectiveness of BDD-based approaches for formal verification. Also, finding other transformations that can be used to alleviate the complexity of FSM verification would be of great interest.

### References


