Enhanced Verification by Temporal Decomposition

Michael L. Case Hari Mony Jason Baumgartner Robert Kanzelman
IBM Systems and Technology Group

***Abstract*— This paper addresses the presence of logic which has relevance only during initial time frames in a hardware design. We examine transient logic in the form of signals which settle to deterministic constants after some prefix number of time frames, as well as primary inputs used to enumerate complex initial states which thereafter become irrelevant. Experience shows that a large percentage of hardware designs (industrial and benchmarks) have such logic, and this creates overhead in the overall verification process. In this paper, we present automated techniques to detect and eliminate such irrelevant logic, enabling verification efficiencies in terms of greater logic reductions, deeper Bounded Model Checking (BMC), and enhanced proof capability using induction and interpolation.**

I. Introduction

Automated verification of sequential hardware designs is often a computationally challenging task. Being a PSPACE problem, the size of the design under verification can often render an automated solution intractable. Many hardware designs contain extraneous artifacts which are largely irrelevant to the core verification task, yet whose presence creates bottlenecks to the verification process. This paper addresses two particular types of artifacts: *transient signals* which after a certain number of time steps settle to a fixed constant value, and *initialization inputs* used to encode intricate initial states which become irrelevant after a certain number of time steps. We present algorithms to automate the identification of such artifacts, as well as to eliminate these artifacts to enhance the overall verification process.

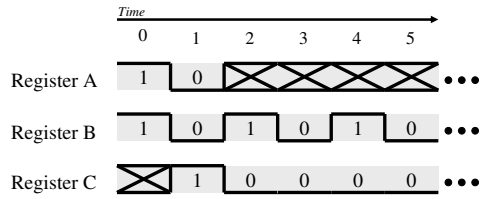

Fig. 1. Register C is a transient signal. X-valuations designate undetermined valuations, i.e. the possibility of either a 0 or 1 value.

Definition 1 (Transient Signal): Let $x(D,t)$ denote the value of signal x in design D at time t. A transient signal is any Boolean signal s such that $\exists T \in \mathbb{N}.\ \exists C \in \{0,1\}.\ \forall$ time $t \geq T.\ s(D,t) \equiv C$. The smallest T for which this holds is referred to as the *transient duration* and C as the *settling constant*.

In Figure 1, register C is a transient because it takes value 0 starting at time 2 and holds this value for all time. Registers A and B are not transients because they remain undetermined in value and do not settle to a fixed constant value, respectively. We have found that transient signals occur frequently both in industrial and benchmark designs.

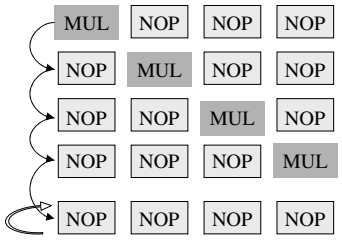

Fig. 2. FPU pipeline as driven by the verification environment

Transient logic arises due to numerous causes. One such cause is the presence of an initialization sequence. A common design style allows a design to power-up in a non-deterministic state from which sequence of state transitions bring it to a "reset state" from which it behaves consistently. Commonly dedicated logic is used to force the design through its initialization phase, and verification assumptions/checkers are tailored to take effect only once the initialization phase is completed. Much of the logic settles to a fixed constant value once the initialization is complete, and this logic is amenable to simplification by our proposed techniques.

Numerous methodologies have been proposed to eliminate such overhead, – e.g. by using three-valued simulations of the initialization phase (applying X-values to reflect non-determinism, as depicted in Figure 1) to conservatively determine a set of states the design can reside in post-initialization [1], [2]. While such methodologies are well-motivated, they require dedicated manual effort to decompose the overall verification task. They also ultimately over-approximate the set of post-initialization states since they conservatively treat non-deterministic signals as non-constant, losing more subtle constraints which may exist over the post-initialization states. This latter problem in turn may prompt a logic designer to conservatively add to the initialization logic to avoid spurious failures, a manual process which may result in suboptimal silicon.

Transient logic may also arise due to the verification testbench itself. Testbenches consist of three components: 1) a *driver* containing enough input assumptions to provide meaningful input stimulus, 2) the design under verification, and 3) a *checker* to verify the correctness of the design under the given inputs. The driver may be constructed to over-constrain the inputs of the design, e.g., to test the design only against a subset of its possible behaviors to facilitate a case-splitting strategy. Under this reduced set of inputs, many internal design signals may settle to constant behavior after a certain number of time steps.

An example of transient logic arising from the verification testbench can be found in the Floating-Point Unit (FPU) verification methodology proposed in [3]. To help cope with

the complexity of such verification, this methodology checks the correctness of a single opcode propagating through an empty pipeline as depicted in in Figure 2. No-operation (NOP) opcodes are driven after the single opcode under evaluation, and after the meaningful opcode is evaluated the internal state of the FPU settles to constant NOP behavior. All signals in the FPU may thus be viewed as transient logic when the inputs are driven in this manner.

A related problem is that of extraneous initialization inputs.

Definition 2 (Initialization Input): Let Σ be a set containing the design's next state functions, properties, and constraints. An *initialization input* is a primary input whose value can only propagate to any $\sigma \in \Sigma$ at time 0. At all other times, the value of the initialization input is not observable with respect to Σ.

It is common for designs to have a set of multiple possible initial states. The driver can non-deterministically select a single initial state from this set by introducing primary inputs. These are initialization inputs because their values after the first time frame are irrelevant. The method of eliminating transient signals presented in this paper produces designs with large sets of possible initial states and many initialization inputs. Therefore we propose a method to identify a subset of these initialization inputs that can be safely replaced with constant values, enhancing our ability to eliminate transients from designs without significantly increasing the total size of the design.

The contributions of this paper are as follows:

1) We propose an algorithm to identify the existence and duration of transient signals in Section II-A. This method utilizes ternary simulation hence is very fast and scalable.

2) We propose an efficient method to eliminate transient logic in Section II-B. This works by decomposing an unbounded verification problem into two sub-problems: bounded verification over the initial time frames during which transient behavior occurs, and unbounded verification over the remaining time frames. The unbounded verification can safely assume that all transients have settled to their post-transient constant values and thus can simplify the complex unbounded verification problem.

3) We propose a technique to identify initialization inputs that can safely be replaced by constants in Section III. This method uses bounded model checking combined with structural analysis, and thus is scalable. It is useful to eliminate inputs that exist in the testbench to model power-on non-determinism. Our method of eliminating transient logic can manufacture such inputs, so as well as being a generally useful simplification, the technique to eliminate initialization inputs in Section III is particularly useful after transient logic is eliminated.

4) We provide a large set of experimental results throughout this paper to illustrate the benefits of our techniques across numerous academic and industrial designs.

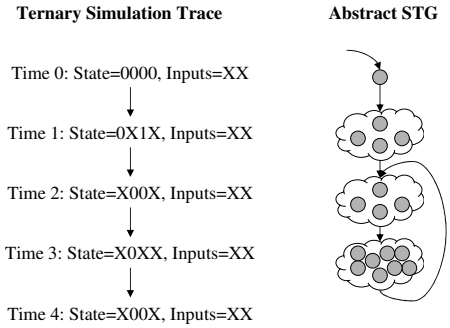

Fig. 3. Example ternary simulation execution

II. Temporal Decomposition

A. Detection of Transient Signals

Ternary simulation is a method to efficiently over-approximate the set of reachable states in a design. It works by conservatively modeling primary inputs with ternary X values and simulating a sequence of 3-valued states until a state is repeated. Upon convergence, the set of observed 3-valued states constitutes an over-approximation to the set of reachable states. An example ternary simulation run is depicted in Figure 3, where the state at time 2 is repeated at time 4 indicating that the reachable states have been over-approximated.

While the over-approximation of such ternary simulation is often too coarse for property checking itself, it is useful to efficiently identify useful design characteristics. For example, certain constant and equivalent signals may be detectable using this approach, and this enables the design to be simplified. Clock-like oscillating signals may also be detectable which enable temporal phase abstraction [5].

Ternary simulation can be augmented to efficiently detect a subset of the transient signals, in addition to the transient duration after which they settle to constant behavior.

Theorem 1 (Transient Detection by Ternary Simulation): Let $x(S,t)$ denote the value of signal x in the 3-valued state seen at time t. Let $i < j$ such that $\forall x$, $x(S,i) = x(S,j)$. If $\exists\, C \in [0,1]$, $\exists\, \sigma$ such that $\forall\, k \in [i,j]$, $\sigma(S,k) = C$ then σ is a transient signal with transient duration at most i.

It is straight-forward to use ternary simulation as per Theorem 1 to find transient signals, and a simple implementation[1] is shown in Algorithm 1. After convergence, a sweep over all signals is performed to see which remained constant within the state-repetition loop. Those which remained constant are added to the transient list, along with the constant they settled to and the latest time frame at which they evaluated the non-constant value. This represents an upper-bound (due to the over-approximation of ternary evaluation) on their transient duration.

B. Simplification of Transient Signals

Given a set of transient signals, consider the maximum transient duration within this set. Before the maximum duration,

[1]It is possible to optimize Algorithm 1 by incorporating the transient signal book-keeping within the ternary simulation loop.

Algorithm 1 Detection of Transient Signals

```
1: function detectTransients(design)
2:    // Execute ternary simulation until convergence
3:    History := ∅
4:    ternaryState := getTernaryInitialState(design)
5:    for (time = 0; ; time++) do
6:       if (ternaryState ∈ History) then
7:          cycleStartTime := calculateCycleStartTime(History, ternaryState)
8:          break
9:       end if
10:      History := History ⋃ ternaryState
11:      ternaryState := getNextTernaryState(design, ternaryState)
12:   end for
13:
14:   // Extract the transient signals
15:   transients := ∅
16:   for all (signals s in design) do
17:      for all (constants C in {0, 1}) do
18:         if (∀ time > cycleStartTime, s = C) then
19:            duration := latestToggle(s, History)   // ≤ cycleStartTime
20:            transients := transients ⋃ (s, C, duration)
21:         end if
22:      end for
23:   end for
24:
25:   return transients
26: end function
```

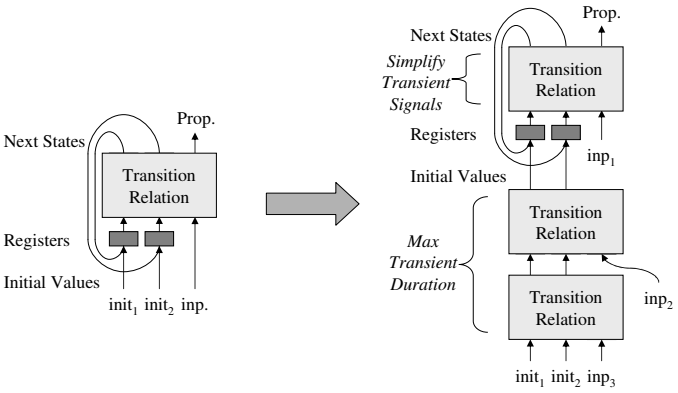

Fig. 4. Time-Shifting a design and simplifying transient signals

one or more of these signals may assume values which differ from their settled constants. We use BMC [6] to check the validity of any properties on these initial time frames. After the maximum duration, all transients in the set have settled to their corresponding constant values. The design can be simplified by replacing these transient signals with their constant values, and an unbounded verification process can check the remainder of the time frames in the simplified model.

We wish to optimize unbounded-verification within a transformation-based verification (TBV) framework [4]. In such a framework, a sequence of transformations is applied to a design (often represented as a *netlist*) prior to the invocation of a terminal verification algorithm, allowing the reductions inherent in the former to yield substantial speedups to the latter. As such, we cast our transient simplification routine as a netlist transformation rather than a customization to an existing verification algorithm. This allows these simplifications to be compatible with any downstream synthesis or verification algorithm.

In order to check properties only after the transient duration using a general-purpose verification algorithm, it is necessary to *time-shift* the design. In this process, we adjust the time basis of the design such that time *maxTransientDuration* in the original design corresponds to time 0 in the time-shifted design. Figure 4 demonstrates how such shifting may be accomplished. The design's initial state is modified such that the time-shifted design starts in any state reachable in *maxTransientDuration* time steps. In this work, we achieve such a transformation by unrolling the transition relation and driving the initial states with the output of the unrolled structure. This is akin to using structural symbolic simulation to compute the new set of initial values.

Algorithm 2 illustrates a simple procedure that will use a set of detected transient signals to simplify the design. BMC is used to check the properties before the maximum transient duration. Then the design is time-shifted, and the netlist is simplified by merging transients with their respective settling constants. We limit the runtime to 10-seconds because most of the benefits are obtained quickly at low time t.

Algorithm 2 Simplification of Transient Signals

```
1: function simplifyTransients(design, transients)
2:    maxTransientDuration := computeMaxTransientDuration(transients)
3:    for (t = 0; (t < maxTransientDuration); t++) do
4:       Validate that properties are not falsifiable at time t with BMC
5:
6:       // Time shift by 1 clock cycle
7:       for all (registers r ∈ design) do
8:          initialValue(r) := nextState(r)
9:       end for
10:
11:      // Incrementally simplify the design
12:      for all (g ∈ transients) do
13:         if (transients[g].settlingTime ≤ t) then
14:            merge(design, g, transients[g].settledConstant)
15:         end if
16:      end for
17:   end for
18: end function
```

C. Results

All experiments in this paper are performed on a set of 135 difficult industrial verification problems seen within IBM and 28 difficult HWMCC '08 designs [7]. The IBM benchmarks are from a suite of microprocessor property checking and sequential equivalence checking benchmarks that are not easily solved with BMC, interpolation, or induction.[2] The HWMCC benchmarks were either unsolved in HWMCC '08 or only solved by a few of the entrants. The size of the sequential designs varied with the largest design consisting of 97,000 registers and 632,000 and gates (when expressed as an AIG [12]).

Table I examines the transients that were present in some of these benchmarks. The columns "T. Found" show the number of transients found with Algorithm 1. In total, transients were present in 48.6% of the industrial designs and 25.0% of the HWMCC designs. On average, there were 22.7 transient signals in each industrial design that had transients and 4 transients in each corresponding HWMCC design.

[2]Each algorithm was run for 10 minutes. Induction incrementally increases k until the 10 minute timeout is reached (or a property is verified).

TABLE I. Transients Found and Simplified, On a Subset of Our 135 IBM and 28 HWMCC Designs

	Design Size			T. Found		Transients Simplified		T. Found With Induction		Logic Size Change	
Design	ANDs	Regs	Inputs	Num.	Dur.	Num.	Dur.	Num.	Runtime	TR	Total
IBM0	3039	291	45	8	1	8	1	120	4.18 s	-1.10%	1.83%
IBM6	77313	9829	544	1	1	1	1	0	0.16 s	-1.44%	2.05%
IBM8	3011	396	86	2	2	2	2	12	11.50 s	-0.27%	2.76%
IBM22	11218	908	924	23	10	23	10	36	9.48 s	-17.17%	19.48%
IBM23	8713	477	6	1	1	1	1	0	0.17 s	-9.57%	0.00%
IBM28	116322	21607	1316	1150	74	0	0	0	64.09 s	0.00%	0.00%
IBM31	7291	696	61	4	3	4	3	0	44.08 s	-2.23%	7.26%
IBM32	8177	698	65	4	3	4	3	7	49.36 s	-1.18%	7.18%
nusmvbrp	464	52	11	2	1	2	1	5	0.07 s	-8.18%	3.18%
nusmvguidancep2	1748	86	84	2	1	2	1	0	0.39 s	-1.89%	15.14%
nusmvqueue	2376	84	82	2	1	2	1	0	0.28 s	-4.10%	11.79%
nusmvreactorp2	1242	76	74	2	1	2	1	0	0.35 s	-5.52%	15.82%
r17b	4087	322	613	18	2	16	1	1	3.30 s	-30.38%	44.03%

Our implementation of Algorithm 2 bounds the total runtime at 10 seconds, and hence not all designs that had transients were simplified. Columns "Transients Simplified" show the transients that were utilized for simplification: in total 43.0% of the IBM designs and all 25.0% of the HWMCC designs were simplified. The transient simplification process entailed time-shifting the industrial designs by an average of 4.5 time frames, and the HWMCC designs were each time-shifted by 1 time frame.

Algorithm 1 uses ternary simulation to find transients. This procedure is fast – merely finding transients takes less than 1 second on all of our benchmarks – but lossy. As an alternative, consider time-shifting the design by 6 time frames, sufficient to capture 83% of the transients found with ternary simulation, and then using an inductive[3] routine to find sequentially-constant signals in the modified design. Each constant signal is a transient, assuming the constant did not exist before time-shifting. The columns "T. Found With Induction" illustrate the results of this experiment. On 24.1% of the designs our ternary simulation procedure found more transients than the inductive procedure, indicating that many of the found transients are non-inductive. On 53.8% of the designs the inductive procedure found more, indicating that on these designs induction was more complete than ternary simulation. The ternary simulation procedure was often orders-of-magnitude faster than the inductive procedure, and for this reason Algorithm 1 uses ternary simulation.

Columns "Logic Size Change" show the effect of transient simplification on the logic size (after finding transients with ternary simulation). Transient simplification decreased[4] the size of the transition relation (TR) in the IBM designs by 1.3% and the transition relations of the HWMCC designs by 7.4%. However, when combined with time-shifting, the resultant designs had complex initial value logic which led to an increase in size by 16.1% on the IBM designs and 25.8% on the HWMCC designs. This increase can be mitigated by removing unnecessary initialization inputs, as described in Section III.

III. Initialization Inputs

As discussed in Section I, some testbenches contain initialization inputs to model complex initial values. Additionally, initialization inputs often arise due to the symbolic simulation used in time-shifting netlists for the simplification of transient signals. (A similar complexity of initial values occurs as a byproduct of peripheral retiming [4], [8].) The increase in size due to the time-shifted initial values is undesirable, in that it may offset the reduction in size resulting from the merging of transient signals to constants. While certain algorithms such as induction may be immune to increases in initial value complexity, in a TBV setting some algorithms may be hindered by this increased complexity. We thus are motivated to simplify complex initialization logic.

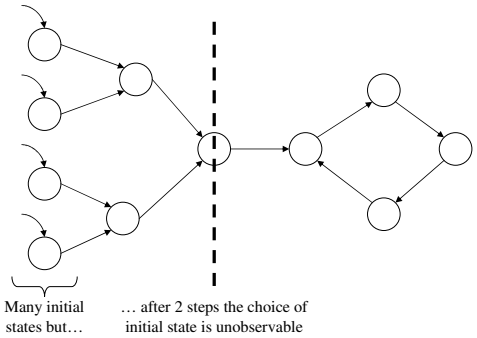

Fig. 5. STG illustrating the irrelevance of certain initialization inputs

A. The Overhead of Unnecessary Initialization Inputs

Recall that an initialization input is one whose value only affects the design at time 0, often used to encode complex initial states (Definition 2). Time-shifting inherently introduces a significant number of initialization inputs (eg.: Figure 4). Not all of these inputs may be relevant to the behavior of the design, and the time-shifted design can be optimized by removing a subset of the new inputs.

Consider the state-transition graph (STG) shown in Figure 5. This design can start in four possible initial states, and hence two initialization inputs may be used to model this set of initial states. All paths in the STG pass through a single dominator state after two time steps, and so for time $t \geq 2$ it is possible to reduce the number of possible initial states without affecting the design's behavior. As the set of initial values is represented in a netlist using extra initialization inputs, such a simplification may be performed by replacing these initialization inputs by constants. This type of simplification is therefore a form of Observability Don't Care (ODC)-based simplification [9] because the individual initial states are not

[3] $K = 5$ induction with unique-state constraints

[4] The size change is measured as $1 - \frac{\text{new regs.}+\text{new ands}+\text{new inputs}}{\text{old regs.}+\text{old ands}+\text{old inputs}}$.

observable after a finite number of time steps.

B. Reduction of Initialization Inputs

A subset of initialization inputs which are irrelevant to the behavior of the design may be detected using structural analysis alone. Theorem 2 illustrates how unrolled Cone of Influence (COI) analysis can be used to identify a subset of inputs that have no influence on the netlist after a fixed number of time steps t.

Theorem 2 (Detection of Irrelevant Initialization Inputs): Recall that $x(D,t)$ denotes the value of signal x in design D at time t. Let UD be a temporal unrolling of D and let Σ be the set of all next-state functions and properties in D. If $\exists\, t \in \mathbb{N}$ and input i s.t. $\forall\, \sigma \in \Sigma, i \notin COI(\sigma(UD,t))$ then we can replace i with an arbitrary signal to produce design D' s.t. $\forall\, t > T$, $\forall\, \sigma \in \Sigma$, $\sigma(D',t) = \sigma(D,t)$.

Once an irrelevant initialization input is identified, it may be replaced with an arbitrary value[5] without affecting the behavior of the design after time t. The modified design is guaranteed to be equivalent to the original design after time t, but before time t the modified design can only visit a subset of the states of the original design. To ensure that valid counterexamples are not missed during this simplification, it is necessary to validate the correctness of the properties up to time t before this simplification. (This is similar to the initial property validation necessary before merging transient signals in the time-shifted netlist as per Algorithm 2.)

Algorithm 3 Eliminate Irrelevant Inputs

```
1: function simplifyInputs(design, maxTime)
2:   unrolledModel := ∅
3:   for all (registers r ∈ design) do
4:     instance of r in unrolledModel := initialValue(design, r)
5:   end for
6:   for (t = 0; t < maxTime; t++) do
7:     Validate that properties are not falsifiable at time t
8:
9:     // Incrementally unroll the model
10:    Append one transition relation to unrolledModel
11:    unrolledModel := resynthesize(unrolledModel)
12:
13:    // Compute the unrolled COI
14:    C := ∅
15:    for all gates g ∈ ( next-state functions ⋃ properties ) do
16:      C := C ⋃ COI(unrolledModel, last temporal instance of g)
17:    end for
18:
19:    // Remove unnecessary inputs
20:    for all (primary inputs i ∈ design) do
21:      if (i ∉ C) then
22:        merge(design, i, 0)
23:      end if
24:    end for
25:  end for
26: end function
```

Algorithm 3 illustrates our initialization input elimination routine. Time t is gradually increased[6] until computational resources are exceeded. For each t, we first validate that the properties cannot be falsified at that time frame. Next, the design is incrementally unrolled and its COI is inspected. In order to reduce the size of this COI and enhance the reduction potential of this technique, synthesis algorithms (SAT-sweeping [10], rewriting [11], and others) are employed on the unrolled design. Inputs that fall out of the COI of the next-state functions and properties are removed from the design by merging them with the constant 0.

We note that unlike most ODC-based simplification routines, all simplifications identified by this routine are inherently compatible. The simplifications can be utilized simultaneously without interfering with one another, resulting in a more efficient algorithm. Additionally, because the approach relies on the circuit structure it is highly scalable. Nonetheless, this technique is incomplete and some irrelevant initialization inputs may not be identified. It can be complemented by a post-processing of traditional, yet more expensive, ODC-based simplification (Section III-C.2).

[5]Our implementation replaces such unnecessary inputs with the constant 0.

[6]We limit $t < 5$ because the simplification potential quickly saturates as t increases. 91.0% of all simplifications are done with $t = 0$, 5.7% with $t = 1$, and $< 1.5\%$ with each of $t = 2, 3, 4$.

TABLE II. Input Simplification, On A Subset of Our Designs

	Design Size			Input Simplification	
Design	ANDs	Regs	Inputs	Init. Inputs	Removed
IBM0	3039	291	45	0	0
IBM6	77313	9829	544	0	0
IBM8	3011	396	86	1	0
IBM22	11218	908	924	275	201
IBM23	8713	477	6	0	0
IBM28	116322	21607	1316	232	13
IBM31	7291	696	61	0	0
IBM32	8177	698	65	0	0

C. Results

1) Initialization Input Simplification: Our first set of experiments considers the impact of our irrelevant input elimination technique alone. A subset of these results are provided in Table II. This simplification technique was effective on 22.4% of the industrial designs, and within those designs 11.7% of the initialization inputs were removed.

We were unable to find any initialization inputs in the HWMCC designs.

2) Transient Simplification + Initialization Input Simplification: Our next set of experiments considers the ability of the irrelevant input elimination technique to reduce the overhead of the symbolic simulation necessary to compute the initial values of the time-shifted netlist produced in transient simplification. Our implementation first eliminates any identified transient signals, and then attempts to eliminate the resulting initialization inputs. A selection of our results is shown in Table III. The transient simplifications are illustrated for each design, and the number of initialization inputs after this simplification step are counted. Initialization input elimination was able to simplify 58.7% of the industrial and 14.3% of the HWMCC designs. The logic bloat of these combined simplification steps was 11.7% on the industrial designs and 9.4% on the HWMCC designs (compare to 16.1% and 25.8%, respectively, without initialization simplifications). This demonstrates that initialization input simplification is effective in mitigating the bloat that is caused by transient simplification.

TABLE III. Input Simplification After Transient Simplification, On a Subset of Our 135 IBM and 28 HWMCC Designs

	Design Size			Transient Simp.		COI-Based Input Simp., Alg. 3			Total	ODC-Based Input Simp.	
Design	ANDs	Regs	Inputs	Num.	Dur	Init. Inputs	Removed	Time	Bloat	Inp. Removed	Time
IBM0	3039	291	45	8	1	25	0	1.59 s	0.73%	0	4.82 s
IBM6	77313	9829	544	1	1	409	1	9.55 s	1.77%	92	3610.39 s
IBM8	3011	396	86	2	2	58	12	0.67 s	2.18%	0	13.78 s
IBM22	11218	908	924	23	10	368	214	1.61 s	-12.35%	0	26.20 s
IBM23	8713	477	6	1	1	0	0	0.00 s	-9.57%	0	0.00 s
IBM28	116322	21607	1316	0	0	0	4	0.34 s	10.49%	4	109.29 s
IBM31	7291	696	61	4	3	178	3	0.36 s	4.99%	0	36.58 s
IBM32	8177	698	65	4	3	178	3	0.37 s	5.96%	0	34.81 s
nusmvbrp	464	52	11	2	1	11	0	0.30 s	-5.00%	0	0.49 s
nusmvguidancep2	1748	86	84	2	1	84	0	1.42 s	13.25%	33	13.09 s
nusmvqueue	2376	84	82	2	1	82	0	1.86 s	7.69%	0	23.05 s
nusmvreactorp2	1242	76	74	2	1	74	0	1.57 s	10.30%	1	18.00 s
r17b	4087	322	613	16	1	558	3	9.25 s	13.54%	17	698.12 s

Algorithm 3 uses BMC and structural analysis to remove initialization inputs, and because of this it is fast yet does not identify all unnecessary initialization inputs. In columns "ODC-Based Input Simp." of Table III we illustrate the relative utility of Algorithm 3 by running a more thorough initialization-input simplification method afterward. This technique is similar to traditional ODC-based optimization algorithms in that it creates a side copy of a logic window of the design for each candidate simplification, assessing whether a particular simplification may be witnessed as altering design behavior with respect to that window [17]. This particular technique was limited to assessing the validity of merging initialization inputs relative to a logic window of a configurable sequential depth. These results illustrate that our method identifies 44.5% of the unneeded initialization inputs in just a fraction of the time needed to do the more exhaustive check; the former was limited to a runtime of 10 seconds total (identical to transient simplification), whereas the latter was limited to 10 seconds per initialization input – resulting in runtimes in excess of 1 hour in numerous cases.

3) Runtime: Most results presented above were presented without runtimes. Transient simplification primarily leverages ternary simulation and BMC, and initialization simplification leverages structural methods and BMC. These techniques are efficient and scalable, and most analysis can be performed incrementally. We thus have designed both of these simplification techniques to incrementally simplify the design over time frames until some computational limit is exhausted. All experiments were run on a cluster of 2 GHz IBM POWER CPUs, and the combined runtime for both transient simplification and initialization simplification was limited to 20 seconds. We often omit runtimes in our tables because this 20-second overhead is negligible in the verification of these complex designs.

IV. Discussion and Experimental Results

The algorithms described above were implemented in the IBM internal verification tool *SixthSense* [8]. *SixthSense* is built upon a TBV framework where various engines incrementally simplify complex properties before leveraging a terminal verification engine to attempt to solve the simplified problem. The techniques presented in this paper have been implemented as one of these engines. In this section we analyze the impact of these techniques on other synthesis and verification algorithms.

TABLE IV. BMC Depth Comparison

Benchmark Set	30-min BMC	Input Simp. + 29-min BMC	Trans. Simp. + Input Simp. + 29-min BMC
IBM (135)	100.00%	100.00%	102.20%
HWMCC (28)	100.00%	100.00%	103.68%
total (163)	100.00%	100.00%	102.39%

A. Relationship with Bounded Model Checking

First we focus on BMC. We compare three different flows:

1) 30-minute BMC
2) Initialization input simplification + 29-minute BMC
3) Transient simplification + initialization input simplification + 29-minute BMC

Note that the combination of transient simplification and initialization input simplification is restricted to run for no more than 20 seconds. This means that the total runtime for each of the three configurations should be approximately the same, with a slight advantage given to Flow 1.

Table IV compares the depths achieved by BMC before the timeout was reached. We normalize all depths to Flow 1 and observe that initialization input simplification alone was not effective in increasing the attained BMC depth. However, transient simplification was effective in increasing the depth by 2.4%. This indicates that BMC was wasting runtime analyzing the signals that were transient, and by removing these signals we enable BMC to do more work in the same amount of time.

B. Relationship with Interpolation

Interpolation [13] is an unbounded SAT-based proof technique. While effective on numerous complex designs, interpolation alone was insufficient to solve many of the benchmarks under analysis.

Table V illustrates the effect of running transient simplification and initialization input simplification before interpolation. Of the industrial designs, 10 designs (13.5%) which interpolation initially could not solve within a 30-minute timeout become solvable after the design is simplified using our proposed techniques. Unfortunately, there were also 4 designs (3.0%) that were previously solvable but are not

TABLE V. Interpolation Comparison

	Transient + Input Simp.	
Benchmark Set	Enables Proof	Breaks Proof
IBM (135)	10	3
HWMMC (28)	0	1
total (163)	10	4

TABLE VI. Design Size after Signal Correspondence, On 163 Designs

Flow	Ands	Regs.
Baseline (after Comb. Synth.)	100.00%	100.00%
Signal Correspondence	76.23%	73.91%
Input Simp. + Signal Correspondence	75.03%	73.89%
Trans. Simp. + Input Simp. + Signal Correspondence	75.39%	72.21%

solvable after the design is simplified. This illustrates the inevitable instability of the underlying interpolation algorithm, yet we nonetheless are encouraged that our simplification helps interpolation substantially more often than it hurts. In a robust verification setting, it may be desirable to invoke interpolation both before and after our simplifications to maximize the chances of obtaining a conclusive result.

C. Relationship with Signal Correspondence

The detection and simplification of sequentially equivalent signals, sometimes referred to as *signal correspondence* is an effective way to reduce the size of a sequential design [14]. Often these reductions either prove the safety properties or are effective in simplifying the problem for another downstream verification engine.

Table VI shows how our proposed algorithms interact with signal correspondence on the combined set of IBM and HWMCC designs. Combinational synthesis is run on all benchmark designs, and all other design sizes are normalized to this baseline. Signal correspondence is able to reduce the number of Ands in an AIG representation of the design by 23.7% and the number of Registers by 26.1%. Running initialization input simplification prior to signal correspondence does not noticeably help the results, but running transient simplification followed by input simplification prior to signal correspondence is effective.

Using our simplification algorithms before signal correspondence enables signal correspondence to achieve 0.9% better And count and 1.7% better Register count. Note that our simplifications can at times increase the size of the design (due to an increase in the initialization logic, Section II-C). Despite this bloat, signal correspondence is able to achieve a smaller design size than it otherwise would without our simplifications.

D. Relationship with Retiming

Our proposed techniques are related to retiming [4] in that both approaches time-shift the design and, as a byproduct, entail complicated initial values. Table VII explores interleavings of these two techniques on the combined set of IBM and HWMCC designs.

The size of the designs after combinational synthesis is used as a baseline for comparison. Running min-area retiming twice in succession decreases the number of registers but increases all other size metrics, on average.

Running transient simplification and initialization input simplification instead of the first retiming pass slightly decreases the number of registers, but also increases the logic bloat because the initial values are being complicated by both techniques.

The best configuration is where the second retiming pass is replaced with transient simplification and initialization input simplification. This decreases the size of the design across all metrics: number of AIG Ands, number of Registers, and number of Inputs. There are two explanations for this:

TABLE VII. Design Size after Retiming, On 163 Designs

Flow	Ands	Regs.	Inputs
Baseline (after Comb. Synth.)	100.00%	100.00%	100.00%
Retiming + Retiming	116.81%	91.06%	174.56%
Algorithms 2,3 + Retiming	118.51%	90.85%	177.55%
Retiming + Algorithms 2,3	108.04%	90.73%	163.65%

- Retiming creates initialization inputs which our techniques are able to eliminate. To test this, we ran just initialization input simplification after retiming and observed our simplification technique reducing Ands by 0.3%, Registers by 0.1% and Inputs by 4.5%. This accounts for some of the reductions of Table VII.
- Transient signals remain in the post-retiming design which Algorithm 2 is able to eliminate.

E. Relationship with Sequential Equivalence Checking and Induction

We have found numerous cases where time-shifting and simplifying transient signals has been crucial to completing a proof by induction, and therefore our techniques are a vital part of the overall scalability of various verification methodologies. Particularly, in sequential equivalence checking (SEC), identification of internal points that are pairwise equivalent is critical to the successful completion of an inductive proof of input-output equivalence. Many of these pairs of internal points are initially inequivalent due to power-on non-determinism, and the inductive proof of input-output equivalence fails because the internal equivalences no longer hold. Numerous techniques have been proposed to decouple the verification of the reset mechanism from post-reset SEC, after which power-on inequivalence has been normalized out [2], but there exist many verification problems that do not benefit from these approaches.

In one particular industrial SEC problem with 804 Registers, 21,726 And gates, and 175 Inputs, the use of time-shifting to eliminate transient signals of duration 7 (due to an "initializing" state machine) followed by the subsequent enhanced synthesis optimization rendered a $k = 2$-inductive sequential equivalence check with a combined overall runtime of 42 seconds. Without the use of our transient-elimination algorithms, induction could not solve the problem even within a runtime of 2 hours and depth $k = 12$.

In another industrial SEC example with 3201 Registers, 19,506 And gates, and 843 Inputs, a 6-frame time-shift to eliminate an "initializing" state machine transients enabled a $k = 1$-inductive sequential equivalence check using a combined runtime of 56 seconds. Without the elimination of transients, the problem was not k-inductive even with depth 5; our only solution to this problem relied upon the use of substantially heavier-weight algorithms such as abstraction and interpolation, and required manual tuning to solve.

As discussed in Section IV-D, transient elimination is related to peripheral retiming in that both techniques can time-shift the design. However, we have found transient elimination

to be more desirable than retiming in certain facets of SEC flows in that the scalability of SEC relies to some extent upon name- and structure-based correlation of registers across the designs being equivalence-checked. Retiming may arbitrarily alter register placement, diminishing the usability of such SEC heuristics.

V. Related Work

Techniques presented in this paper are similar to Kth invariants [15]. In the previous work, equivalences that hold sequentially in all time frames after time K are found by induction and leveraged in unbounded verification of safety properties. This work differs from our own in the following ways:

- In [15], leveraging Kth-invariants requires a specialized verification algorithm that performs backward reachability. Our technique transforms the design such that any downstream verification algorithm in our TBV flow can leverage the information.
- The previous work assumed that K was an input parameter to be set manually. This corresponds to the maximum transient duration in our own work and is automatically discovered.
- Kth-invariants are sequential equivalences that hold after time K, proved with induction which is similar to [14]. We look for transients that are constant after a finite number of time steps and leverage ternary simulation. This means that we likely find fewer transients but are much more scalable. All results presented in this paper were obtained with less than 20 seconds of runtime spent in our proposed simplification steps.

Despite the fact that transients are stronger yet less numerous than Kth-invariants, our approach subsumes [15]. We can time-shift the design by K frames and then pass the design to a general-purpose signal correspondence implementation to find equivalences that hold in all reachable states of the shifted design. This uncovers all Kth-invariants and subsumes the work of [15]. This also explains why simplifying transients and time-shifting the design was beneficial in enhancing the signal correspondence results in Table VI.

Our work is also related to [16] where a sequence of unrolled time frames is used to generate ODC conditions that can be used to simplify the transition relation. [16] asserts that they are simplifying logic that is only used for the initialization of the design, similar to our transients. The prior work was aimed at synthesis and described several exotic scenarios under which the transition relation could be simplified while retaining the correct circuit behavior. Our work differs in that we detect transients and replace them with constants without any need for expensive ODC analysis. We are also aimed at simplifying problems for verification, and in such an environment we do not need to strictly preserve the initialization behavior after it is verified with BMC. However, we do need to time-shift the design to be compatible with other algorithms, a problem that [16] does not address.

VI. Conclusion

This work is concerned with two types of redundant information present in RT-level designs: transient signals and initialization inputs. Algorithms to identify and remove both phenomena were presented, and these algorithms were effective in efficiently simplifying our designs.

The proposed algorithms were implemented in an industrial verification environment as a light-weight design simplification step prior to the invocation of heavy-weight formal verification algorithms. This setup allowed us to examine the relationship of our techniques to existing synthesis and verification algorithms in detail:

- Our techniques increase the number of time frames checked by BMC in a 30-minute timeout by 2.4%.
- Many of the safety properties in the designs we examined were not provable by interpolation. After our simplifications an additional 13.5% of these properties were provable with interpolation.
- Applying our simplifications before signal correspondence allowed for a 0.9% better reduction in Ands and 1.7% better reduction in Registers.
- Our simplifications were effective in simplifying designs after min-register retiming, reducing Ands by 8.1%, Registers by 0.3%, and Inputs by 10.9%.
- Applying our simplifications as a pre-processing step was vital to the completion of an inductive proof of input-output equivalence on large industrial designs.

These results demonstrate that our methods to simplify transient signals and initialization inputs are effective tools in a TBV verification environment.

References

[1] S. Hazelhurst, O. Weissberg, G. Kamhi, and L. Fix, "A Hybrid Verification Approach: Getting Deep into the Design," in *DAC* 2002.

[2] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G/ Janssen, "Scalable Sequential Equivalence Checking across Arbitrary Design Transformations", in *ICCD* 2006.

[3] C. Jacobi, K. Weber, V. Paruthi, and J. Baumgartner", "Automatic Formal Verification of Fused-Multiply-Add FPUs", in *DATE* 2005.

[4] A. Kuehlmann and J. Baumgartner, "Transformation-based verification using generalized retiming," in *CAV* 2001.

[5] P. Bjesse and J. Kukula, "Automatic Generalized Phase Abstraction for Formal Verification", in *ICCAD* 2005.

[6] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded Model Checking Using Satisfiability Solving," in *FMSD* 2001.

[7] "Hardware Model Checking Competition 2008: Benchmarks," `http://fmv.jku.at/hwmcc08/benchmarks.html`

[8] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman and A. Kuehlmann, "Scalable Automated Verification via Expert-System Guided Transformations," in *FMCAD* 2004.

[9] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A.R. Wang, "MIS: A Multiple-Level Logic Optimization System," in *TCAD* 1987.

[10] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *DAC 1997*.

[11] A. Mishchenko, S. Chatterjee, and R.K. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *DAC* 2006.

[12] A. Biere, "The AIGER And-Inverter Graph (AIG) Format Version," `http://fmv.jku.at/aiger/FORMAT.aiger`, 2007.

[13] K.L. McMillan, "Interpolation and SAT-Based Model Checking," in *CAV* 2003.

[14] C.A.J. van Eijk, "Sequential equivalence checking based on structural similarities," in *TCAD* 2000.

[15] F. Lu and K.T. Cheng, "Sequential Equivalence Checking Based on K-th Invariants and Circuit SAT Solving," in *HLDVT* 2005.

[16] N. Kitchen and A. Kuehlmann, "Temporal Decomposition for Logic Optimization," in *ICCAD* 2005.

[17] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't-cares," in *DAC* 2006.