2015 IEEE Computer Society Annual Symposium on VLSI

Equivalence Checking using Trace Partitioning

Rajdeep Mukherjee
University of Oxford

Daniel Kroening
University of Oxford

Tom Melham
University of Oxford

Mandayam Srivas
Chennai Mathematical Institute

***Abstract*—One application of equivalence checking is to establish correspondence between a high-level, abstract design and a low-level implementation. We propose a new partitioning technique for the case in which the two designs are substantially different and traditional equivalence-point insertion fails. The partitioning is performed in tandem in both models, exploiting the structure present in the high-level model. The approach generates many but tractable SAT/SMT queries. We present experimental data quantifying the benefit of our partitioning method for both combinational and sequential equivalence checking of difficult arithmetic circuits and control-intensive circuits.**

I. Introduction

When a new device is designed, a "golden model" is often written in a high-level programming language such as ANSI-C, C++ or SystemC. This model is extensively simulated to ensure both correct functionality and that performance targets are met. Later, a Verilog or VHDL implementation is created. It is essential to check that the C and the Verilog models are consistent. High-level models come in a broad variety of modeling styles, which affects the choice of algorithm for performing the consistency check. A combinational equivalence checker is sufficient for a pair of models that are cycle accurate and have the same set of state-holding elements. But when the high-level model is not cycle-accurate or has a substantially different set of state-holding elements, a sequential equivalence checker is required [1], [2].

In this paper, we address the most general case and thus most difficult variant of equivalence checking: we consider the case in which the high-level and the low-level design are substantially different. In this scenario, methods that rely on equivalence points [3] are not very effective, and the equivalence checking problem becomes a general Hardware/Software (HW/SW) co-verification problem.

A widely used method to do formal HW/SW co-verification is to apply symbolic simulation to both models, and then merge the resulting formulas. The low-level model is usually provided in register-transfer level (RTL) Verilog and the high-level models are written in ANSI-C or SystemC. Bounded model checking (BMC) using propositional SAT or satisfiability modulo theories (SMT) can be easily applied to both RTL and C-style languages [4], [5]. Figure 1 illustrates the flow of a bounded co-verification tool, HW-CBMC. Here, the transition relations for both models are unwound to obtain two formulas, which are then conjoined and checked for satisfiability using an efficient SAT or SMT procedure. The BMC approach relies on a user-specified unwinding bound; if the bound is too small, BMC is limited to bug-finding.

Supported by EPSRC EP/J012564/1, ERC project 280053 and the Semiconductor Research Corporation (SRC) under task 2269.001.

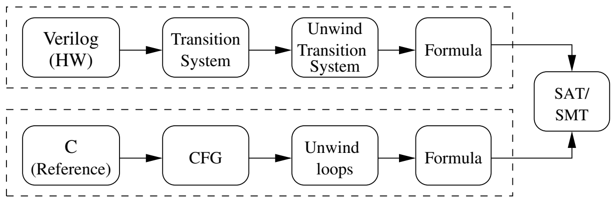

Fig. 1: Equivalence checking tool flow

In practice, equivalence checking is limited to block-level designs and does not scale up to IP-level or SoC-level designs. State-of-the-art industrial equivalence checkers such as HECTOR (Synopsys) [6] and SLEC (Calypto)[1], use combinations of bit-level and word-level solvers to show equivalence between C and RTL models. Both HECTOR and SLEC exploit structural similarities present in the input designs to simplify the equivalence proofs. Thus, they strongly rely on word-level proof engines and aggressive pre-processing/rewrite engines to exploit the similarities present in the word-level formulas derived from two input models. Furthermore, in order to deal with difficult instances, they support manual case-splitting of the design—for example input-based case-splitting or slicing.

In this paper, we experimentally evaluate the benefit of search-space partitioning for both combinational equivalence checking (CEC) and sequential equivalence checking (SEC). Case-splitting followed by slicing is the most commonly used simplification technique to decompose harder proofs into simpler sub-proofs for equivalence checking for block level circuits [7], [8], [6]. In the case of combinational equivalence checking, partitioning is usually straight-forward: it is not difficult for the engineer to split up the set of inputs into suitably easy sub-cases.

Partitioning is, however, more subtle in the case of sequential designs. We use transaction or scenario-based partitions for our sequential benchmarks. We do not rely on information from a high-level synthesis stage to guide the equivalence verification and also do not make use of simplifying assumptions about structural similarity of the two models.

To obtain broadly valid experimental data, we target two different classes of circuits: data-path intensive designs, exemplified by circuits for floating-point arithmetic, and control-path intensive designs, which are represented by a USB physical IP core and an implementation of an Ethernet MAC controller IP.

Contributions: We address the broad area of "symbolic co-simulation" of two models, which are typically a high-level

[1]http://calypto.com/en/products/slec/

reference design and a low-level implementation. We recognize that the existing symbolic methods do not scale to the full size of complex hardware IPs. This paper makes two main contributions that address this problem:

- We present a novel transaction/scenario-based trace partitioning technique for efficient symbolic co-simulation of hardware descriptions (in Verilog) together with a high-level reference model (in C). The technique is applicable to both combinational and sequential equivalence checking problems. The technique is aimed at the most difficult variant of equivalence checking in which the high-level reference design and the low-level implementation design are substantially different and traditional equivalence-point insertion technique fails.
- We experimentally evaluate the benefit of state-space partitioning for combinational and sequential equivalence checking of complex hardware IPs. To this end, we experimentally evaluate the performance of the word-level bounded equivalence checker, HW-CBMC, with and without partitioning. We benchmark the alternative approaches using a wide variety of circuits, ranging from difficult floating point arithmetic circuits to complex control-path intensive circuits exemplified by *USB PHY IP core* and *Ethernet MAC IP*.

II. Background

This section introduces formalism for program traces, and reviews how to construct a miter for checking equivalence of a hardware and a software model, and briefly introduces Bounded Model Checking.

A. Traces

The semantics of program statements is defined relative to an environment function that maps program variables to their values, $Env = Var \to Val$. A statement s defines a function $\delta_s : \wp(Env) \to \wp(Env)$. A *state* is a location n with an environment ε_i. A *trace* is a sequence of states $(n_0, \varepsilon_0), \ldots, (n_k, \varepsilon_k)$ such that for all $0 \le i < k$, (n_i, n_{i+1}) is a control-flow graph (CFG) edge and $\varepsilon_{i+1} \in \delta(n_i, n_{i+1})(\{\varepsilon_i\})$. A program is *safe* if there is no trace with $n_0 = init$ and $n_k = ↯$.

B. Miters for HW/SW Equivalence Checking

A miter circuit is built from two given circuits A and B as follows: identical inputs are fed into A and B, and the outputs of A and B are compared using a comparator. We consider the case in which one of the circuits is a software program. Figure 2 shows an example miter for checking combinational equivalence of a 32-bit floating-point adder/subtractor circuit. We provide the same floating-point numbers as inputs to the reference design (in C) and the hardware implementation (in RTL Verilog) using a function `set_inputs()`. Subsequently, we indicate that we want to perform a floating-point addition by setting `isAdd=1`. The results computed by the hardware design and the C reference model are compared using the `compareFloat()` function.

```
void miter(float f, float g) {
  // setting up the inputs to hardware FPU
  fp_add_sub.f = *(unsigned*)&f;
  fp_add_sub.g = *(unsigned*)&g;
  fp_add_sub.isAdd = 1;
  // propagates inputs of the hardware circuit
  set_inputs();
  // get result from hardware circuit
  float Verilog_result = *(float*)&fp_add_sub.result;
  // compute fp-add in Software with rounding mode RNE
  float C_result = add(RNE,f,g);
  // compare the outputs
  assert(compareFloat(C_result,Verilog_result));
}
```

Fig. 2: Miter for combinational equivalence checking for a 32-bit floating-point adder/subtractor for the case of addition

C. Bounded Model Checking

Bounded Model Checking (BMC) [5] can be briefly sketched as follows. Given a depth k and a set of error states F, BMC operates by unwinding the transition relation T up to depth k starting from initial state x_0, represented by an initial state predicate I. This forms the following formula:

$$I(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-1}, x_k) \wedge (F(x_1) \vee \ldots \vee F(x_k))$$

A SAT solver is used to determine if the formula is satisfiable. If so, there exists an error trace of length at most k and the procedure terminates, reporting the error. Otherwise, the property holds true on the transition system up to the bound of k. The main challenge of this approach is scalability: the basic SAT/SMT approach works reasonably well up to the module level, but generates instances that are too large for bigger verification tasks. We present a technique for generating many but easy SAT/SMT instances to overcome this problem.

III. Equivalence Checking with Trace Partitioning

Previous research has shown that case splitting and slicing make bit-level combinational equivalence checking easier [7], [8], [6]. The approach is straightforward to implement. But partitioning is more subtle in case of sequential designs. This section discusses the various trace partitioning techniques for application in the setting illustrated in Figure 3.

A. Trace Partitioning

Trace partitioning [9] is a method for independently analysing sets of program traces generated by a suitably chosen partitioning function, and was introduced in the context of abstract interpretation [10] as a way to increase the precision of a given abstract domain for a single program.

By contrast, we apply trace-driven partitioning in sequential equivalence checking with the goal to split the state space in a way that enables sufficiently tractable analysis of each of the partitions with the final SAT solver calls. The effort required for analysis increases linearly with the number of partitions generated by the partition function, and exponentially with the size of the partitions. The goal is thus a partitioning function that yields partitions of essentially constant difficulty.

The idea of trace partitioning is to distinguish traces using a function $\alpha : K \to \wp(S^*)$ that maps some set of tokens K to

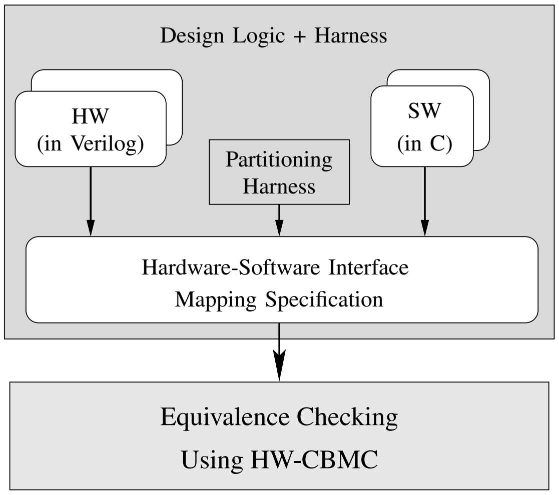
Design Logic + Harness
HW (in Verilog)
Partitioning Harness
SW (in C)
Hardware-Software Interface Mapping Specification
Equivalence Checking Using HW-CBMC

Fig. 3: Equivalence checking using trace partitioning

sets of traces S^*. We consider two extreme cases. If $|K| = 1$ and $\alpha(K) = S^*$, then there is no discrimination between traces. On the other hand, if $|K| = S^*$ and $\alpha(k) = id_{S^*}$ for $k \in K$ and *id* is the identity function, then each trace is considered in isolation and an analysis with this partition is basically an explicit exploration of all program traces. In practice, we aim at a partition that falls between the two extremes. We systematize the search for a suitable partition using *trace partitioning templates*, described next.

B. Trace Partitioning Templates

The choice of partition function helps to fine-tune the scalability of the analysis by performing coarser splitting where possible and only increasing the number of partitions where necessary. There exists several partitioning techniques in the literature [11], [12], [7] that can be used to partition the state-space of the design. In this paper, we present our experimental results based on input-based partition for CEC and transaction/scenario-based partition for SEC, which are described next.

Value, control-flow and length-based Partitioning: Value-based trace partitioning restricts the range of a variable at one or more program locations. For example, if x is a program variable, then the partition of the traces may depend on grouping the input values of the variable x into three separate cases $\{x < 0, x > 0, x = 0\}$. *Control-flow based trace partitioning* distinguishes traces according to control-flow history. Further, *length* or *parity* based trace partitioning is based on the length of a trace. In this case, the partition function is given by $\alpha : \{0,1\} \to S^*$, where $\alpha(0)$ corresponds to those traces that execute the loop an even number of times and $\alpha(1)$ corresponds to those traces that execute the loop an odd number of times.

Input-based Partitioning: Input-based trace partitioning is a special case of value-based trace partitioning. To illustrate input-based partitioning [7], [8], [6], let us revisit the harness given in Fig. 2. The `CPROVER_assume(c)` statement instructs the HW-CBMC tool to restrict the analysis to only those paths satisfying a given condition `c`. We can limit the analysis to those paths that are exercised by inputs where the rounding mode is nearest-even (RNE) and both input numbers are NaNs by adding the following statements:

```
CPROVER_assume(fp_add_sub.roundingMode==RNE);
CPROVER_assume(fp_add_sub.uf_nan);
CPROVER_assume(fp_add_sub.ug_nan);
```

Transaction/scenario-based Partitioning: We propose a novel partition technique, known as transaction/scenario-based trace partitioning, to systematize the search-space partitioning for scalable equivalence verification of complex IP designs. Let us consider a model M with input signals $\{I\}$ and output signals $\{O\}$. Here, a *transaction* can be defined by two events:

1) **Initiation event**, *init* – a condition on a subset of input signals, $\{I\}$.
2) **Completion event**, *comp* – a condition on a subset of output signals, $\{O\}$.

Both events, *init* and *comp*, can be possibly spread over time. But, assuming the duration of the transaction event is finite, there must exist some distinguishable last completion event. Now, recall the definition of a *trace* which is a sequence of states, $\{(n_0, \varepsilon_0), \ldots, (n_k, \varepsilon_k)\}$. For the purpose of illustration, let us assume that the $Var \in \varepsilon_0$ corresponds to I and $Var \in \varepsilon_k$ corresponds to O. A *transaction* can then be defined as a projection over a legal trace T of a model M. In other words, a *transaction* is an occurrence of a finite series of events of particular pattern satisfying *init* and *comp* and can be represented as a path constraint. A transaction-based partitioning is an enhancement of input-based case splitting. The reason being that a transaction may involve some input signals, output signals as well as some internal states of the model, as for example, device configuration registers.

A use-case scenario is defined as either a particular finite sequence of transactions or an infinite sequence of transactions satisfying a recurring pattern. This can be manifested in the form of a state machine or using a regular expression. Thus a *scenario* is characterized by the set of allowable control-flow paths in a model.

Figure 4 presents a fragment of a high-level model (in C) and two transactions. The high-level model consists of a single outer *while* loop that encodes complex control flow and interacts with the environment. This code fragment implements a high-level power management strategy in an IP, say a transmitter (*trans*). Assuming that only a small fragment of the logic is enabled inside *trans* module during power gating (i.e. *buf_out* is "don't care"). Thus, partitioning using transaction 1 (*voltage_level* == 10) simplifies the logic for the non-power gated components inside *trans* through the use of assumptions, which restricts the tool to those set of paths satisfying transaction 1 thereby making the final SAT solver query tractable. On the other hand, transaction 2 partitions the design to those set of paths which involves only the normal logic (*voltage_level* == 20) and prunes the logic involved for power gating and other modes of operation (e.g. TURN_OF, TURN_ON, STAND_BY). These transactions are fed to the tool as an assumption to partition the verification state-space.

ANSI-C	Transactions
	$Transaction1:$ $(reset == 0) \wedge (env == 1)$ $\wedge(mode == STAND\_BY)$ $\wedge(voltage\_level == 10)$ $Transaction2:$ $(reset == 0) \wedge (env == 1)$ $\wedge(mode == HIBERNATE)$ $\wedge(voltage\_level == 20)$

```
#define threshold 15
if(reset) {
 mode=0,turn=0;
 feedback=0;
}
else { // code fragment for IP
// Trigger IP if env is set
if(env) {
 turn = 1; mode = 1;
 // check the voltage level
 if(voltage_level < threshold)
  power_gated = 1;
 else power_gated = 0;
 // check the low-power modes
 if(mode == STAND_BY ||
 mode == TURN_OFF) {
 // power gated logic.
 if(power_gated) {
  trans(reset,mode,power_gated
  ser_in,&buf_out);
  feedback = LOW; // no transmission
 }
 else { // normal logic
  trans(reset,mode,power_gated
  ser_in,&buf_out);
  feedback = buf_out;
 }} }
```

Fig. 4: High-level model in C and some transactions

IV. Experimental Results

We report experimental results for equivalence checking of two different classes of circuits. We perform trace-partitioning experiments on data-path intensive circuits, namely single-precision and double-precision floating point arithmetic circuits. As representatives for control-path intensive circuits we use an *USB_PHY IP* core and *Ethernet MAC IP* from opencores[2]. To enable other researchers to reproduce our results, all benchmarks are available online at www.cprover.org/hardware/partitioning-isvlsi/.

Data-path Intensive Benchmarks:

IEEE 754 Floating-point Arithmetic Circuits: We have developed both a C and a Verilog implementation of an IEEE-754 32-bit single-precision dual-path floating point adder/subtractor, which takes two 32-bit floating point numbers as input and returns a 32-bit floating point number as result. The floating-point design includes various modules like packing, unpacking, normalizing, rounding and handling of infinite, normal, subnormal, zero and NaN. We used Softfloat[3] as our reference implementation. Softfloat is a well-known software implementation of the IEC/IEEE standard for binary floating-point arithmetic.

Control-path Intensive Benchmarks:

USB_PHY IP: The USB 1.1 Physical Interface core IP provides all functions essential to interface to the USB 1.1 bus. This includes serial/parallel conversion, bit stuffing and unstuffing, NRZI encoding and decoding and a DPLL. The core USB PHY 1.1 supports the industry standard UTMI interface specification. The *phy_mode* signal in the core selects between single ended and differential *tx_phy* output. Currently, the PHY IP from opencores only operates in full speed mode. The required clock frequency is 48 MHz, from which the 12 MHz USB transmit and receive clocks are derived.

Ethernet MAC IP: The tri-mode Ethernet MAC implements a MAC controller conforming to IEEE 802.3 specification and supports serial PHY and parallel PHY interfaces. It also supports automated pause frame generation and termination as well as half-duplex for 10 and 100 Mbps mode. The default FIFO depth of the transmitter ($MAC\_TX\_FF\_DEPTH$) and the receiver ($MAC\_RX\_FF\_DEPTH$) is 9, which means that the FIFO can contain 512 words.

A. Experimental Setup

All our experiments were run on an Intel Xeon machine with 8 cores at 3.07 GHz with 48 GB RAM. All times in Table I and Table II are reported in seconds. CBMC[4] is used to perform bounded equivalence checking of C designs—the tool takes as input a C program with assertions. The tool HW-CBMC is built on top of CBMC. HW-CBMC is used for word-level bounded equivalence checking of designs at different levels of design abstraction—C-RTL models and RTL-RTL models. The tool takes as input a C program and a Verilog RTL implementation. The RTL hardware description is unrolled for each program step using a function call `next_timeframe()`. Other C-RTL and RTL-RTL SEC tools are HECTOR and SLEC, but these tools were not easily obtainable by us for experimentation.

Table I presents the timings for equivalence checking of IEEE-754 single-precision and double-precision floating point arithmetic circuits. Columns 1–4 give the name of benchmark, design sizes for reference and implementation models, and the total time for equivalence checking without partitioning, respectively. Columns 5–10 in Table I report the CEC time with input-based case splitting. Note that the total time in Table I indicates the time required to verify all possible combinations of the input numbers and roundingMode. The timeout is set to 16 hours for CEC. We note that the verification of the single-precision and double-precision multiplier and divider circuit did not terminate without partitioning.

Table II reports the effect of search-space partitioning for sequential equivalence checking of hardware IPs using HW-CBMC. Columns 1–3 give the name of benchmark and the size of the reference and implementation models, respectively. Columns 4–6 give the bound up to which the hardware transition system is unrolled, the unwinding limit for the reference design and the time taken by SEC without partitioning. Columns 7–8 give the run-times when using two different partition harnesses, TP1 and TP2, respectively. The timeout is set to 7 hours for SEC.

B. Discussion

HW-CBMC implements both bit-level and word-level equivalence checking engines. We observe that the bit-level equivalence checker performed badly when compared to the word-level equivalence checker, both for data-path and control-path intensive circuits. This holds true irrespectively of the back-end solver engines used. The default SAT solver engine used is MiniSAT 2.2.0[5]. HW-CBMC also supports different

[2]http://opencores.org/
[3]http://www.jhauser.us/arithmetic/SoftFloat.html

[4]http://www.cprover.org/cbmc/
[5]http://minisat.se/

backend SMT solvers, namely Z3, MathSAT, CVC4 and Yices. Our experience with word-level equivalence checkers suggests that SAT solvers still outperform SMT solvers, especially when the input models are substantially different. Thus, we report our results using word-level equivalence checker with MiniSAT 2.2.0. It is worth emphasising that bit-level reasoning engines can benefit from word-level input, as custom clause-level encodings into CNF can be used.

Combinational Equivalence Checking: In Table I, we report the results when using input-based case splitting based on rounding mode, *(RNE)*, and types of numbers *(subnormal, NaN, zero, infinity, normal)*. The top three rows reports C-RTL equivalence checking with three different reference implementations, *Dual-path*, *CBMC* and *Softfloat*. Note that the C and RTL designs are structurally different and thus techniques which rely on equivalence points fail [7], [8]. However, HW-CBMC is able to handle arbitrary designs and proves equivalence in a reasonable amount of time. The dominant times are required for the case of normal and subnormal numbers.

We highlight that trace partitioning has enabled us to verify the equivalence of single precision and double-precision floating-point multiplier and divider circuits, when, by contrast, the verification of these complex arithmetic circuits exceed the capacity of the state-of-the-art SAT solvers due to complex data-path logic. Our result in Table I reconfirms previous observations that input-based case splitting is an effective partitioning technique for block-level arithmetic circuits [7], [8], [6].

Sequential Equivalence Checking: Compared to the input-based case-splitting that is used to partition the search-space for combinational equivalence checking of block level arithmetic circuits, the partitioning for sequential hardware IPs is more subtle. Here, we perform partitioning based on transactions or what could be interpreted as a "use-case scenario". Consequently, the modular and hierarchical design structure of these IPs makes it easier to identify suitable transaction/scenario pairs for effective partitioning of the design. We now briefly discuss the various partitions used for our sequential benchmarks.

The modular structure of *USB_PHY IP* helped us to partition its operations based on different use-case scenarios in *transmitter* and the *receiver* module, which are identified as *TP1* and *TP2* respectively in Table II. *TP1* checks equivalence of the *DPLL* and the *NRZI Encoder* logic and equivalence of the output enable logic and output registers in the *transmitter* module. *TP2* performs equivalence checking of the *NRZI decoder* logic and the *serial to parallel converter* logic in *USB_PHY receiver* module. For the Ethernet MAC IP, *TP1* and *TP2* partition the design state-space based on different operation modes such as promiscuous (transparent) and non-promiscuous (filtered) operation, respectively.

It is important to note that the high-level use-case scenarios derived from the reference design (in software) are used to partition the verification space in hardware (RTL implementation). We emphasize that the discovery of a transaction/scenario pair is non-trivial. However, our experience with the application of different partitioning techniques (input-based and transaction/scenario based) with HW-CBMC tells us that the control paths in the reference model can be exploited to derive suitable transaction/scenario pairs for effective partitioning of the design state-space in sequential equivalence checking. Our experimental results show the efficacy of this partition technique for equivalence checking on our benchmarks.

V. Related Work

Over the past years, there have been several works that focus on equivalence checking between designs at different abstraction levels [13], [1], [2], [3], [6]

The work of [14], [6] proposed a theoretical framework for checking equivalence between system-level designs against an RTL design. All these techniques are tuned for the case of equivalence verification of structurally identical designs [7], [8]. Further, to scale up SEC, trace partitioning [9] has so far been usually performed in an ad-hoc fashion, the most simplest form of which is case-splitting followed by slicing [11], [7].

VI. Conclusion

In this paper, we experimentally evaluated the benefit of state-space partitioning for both combinational and sequential equivalence checking of two arbitrary input designs—the hardware descriptions (e.g., given in Verilog) together with a high-level reference model (in C). The key to scalability for SAT-based symbolic co-simulation techniques is to perform partitioning of both models, resulting in multiple but manageable queries for today's SAT/SMT solvers. Our experimental results show that equivalence checking with transaction/scenario-based partitioning can scale up existing SEC tools. We also experimentally evaluated the benefit of input-based case-splitting for CEC of complex arithmetic circuits. In future, we plan to developing a technique that effectively performs such partitioning in an automated way that is inspired by the decision heuristics and deduction methods in modern propositional SAT solvers.

References

[1] C. A. J. van Eijk, "Sequential equivalence checking without state space traversal," in *Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, 1998, pp. 618–623.

[2] J. Baumgartner, H. Mony, V. Paruthi, R. Kanzelman, and G. Janssen, "Scalable sequential equivalence checking across arbitrary design transformations," in *International Conference on Computer Design (ICCD)*. IEEE, 2006, pp. 259–266.

[3] W. Wu and M. S. Hsiao, "Mining global constraints for improving bounded sequential equivalence checking," in *Design Automation Conference (DAC)*. ACM, 2006, pp. 743–748.

[4] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, pp. 117–148, 2003.

[5] E. M. Clarke, D. Kroening, and F. Lerda, "A tool for checking ANSI-C programs," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 2988. Springer, 2004, pp. 168–176.

[6] A. Kölbl, R. Jacoby, H. Jain, and C. Pixley, "Solver technology for system-level to RTL equivalence checking," in *Design, Automation and Test in Europe (DATE)*. IEEE, 2009, pp. 196–201.

[7] B. Xue, P. Chatterjee, and S. K. Shukla, "Simplification of C-RTL equivalent checking for fused multiply add unit using intermediate models," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2013, pp. 723–728.

[8] M. Fujita, "Verification of arithmetic circuits by comparing two similar circuits," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 1102. Springer, 1996, pp. 159–168.

Circuit	# Lines of code		Combinational Equivalence Checking With Trace Partitioning						
			Without Partitioning	With Input-based Partitioning (Verification Time in Seconds)					
	Reference Model	Implementation Design	Total Time	Subnormal	Infinity	Zero	NaN	Normal	Total Time
	CBMC	RTL	**C versus RTL Equivalence Checking (Single-Precision Dual-Path versus CBMC Adder)**						
32-bit FP Adder	13	680	309.5	81.3	2.1	2.3	1.8	9.8	315.6
	Dual-path	RTL	**C versus RTL Equivalence Checking (Single-Precision Dual-Path Adder)**						
32-bit Adder	650	680	535.8	171	8.1	8.0	7.9	297.4	546.2
	Softfloat	RTL	**C versus RTL Equivalence Checking (Single-Precision Dual-Path versus Softfloat Adder)**						
32-bit FP Adder	1847	680	623.5	61.2	3.7	3.5	3.4	28.5	618.7
	RTL	Optimized RTL	**RTL versus RTL Equivalence Checking (Dual-Path versus Optimized Dual-Path Adder)**						
32-bit FP Adder	680	715	13.8	2.4	2.3	2.7	2.4	2.3	14.2
	C	C	**C versus C Equivalence Checking (CBMC versus SoftFloat versus Dual-Path Adder)**						
Dual-Path vs. CBMC	756	13	560.7	159.1	0.5	0.6	0.5	430.8	598.5
Softfloat vs. CBMC	1847	13	24.9	1.6	0.5	0.5	0.5	22.9	29.3
Softfloat vs.Dual-Path	1847	756	1145.8	98.7	0.5	0.5	0.5	1067.5	1174.8
	Softfloat	Slowfloat	**C versus C Equivalence Checking (Single-Precision Softfloat versus Slowfloat)**						
32-bit FP Multiplier	2136	1045	Did not Terminate	0.8	0.5	0.4	0.5	29866.3	29997.5
32-bit FP Divider	2369	1114	Did not Terminate	26789.4*	0.9	0.8	0.8	54341.7*	81733.6*
	Softfloat	Slowfloat	**C versus C Equivalence Checking (Double-Precision Softfloat versus Slowfloat)**						
64-bit FP Adder	2309	1174	4449.7	387.1	1.6	1.7	33.8	4056.2	4492.4
64-bit FP Multiplier	2383	1186	Did not Terminate	2.1	1.6	1.5	1.7	38764.5*	38824.4*

TABLE I: Run times for combinational equivalence checking for IEEE 754 floating-point arithmetic circuits (runs marked with * are limited to RNE+Type+Even+Odd, the timeout was set to 16 hours)

Circuit	# Lines of code		Sequential Equivalence Checking With Trace Partitioning				
			Bound Depth	Unwind Limit	Without Partitioning	With Transaction/Scenario based Trace Partitioning	
	Reference Model	Implementation Design			Total Time	TP1	TP2
	C	RTL	**C versus RTL Equivalence Checking**				
SERIAL ADDER	47	62	10	–	0.9	0.9*	0.9*
PIPELINED ADDER	52	68	10	–	0.8	0.8*	0.8*
ETHERNET MAC IP	1050	1200	100	15	Did not Terminate	21578.9	22451.3
USB_PHY IP	860	950	200	50	Did not Terminate	22681.4	21822.3
	RTL	Optimized RTL	**RTL versus RTL Equivalence Checking**				
SERIAL ADDER	62	54	10	–	0.6*	0.6*	0.6*
PIPELINED ADDER	68	58	10	–	0.5*	0.5*	0.5*
ETHERNET MAC IP	1200	1220	500	30	1384.5	956.2	948.7

TABLE II: Run times for sequential equivalence checking for control-intensive circuits ($*$ – no partitioning possible, the timeout set was to 7 hours)

[9] X. Rival and L. Mauborgne, "The trace partitioning abstract domain," *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 5, 2007.

[10] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Symposium on Principles of Programming Languages (POPL)*. ACM, 1977, pp. 238–252.

[11] C. Karfa, D. Sarkar, and C. Mandal, "Verification of datapath and controller generation phase in high-level synthesis of digital circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 3, pp. 479–492, 2010.

[12] L. Liu and S. Vasudevan, "Scaling RTL property checking using feasible path analysis and decomposition," in *Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2013, pp. 173–178.

[13] E. M. Clarke, D. Kroening, and K. Yorav, "Behavioral consistency of C and verilog programs using bounded model checking," in *Design Automation Conference (DAC)*. ACM, 2003, pp. 368–371.

[14] Z. Khasidashvili, M. Skaba, D. Kaiss, and Z. Hanna, "Theoretical framework for compositional sequential hardware equivalence verification in presence of design constraints," in *International Conference on Computer-Aided Design (ICCAD)*. IEEE/ACM, 2004, pp. 58–65.