Exploiting Reversible Computing for Verification

Potential, Possible Paths, and Consequences

Lukas Burgholzer
Institute for Integrated Circuits,
Johannes Kepler University Linz, Austria
lukas.burgolzer@jku.at

Robert Wille
Chair for Design Automation,
Technical University of Munich, Germany
Software Competence Center Hagenberg GmbH, Austria
robert.wille@tum.de

ABSTRACT

Today, the verification of classical circuits poses a severe challenge for the design of circuits and systems. While the underlying (exponential) complexity is tackled in various fashions (simulation-based approaches, emulation, formal equivalence checking, fuzzing, model checking, etc.), no "silver bullet" has been found yet which allows to escape the growing verification gap. In this work, we entertain and investigate the idea of a complementary approach which aims at exploiting reversible computing. More precisely, we show the potential of the reversible computing paradigm for verification, debunk misleading paths that do not allow to exploit this potential, and discuss the resulting consequences for the development of future, complementary design and verification flows. An extensive empirical study (involving more than 30 million simulations) confirms these findings. Although this work cannot provide a fully-fledged realization yet, it may provide the basis for an alternative path towards overcoming the verification gap.

1 INTRODUCTION

Verification is an essential step of today's design flow for circuits and systems which checks whether an obtained implementation realizes the desired specification or not. In the past decades, several approaches have been developed which include, besides many others, simulation-based verification (see, e.g., [1]–[5]), emulation (see, e.g., [6], [7]), formal equivalence checking (see, e.g., [8]–[11]), fuzzing (see, e.g., [12], [13]), and model checking (see, e.g., [14], [15]).

However, all of these approaches suffer from the steadily increasing complexity of circuits triggered by the accomplishments of the semiconductor industry in further miniaturizing feature sizes and increasing transistor counts. In fact, the number of transistors that can be physically implemented on a chip still grows faster than the ability to fully exploit them during the design and, especially, to (efficiently) verify the resulting circuits (commonly known as *verification gap*).

Thus far, no "silver bullet" has been found for this yet and there is a certain chance that we might not be able to escape this situation by solely focusing on iteratively improving existing methods for verification. Instead, in addition to the current state of the art, we most likely will need further, more complementary changes on the major pillars of today's design and verification algorithms.

In this work, we entertain such an approach and investigate whether the concept of *reversible computing* might provide such a complementary alternative. Reversible computing is a computing paradigm which is solely based on bijective operations, i.e., reversible n-input n-output functions that map each possible input pattern to a unique output pattern. As a result, in circuits based on reversible logic all computations can be reverted (i.e., the inputs can be obtained from the outputs *and* vice-versa). Recently, reversible computing found great interest, mostly because of its application in quantum computing [16], but also in the domain of low-power design [17], encoders [18], [19], and adiabatic circuits [20], [21], where it is exploited that no information is lost in reversible computation.

We believe that, besides these applications, this paradigm also offers distinct characteristics to improve verification and potentially shows an alternative route towards escaping the verification gap. This is mainly motivated by observations that errors in reversible circuits often seem to be easier to detect than in classical circuits (discussed in more detail later in Section 2). Moreover, additional characteristics such as an increased controllability and observability of corresponding reversible circuits [22], [23] as well as recent findings in the verification of quantum circuits [24]–[26] (which, as stated above, share characteristics with reversible circuits) seem to support this idea. However, thus far, all this latent potential has not yet evolved into a true alternative.

In this work, we shed light on why this is the case and whether reversible computing indeed may provide an alternative or not. To this end, we first show the potential of reversible computing for the verification of circuits and systems in Section 2. Afterwards, we debunk misleading paths that actually do not allow to exploit this potential in Section 3—leading to a better understanding of the issue and an identification of what is needed in order to leverage the potential. Those findings are eventually confirmed by means of an intensive empirical study (including more than 30 million simulations), whose results are summarized in Section 4. Overall, this does not yet lead to a fully-fledged realization of this alternative verification flow, but may provide the basis for an alternative path towards overcoming the verification gap. Accordingly, we conclude this paper with a discussion of the consequences of these findings for the development of future, complementary design and verification flows in Section 5.

2 BACKGROUND & MOTIVATION

In order to provide the basis for this work, this section first reviews classical circuits and briefly sketches the key challenge in their verification thus far. Afterwards, we give a short introduction on the basics of reversible computing and illustrate the raw potential this alternative computing paradigm might offer for the verification of circuits.

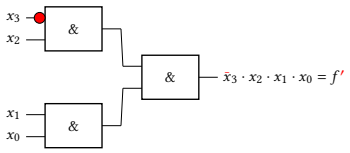

Fig. 1: Classical circuit realizing the functionality $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$ using three AND gates. A bit-flip error (denoted by ●) changes the overall functionality to $f'(\vec{x}) = \bar{x}_3 \cdot x_2 \cdot x_1 \cdot x_0$. This error can only be detected by 2 out of $2^4 = 16$ inputs.

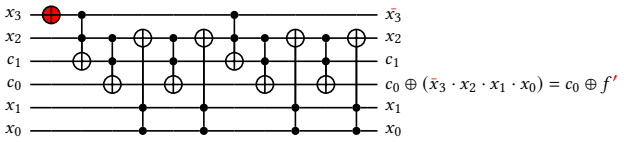

Fig. 2: Reversible realization of $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$ using ten TOFFOLI gates and six lines. The target function is realized on the fourth line when initializing c_0 to 0. A bit-flip error (now denoted by ⊕) on x_3 unconditionally affects the topmost output. *Every* input pattern allows to detect this discrepancy.

2.1 Classical Computing and Verification

As a basis for the discussion on classical circuits, we consider propositional or Boolean functions $f\colon \mathbb{B}^n \to \mathbb{B}^m$ over the variables $\vec{x} = (x_{n-1}, \ldots, x_0)$. These functions are typically realized as netlists of logic gates, such as AND, OR, XOR, etc.

EXAMPLE 1. *Consider the Boolean function f over four variables given by $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$. Then, Fig. 1 shows one possible circuit realization using three cascaded AND gates. The output f is equal to 1 if and only if $\vec{x} = 1111_2$.*

Such netlists are usually the result of sophisticated design flows in which the desired functionality is originally described using hardware description languages and, afterwards, transformed (or synthesized) through various stages of abstraction. During this process, it is of utmost importance that the original functionality is preserved throughout all levels of abstraction. Methods such as reviewed in Section 1 are applied to verify the resulting circuits and, by this, guarantee its correctness. They, however, frequently suffer from the underlying complexity of the problem and the hardness of efficiently detecting errors.

EXAMPLE 2. *Consider again the circuit from Fig. 1 and assume that a single bit-flip error (denoted by ●) affects input x_3. This alters the circuit's functionality to $f'(\vec{x}) = \bar{x}_3 \cdot x_2 \cdot x_1 \cdot x_0$, which obviously does not realize the originally intended function anymore. At the same time, however, this error is really hard to detect. In fact, in the (exponentionally) vast majority of possible input patterns, f and f' generate the same output pattern. Only for the input patterns $\vec{x} = 0111_2$ and $\vec{x} = 1111_2$, the error can be detected, i.e., only in 2 out of $2^4 = 16$ cases. This constitutes the main challenge for any approach checking the correctness of circuits and systems*[1].

2.2 Reversible Circuits

A Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ is called *reversible* if $n = m$ and each input pattern maps to a unique output pattern, i.e., a reversible function realizes a bijection or one-to-one mapping. Reversible functions can be realized by reversible circuits in which each variable of the function is represented by a circuit line. Fan-out and feedback are not allowed in reversible circuits since they would destroy the one-to-one characteristic of the reversible function. Consequently, reversible circuits are realized as a cascade of reversible gates. For the purpose of this work, we consider so-called TOFFOLI gates in the following, which are described by a set of control lines and a target line. A Toffoli gate inverts the value of the target line if and only if the values of *all* control lines are 1. For a general introduction, the reader is referred to [27]. This gate type alone already allows for universal computations, i.e., allows for realizing arbitrary (reversible) functionality [28].

EXAMPLE 3. *Fig. 2 shows a reversible circuit composed of ten TOFFOLI gates, where solid dots (●) denote control lines and ⊕ marks the target line of each TOFFOLI gate (the red components shall be ignored for the moment). This circuit realizes the Boolean function f from Example 1 in its fourth output when initializing c_0 to 0. If any of the input values of $\vec{x}$ is 0, the circuit effectively realizes the identity function on all its lines—leaving $f(\vec{x}) = 0$. Only if $\vec{x} = 1111_2$, the target line is inverted and evaluates to 1.*

2.3 Potential for Verification

As sketched in Example 2, it can be notoriously hard to detect errors in classical circuits—often only a handful of (out of the exponentially many) input patterns allow to detect certain discrepancies. This may change when reversible circuits are considered.

EXAMPLE 4. *Consider again the reversible circuit shown in Fig. 2 and assume that, as in Example 2, a single bit-flip error (now denoted by a red NOT gate ⊕, i.e., a Toffoli gate without any control lines) affects x_3. Then, the top line of the circuit is unconditionally affected by this error and, as such, this error may be detected by* all *of the $2^4 = 16$ possible input patterns for $\vec{x}$ (and not only in 2 out of 16 cases). Moreover, a single bit-flip error actually may occur in any part of the reversible circuit and will always be detected by all possible input patterns. That is, design errors in this circuit are obviously much easier to detect.*

This simple observation illustrates the (raw) potential reversible computations/reversible circuits might offer to address the challenges of verification. In the past, it has also been observed that reversible circuits offer some promising characteristics for testing (such as 100% controlability and observability; cf. [22]) and in the domain of quantum computing (which heavily rests on principles of reversible computation). In fact, it has recently been shown that, in quantum circuits, even small errors frequently affect the *entire* functionality and, hence, often can be detected with very few input patterns only (see [25]). Moreover, alternative verification approaches have recently been developed that exploit the reversibility of quantum computation and, by this, are capable of verifying the results of entire compilation flows [24]. However, thus far, all this latent potential has not yet evolved into a true alternative for

[1] Obviously, the considered example is rather artificial, but we believe it is sufficient to make the point.

the verification of conventional circuits and systems that exploits reversible computing for verification and, by this, addresses the challenges discussed in Section 1 in a complementary fashion. This leaves the questions why this is the case and whether reversible computing indeed may provide an alternative or not.

3 HOW TO LEVERAGE THIS POTENTIAL?

In this section, we shed light on the question raised above. For the first time, we conduct a thorough investigation of the true power of reversible computing as a complementary approach for verification of circuits and systems. To this end, we explore possible paths to exploit this potential and illustrate them by examples. Afterwards, we critically discuss them and eventually show why, after all, this hardly helps in order to address the verification challenge. By this, we are debunking misleading paths and provide the basis for using reversible computing as a complementary approach for verification in the future.

3.1 Using Reversible Gates

Probably the most straight-forward approach of using reversible computing is to *not* use classical gate libraries anymore, but to restrict the design flow to corresponding reversible gate libraries. An obvious choice would be Toffoli gates, since they constitute a universal gate library which can realize arbitrary functions [28]. This, of course, would require corresponding mappings from established gate libraries which, however, does not pose a significant obstacle. In the following, we are illustrating this by considering a design flow relying on *And-Inverter-Graphs* (AIGs, [29]) as a representative (in a similar fashion, this can be done for other design flows as well).

An AIG represents a classical circuit/Boolean function as a directed, acyclic graph. Starting from the root notes which represent primary inputs, each of the graph's following nodes represents an And operation that connects to other nodes by possibly complemented (i.e., inverted) edges. Eventually, the graph's terminal nodes represent the primary outputs of the circuit. Usually, this can be realized as a circuit by simply traversing the AIG from top to bottom and replacing each node (complement edge) with an And gate (Not gate) or corresponding Nand gate. This process can be kept almost identical using reversible gates: Rather than And and Nand gates, simply Toffoli gates can be used, where the respective inputs (incoming edges) are realized by control lines and the output is realized by an additional circuit line with constant input 0 (in case of And) or constant input 1 (in case of Nand) which serves as target line.

Example 5. *The circuit shown in Fig. 1 (without the error) already resembles an AIG representation of the Boolean function* $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$. *It consists of four inputs, one output, and three nodes. A one-to-one translation of this AIG to a Toffoli gate structure would result in the reversible circuit shown in Fig. 3. If* $\vec{x} = 1111_2$, *all control values of the Toffoli gates evaluate to* 1 *and, hence, the value of the fourth line (representing* f*) is inverted—resulting in the desired functionality. If any input is* 0, *the output line remains unchanged.*

As a hypothesis, we now assert that this already allows to exploit the potential sketched in Section 2.3. More specifically, it asserts that similar errors affect AIG representations of classical circuits

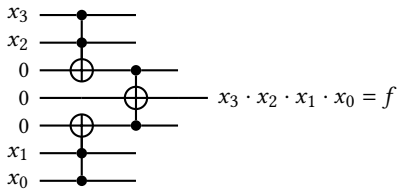

Fig. 3: Reversible circuit translation of the AIG resembled by the circuit shown in Fig. 1 using three additional circuit lines initialized with 0 and three Toffoli gates with two controls each. The target functionality f is realized on the fourth circuit line.

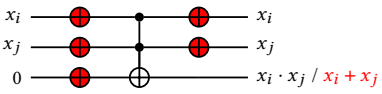

Fig. 4: The difference between a reversible And operation (circuit without ⊕) and a reversible Or operation (circuit including ⊕) is as hard to detect as with classical gates.

in different ways than they affect reversible circuits. In order to probe this hypothesis, the following example explores the effect of a common type of error on classical as well as reversible descriptions.

Example 6. *Consider a classical circuit described as an AIG and a corresponding reversible circuit (see, e.g., Fig. 1 and Fig. 3). Assume that an error conceptionally changes one And to an Or. Due to DeMorgan's Law (i.e.,* $x + y = \overline{\bar{x} \cdot \bar{y}}$*), this manifests in the AIG as a change in the polarity of all incoming and outgoing edges of the affected node. Such an error in general affects* 50 % *of all output patterns of the respective gate. In the reversible case, an And differs from an Or by the red Not gates shown in Fig. 4. Since the top two lines are returned to their original value in both cases, the error only manifests itself on the third line and, more importantly again in* 50 % *of all output patterns of the respective gate. Thus, there is no difference between the classical and the reversible And/Or per se.*

In general, proper reversible gates give rise to the same observable characteristics as their classical counterparts. After all, they are just another means to represent a Boolean function. Consequently, it can be concluded that potential benefits of the reversible computing paradigm for verification do not come from simply using reversible gates. Most likely, the function to be realized itself must assume more "reversible characteristics".

3.2 Just Make the Function Reversible

Most of the functions of practical interest are not reversible (they often differ in their number of inputs/outputs and do not realize one-to-one mappings). But luckily, an originally desired target function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ can be made reversible through a process called *embedding* [30], [31]. Here, so-called garbage outputs are added that are used to distinguish equal output patterns. Afterwards, constant inputs are added to ensure $n = m$ and, thus, make the function reversible. An examples illustrates this process:

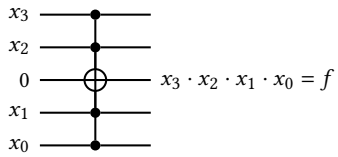

Fig. 5: Reversible embedding of $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$ using a single Toffoli gate with four controls—inverting the third line if all inputs are assigned 1. Thus, if this line is initialized with a constant 0, it realizes f. By leaving the input lines untouched, the circuit allows to discern all patterns that f maps to 0 and, thus, realizes f in a reversible fashion.

Example 7. *Consider again the function $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$. Originally, this function evaluates to 0 in fifteen cases and to 1 in a single case, i.e., a total of $\lceil \log_2(\max(15,1)) \rceil = 4$ additional (garbage) outputs are required to uniquely distinguish these output patterns. The values/functions of these garbage outputs are not relevant (since we are eventually only interested in f, those outputs are usually treated as don't cares anyway); but they, of course, have to guarantee a unique input/output-mapping. Since the original output f plus the four added garbage outputs yield a total of five outputs and to ensure $n = m$, one more (constant) input is added (to the existing four inputs)—eventually yielding a reversible function with five inputs and five outputs in which the originally intended function is embedded into. A reversible circuit realizing this function is shown in Fig. 5. As can be seen, this circuit realizes the target function f on its third line (as long as the constant input is assigned 0).*

As a hypothesis, we now assert that this allows to exploit the potential sketched in Section 2.3 in a better fashion than by only changing the gate library to a reversible one. Recall that we argued that, in reversible circuits as shown in Fig. 2 and discussed in Example 3, errors can easily be detected since they would unconditionally affect at least one of the outputs. However, as already stated in the example above, the newly added (garbage) outputs are often considered "don't cares" and, hence, their precise output function is usually not known (even embedding methods treat those garbage outputs as don't cares in order to allow for a scalable embedding [32]). Because of this, if an error only propagates to these garbage outputs, it cannot really be observed or used by a verification approach. In fact, following this procedure eventually only gives you yet another description of the originally given target function. Any verification method (independent of whether it would rely on simulation, formal methods, or other techniques) could only reason over the primary inputs and the primary outputs of the original function—not really providing any further benefit to the state of the art. Hence, just making the function reversible certainly does not provide an alternative to classical verification as well.

3.3 Using the Garbage Outputs

One obvious way to mitigate the problem observed above is, of course, to make better use of the garbage outputs, i.e., to make sure that any error which manifests on either output (primary *or* garbage) can be used for reasoning during verification. This would require all garbage outputs to be explicitly defined and known to the designer/the verification approach. An established way of doing that is by employing so-called uncomputation schemes[2] [34]. Here, the actually desired computations are realized first (using additional lines if required in order to guarantee reversibility; yielding a Compute-netlist). Once the respective (intermediate) results of those computations are not required anymore, they are uncomputed (which, in reversible computing, can easily be done, e.g., by symmetrically applying the Compute-netlist in reverse order; yielding an Uncompute-netlist). By this, no "garbage" outputs remain since all outputs are either primary outputs realizing the target functionality or reset to an initially known value (either the value of a primary input or a constant value). Again, an example illustrate the idea.

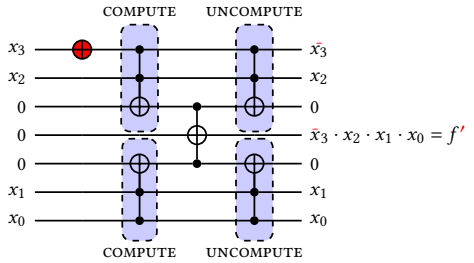

Fig. 6: Reversible realization of $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$ uncomputing intermediate results by applying the Compute-netlist in reverse order—fixing the output pattern to $(x_3, x_2, 0, f, 0, x_1, x_0)$. A bit-flip error (denoted by ⊕) on x_3 still unconditionally affects the whole output.

Example 8. *Fig. 6 shows the resulting circuit when applying the principle of uncomputation in order to realize $f(\vec{x}) = x_3 \cdot x_2 \cdot x_1 \cdot x_0$ (again, the red components shall be ignored for the moment). As can be seen, first the desired computations are conducted (using single Toffoli gates together with an additional constant input to realize the pairwise And operations). Once the desired result is realized (in this case f), the same gates are applied again to uncompute them—eventually resulting in the values of the primary inputs or the constant inputs. By this, a reversible circuit results in which* all *outputs (both, primary outputs and garbage outputs) are clearly defined and can be used for reasoning in verification.*

Now, the hypothesis is that, following this design scheme (which, as said above is quite common in the realization of quantum circuits or of reversible hardware description languages), allows to exploit the potential sketched in Section 2.3. And indeed, single errors would frequently manifest on the garbage outputs of a reversible circuit. Since the functionality of *all* outputs is known now, this increased manifestation of errors in the output can be exploited for increasing the probability of detecting such errors.

However, it remains debatable whether such scenarios would be realistic in practice. After all, there is no apparent reason why the Compute-netlist and the Uncompute-netlist should be designed independently from each other, since the Uncompute-netlist can

[2] Uncomputation is especially prominent in quantum computing (in order to avoid unwanted entanglement) or in the design of circuits from reversible hardware description languages [33].

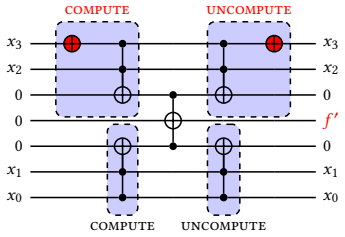

Fig. 7: A design process error equally affecting the COMPUTE as well as the UNCOMPUTE part of the original circuit shown in Fig. 6 (denoted by ⊕). This reduces the problem back to the complexity of the classical case—where only 2 out of 16 possible simulations reveal the discrepancy.

be constructed from the COMPUTE-netlist by just reversing its order. Then, however, effects of an error in the COMPUTE-netlist which propagate towards the garbage outputs would be canceled by its corresponding uncomputation (using the same netlist just in reverse order). Eventually, only errors would remain which propagate through the primary outputs.

EXAMPLE 9. *Consider again the circuit from Fig. 6 and a bit-flip error affecting the topmost line. Then, as discussed in Section 2.3, this would unconditionally affect at least one of the outputs (the topmost in this case) and can be detected by any of the 16 possible input assignments. However, if such an error is, e.g., the result of a design process which realizes this particular AND operation in a wrong fashion, it would not yield a circuit as shown in Fig. 6, but instead a circuit as shown in Fig. 7. Here, the UNCOMPUTE-netlist is not designed from scratch again, but simply the COMPUTE-netlist is applied in reverse order. By that, the error only propagates towards the primary output f, while it is canceled towards the topmost garbage output. After all, the error only manifests in 2 out of 16 cases; no improvement compared to the classical circuit.*

Because of this, also using all garbage outputs by merely employing a straight-forward symmetric uncomputation scheme does not provide an alternative to classical verification.

3.4 Using Pure Reversible Computing Makes the Difference

Apparently simple and/or established ways of realizing the originally desired target function in a reversible fashion seem to mitigate the potential for verification as sketched in Section 2.3. This may also explain why using the reversible computing paradigm for verification has not become mainstream yet; although characteristics such as discussed in Section 2.3 certainly look intriguing. By clearly debunking them, we now see what really makes the difference and what may open a path towards approaches that may be able to use reversible computing for verification. To that end, let's re-visit the example from the very beginning of this paper again:

EXAMPLE 10. *Consider again the reversible circuit shown in Fig. 2. As discussed in Example 4, a single bit-flip error such as the one denoted by the red NOT gate can be detected by* any *of the possible input patterns, i.e., even a randomly generated stimulus will show the error by a simple simulation. With the discussions from above, we can now conclude that this is the case, because*

- *the error's effect is always fully propagated towards the outputs of the circuit due to the reversibility of all gates after the error (in contrast to conventional circuits, where certain values at one input of, e.g., an AND gate, can "block" the value of the other input and all possible errors coming from it it)*[3].
- *the outputs of the circuit are all clearly defined—leaving no "garbage" outputs in an unknown don't care state and, thus, the complete output can be used for detection and reasoning, as well as*
- *the circuit has been realized in an asymmetric fashion eliminating the re-use of (possibly erroneous) building blocks in reverse order, i.e., the additional effects of an error on certain lines are not canceled later on in the circuit.*

These observations can be generalized for generic errors and pure reversible circuits (i.e., reversible circuits realizing one-to-one mappings on all lines): In fact, the probability of triggering a single error in a pure reversible circuit and observing its effect at the circuit's output is independent of (1) the position where the error manifests itself as well as (2) the circuit itself. This is the case, because of the following: Any reversible circuit containing an error can always be split into a sequence of gates prior as well as after the error. Since the gates prior to the error also realize a one-to-one mapping of the circuit's input values (after all they just form another reversible circuit), the probability of triggering the error (i.e., causing an effect on some of the circuit lines) is not affected by these prior gates. At the same time, if there are no "garbage" outputs (with don't care values) and if no cancellation effects occur by re-using building blocks in reversed order, any gate after the error cannot reduce its effect on the output (due to the reversible gates, again, realizing a bijective mapping). Eventually, in this setting, everything that happens prior as well as after the error has no influence on the "hardness" of detecting it. While this cannot be guaranteed that stringently for multiple errors anymore (indeed, in case of multiple errors, masking effects may occur even in reversible computations; as witnessed in Example 9), evaluations (summarized in the next section) show that, in the vast majority of cases, huge benefits in detection probability can be achieved.

Overall, this shows that, if circuits are designed and/or realized in a pure reversible fashion, verification gets substantially easier—providing a path to address some of today's verification challenges. To this end, we would need either

(1) dedicated embedding techniques that allow to realize an (irreversible) Boolean target function in a "purely reversible" fashion (i.e., techniques which avoid the use of don't cares for outputs and the use of methods leading to cancellation effects as discussed above) and/or
(2) a completely new design and synthesis flow which considers reversible computing right from the beginning of the circuit design (e.g., starting with reversible specifications of the target functionality) and, by this, avoids simple and/or established ways, e.g., for embedding thus far, which mitigate the potential for verification.

[3] Those masking effects are the reason why, in the conventional circuit from Example 2, the (basically same) error is only detected in 2 out of 16 cases.

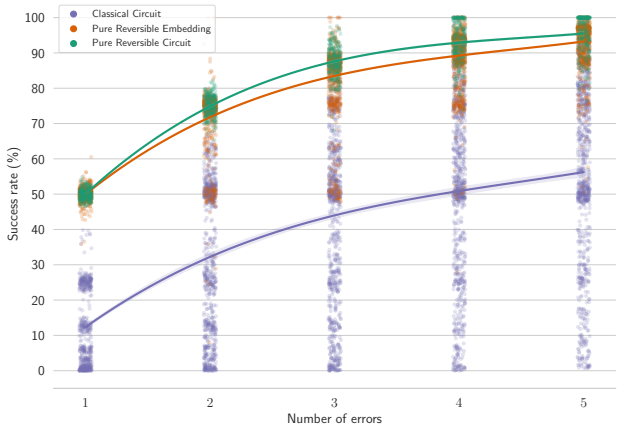

Fig. 8: *Potential of reversible computing*: number of errors (x-axis) vs. success rate of detecting these errors (y-axis). Different colors denote the distinct scenarios investigated in Section 3. Each dot represents the result of a single instance, whereas the respective lines represent the corresponding average value. Classical Circuits have the worst success rate; in many cases close to 0 %. Pure Reversible Embedding and Pure Reversible Circuits show astonishing success rates; in the "worst" case at ≈ 45 % (i.e., almost every second input is going to detect the error).

We understand that those are bold claims and a realization of them certainly is out of scope for a single research paper. But the evaluations summarized in the next section show the potential such a comprehensive and disruptive endeavor could have for improving the performance of verification. Motivated by that, we also discuss afterwards in Section 5 why following such a path, although bold and ambitious, is not completely impossible.

4 EMPIRICAL EVALUATION

All discussions above have been conducted in a conceptual fashion only and, for the sake of illustration, have been demonstrated by rather simple examples thus far. To additionally strengthen our investigations, we also aimed to confirm our findings by extensive empirical evaluations. To this end, we considered the possible paths of exploiting reversible computing (as discussed above) and evaluated their possible impact on the required verification effort. More precisely, we considered a set of circuits from (classical) benchmark libraries such as LGSynth91 or ESPRESSO and compared their verification effort when treating them as the original (classical) circuit (denoted *Classical Circuit* in the following) compared to a pure reversible embedding generated by RevKit [35] (which satisfies all the requirements discussed in Section 3.4 and is denoted *Pure Reversible Embedding* in the following). To additionally evaluate the verification effort of reversible circuits resulting from a design flow which considers reversibility from the beginning, we additionally generated some random reversible functions (the resulting circuits are denoted *Pure Reversible Circuits* in the following).[4]

[4]Since no such design flow exists yet, we needed to resort to random functions. However, we strongly believe the results on the verification effort will be the same for more specific functions as well.

Afterwards, for all three types of circuits, we randomly injected errors in the corresponding realizations and measured how hard it is to detect these errors. Specifically, for each type of circuit, we

- created 64 differently seeded random instantiations of 1-5 errors which we injected into each of the circuits,
- simulated all circuits with 1024 randomly chosen inputs, and
- recorded the success rate, i.e., recorded how many of those 1024 inputs allowed to detect this error.

This resulted in more than 30 million conducted simulations which allows us to derive thorough conclusions about the required verification effort of all three types of circuits (classical circuits, purely reversible embedding circuits, and pure reversible circuits).

Fig. 8 summarizes the obtained results. Here, the x-axis denotes the number of errors which have been injected while the y-axis denotes the success rate for each circuit/instance, i.e., how many of the 1024 randomly chosen inputs for each circuit/instance actually detected the error(s). Each dot represents the result of a single case, whereas the respective lines represent the corresponding average value.

The results clearly confirm the discussions and observations made in this work: Not surprisingly, classical circuits have the worst success rate, i.e., many of the injected errors cannot be detected at all (success rate of 0%) or only with a fraction of the applied inputs[5]. In contrast, the purely reversible embedding circuits and the pure reversible circuits have very good success rates. In particular, the results for pure reversible circuits are astonishing: Here, the "worst" success rates are at approx. 45%, i.e., almost every second (randomly generated) input is going to detect the error. This really shows the potential of reversible circuits for verification as it suggests that already purely simulation-based verification methods could yield high coverage. The results for the purely reversible embedding circuits are slightly worse than that, but still almost all errors can be detected within few simulations.

Overall, this also empirically confirms the potential of reversible computing: If all requirements discussed in Section 3.4 are satisfied, circuits result for which errors can be detected within a couple of simulations (while, for classical circuits, still a substantial amount is required or the error can hardly be detected at all).

5 RESULTING INSIGHTS, CONSEQUENCES, AND CONCLUSIONS

The investigations and evaluations from above confirm that, if a circuit is realized in a purely reversible fashion, design errors indeed can be detected easier (often within few simulation runs). This certainly would be an improvement compared to the state of the art, where, e.g., formal verification and model checking are reaching their limits and errors frequently escape when using simulation-based verification. At the same time, it is also shown that this benefit quickly dissolves if the desired target function is not reversible or only "made reversible" in a naive fashion (e.g., by simply using reversible gates, by an embedding which treats

[5]Many instances also have a rather high success rate, but verification is hard because of the "hard to detect"-corner cases, not because of the easy to detect errors.

garbage outputs as don't care, or by using straight-forward uncomputation schemes). All these insights eventually raise the question what consequences and conclusions can be derived from that?

Our assessment is that these investigations provide the basis for an alternative path towards closing the verification gap, namely by using the reversible computing paradigm rather than solely relying on evolutionary improvements in the verification of classical circuits. We are aware that this would be a bold (and most likely controversy) endeavor: Thus far, most of the established design flows completely rely on the classical computing paradigm and provide no or only limited support for reversible circuits. Exploiting reversible computing for verification would require an enormous paradigm shift. But developments in other areas show that such paradigms shifts are not impossible. For example:

- Quantum computing [16] also works completely different than classical computing, but offers the prospect of solving hard problems substantially faster than classical circuits. While first ideas in this direction have been proposed by Shor [36] and Grover [37] in the last millennium, meanwhile first practical relevant applications are emerging—triggering an entire community (including "big players" such as IBM, Google, etc.) to develop corresponding design flows and realizing the paradigm shift from classical to quantum.
- Recent developments in low-power design show that improvements in this domain are also reaching limits. Accordingly, researchers investigated alternatives as well—with very promising accomplishments made, e.g., in the development of fully-adiabatic and reversible circuits for low-power design [20], [21]. This also triggered the proposal of a paradigm shift as, e.g., outlined in the *IEEE Spectrum* article "The Future of Computing Depends on Making It Reversible" [38].

We see no reasons, why similar developments towards a design flow for circuits and systems (at least for safety-critical applications) that are easier to verify should not be possible as well. With this work, we hope to have laid out the basis for such an endeavor.

ACKNOWLEDGMENTS

This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318), was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus, and has been supported by the BMWK on the basis of a decision by the German Bundestag through project QuaST, as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

REFERENCES

[1] J. Yuan, C. Pixley, and A. Aziz, *Constraint-based verification*. Springer, 2006.
[2] R. Wille *et al.*, "SMT-based stimuli generation in the SystemC Verification library," in *Forum on Specification and Design Languages*, 2009.
[3] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrained random simulation," in *Int'l Conf. on CAD*, 2007.
[4] J. Bergeron, *Writing Testbenches using System Verilog*. Springer, 2006.
[5] T. Zhang, D. Saab, and J. A. Abraham, "Automatic assertion generation for simulation, formal verification and emulation," in *IEEE Annual Symp. on VLSI*, 2017.
[6] A. Koczor *et al.*, "Verification approach based on emulation technology," in *Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2016.
[7] Y. Kuwabara, T. Yokotani, and H. Mukai, "Hardware emulation of IoT devices and verification of application behavior," in *Asia-Pac. Conf. Commun.*, 2017.
[8] S. Disch and C. Scholl, "Combinational equivalence checking using incremental SAT solving, output ordering, and resets," in *Asia and South Pacific Design Automation Conf.*, 2007.
[9] J. Marques-Silva and T. Glass, "Combinational equivalence checking using satisfiability and recursive learning," in *Design, Automation and Test in Europe*, 1999.
[10] P. Molitor and J. Mohnke, *Equivalence checking of digital circuits: Fundamentals, principles, methods*. Springer, 2010.
[11] A. Biere and W. Kunz, "SAT and ATPG: Boolean engines for formal hardware verification," in *Int'l Conf. on CAD*, 2002.
[12] H. M. Le *et al.*, "Detection of hardware Trojans in SystemC HLS designs via coverage-guided fuzzing," in *Design, Automation and Test in Europe*, 2019.
[13] K. Laeufer *et al.*, "RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs," in *Int'l Conf. on CAD*, 2018.
[14] A. Biere *et al.*, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, 1999.
[15] E. M. Clarke *et al.*, *Model Checking*. MIT Press, 2018.
[16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
[17] A. Bérut *et al.*, "Experimental verification of Landauer's principle linking information and thermodynamics," *Nature*, 2012.
[18] A. Zulehner and R. Wille, "Taking one-to-one mappings for granted: Advanced logic design of encoder circuits," in *Design, Automation and Test in Europe*, 2017.
[19] R. Wille *et al.*, "Synthesis of approximate coders for on-chip interconnects using reversible logic," in *Design, Automation and Test in Europe*, 2016.
[20] M. P. Frank *et al.*, *Reversible computing with fast, fully static, fully adiabatic CMOS*, 2020. arXiv: 2009.00448.
[21] A. Zulehner, M. P. Frank, and R. Wille, "Design automation for adiabatic circuits," in *Asia and South Pacific Design Automation Conf.*, ACM, 2019.
[22] I. Polian *et al.*, "A family of logical fault models for reversible circuits," in *Asian Test Symp.*, 2005.
[23] R. Wille, H. Zhang, and R. Drechsler, "ATPG for reversible circuits using simulation, boolean satisfiability, and pseudo boolean optimization," in *IEEE Annual Symp. on VLSI*, 2011.
[24] L. Burgholzer, R. Raymond, and R. Wille, "Verifying results of the IBM Qiskit quantum circuit compilation flow," in *Intl Conf Quantum Comput. Eng.*, 2020.
[25] L. Burgholzer, R. Kueng, and R. Wille, "Random stimuli generation for the verification of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2021.
[26] L. Amarú *et al.*, "Exploiting inherent characteristics of reversible circuits for faster combinational equivalence checking," in *2016 Des. Autom. Test Eur. Conf. Exhib. DATE*, 2016.
[27] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. Springer, 2010.
[28] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, Springer, 1980.
[29] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*, Springer, 2010.
[30] D. Maslov and G. W. Dueck, "Reversible cascades with minimal garbage," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2004.
[31] A. Zulehner and R. Wille, "Make it reversible: Efficient embedding of non-reversible functions," in *Design, Automation and Test in Europe*, 2017.
[32] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2018.
[33] R. Wille *et al.*, "SyReC: A hardware description language for the specification and synthesis of reversible circuits," *Integration*, 2016.
[34] C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev.*, 1973.
[35] M. Soeken *et al.*, "Revkit: An open source toolkit for the design of reversible circuits," in *Int'l Conf. of Reversible Computation*, 2011.
[36] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, 1997.
[37] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, 1996.
[38] M. P. Frank, "The Future of Computing Depends on Making It Reversible," *IEEE Spectrum*, 2017. [Online]. Available: https://spectrum.ieee.org/computing/hardware/the-future-of-computing-depends-on-making-it-reversible.