Embedding Modulo Counter Circuits for their Polynomial Formal Verification

Caroline Dominik[1], Rolf Drechsler [1]
[1]University of Bremen, Bremen, Germany, {cardom, drechsler}@uni-bremen.de

Abstract

Proving the correct behavior of hardware by verifying it is a central aspect of maximizing the quality of digital systems. But this task has a high computational complexity and the required resources of a specific application are usually unpredictable. This makes the integration into the workflow of modern circuit design challenging. The recent approach of *Polynomial Formal Verification* (PFV) aims to identify verification methods and classes of circuits for which an efficient verification can always be guaranteed - so far with a focus on combinational circuits. A first application to sequential circuits considered full counter circuits and proposed a method applicable to bijective functions. An efficient verification could be ensured despite their exponential sequential depth.
Within this paper, this verification approach is extended to be applicable to any modulo counter circuit. For this, bijectivity of the circuits function is generated by utilizing the concept of embedding. It is theoretically proven that the application can be carried out in polynomial time and space. Experimental evaluations confirm these theoretical findings.

1 Introduction

As digital systems have become a substantial part of our society, we rely heavily on their correct behavior. With formal verification, it can be mathematically proven that any hardware design acts according to its specification, but this can be arbitrarily complex. In industrial hardware design though, the required time and memory usage are central parameters as well.
While symbolic representations like *Binary Decision Diagrams* (BDDs) can reduce memory usage, they are not a universal solution. A counterexample is the integer multiplication, for which the size of a BDD always grows exponentially [1]. In addition to that, to determine the efficiency of a verification procedure, **analyzing the size of only the final result is not sufficient**. When examining the equivalence of two circuits using a miter circuit [2], the final result will be "0", if they are equivalent. The BDD for this consists of only one terminal node, but some **intermediate results** can be of exponential size, e.g. when comparing two multiplier circuits. In general, determining the satisfiability of a miter circuits is coNP-complete. Any intermediate blowups have to be prevented, so that formal verification can be applied on a larger scale. The goal of *Polynomial Formal Verification* (PFV) [3], [4] is to ensure that the time and memory resources used throughout the entire verification stay polynomial with respect to the number of inputs. This can be achieved by selecting or modifying the verification approach per class of circuits. E.g. PFV could be proven for multipliers based on *Symbolic Computer Algebra* (SCA) [5]. There are several more successful applications to combinational circuits, such as circuits with a limited cutwidth [6], [7] or a simple RISC-V processor [8].

This work was supported by DFG within the Reinhart Koselleck Project *PolyVer* (DR287/36-1) and the research training group *CAUSE*.

The verification problem becomes even more complex for sequential circuits, which further include state elements. Whereas the mentioned miter circuit, when used for the equivalence of two combinational circuits, has to evaluate to "0" for all possible input patterns, the same has to hold for all sequences of inputs for sequential circuits. Because of this, PFV has barely been applied to sequential cases yet. A major challenge here is the number of consecutive steps needed until all possible states have been reached, which is called the sequential depth. E.g. a binary counter with n bits, that starts with the state "0" and counts up to the state $2^n - 1$ is technically a very simple circuit. But since the sequential depth grows exponentially with respect to n, so does the runtime when established verification methods are applied. **Despite of this exponential sequential depth, PFV was proven** for full counter circuits in [9]. For this, a method was proposed, which uses their bijectivity.
This paper now aims to extend this method to be applicable to modulo counter circuits, which count up to a modulo value m with $0 < m \leq 2^n$. This is not directly possible since bijectivity is not guaranteed anymore, due to the reset to "0" for several states. To overcome this, a concept used for reversible computation is utilized: Embedding a non-reversible function alters it with the goal of achieving reversibility, without changing its general behavior. Hence, by extending the method of [9] by embedding, an efficient verification procedure for modulo counter circuits is proposed in this paper. This will be theoretically proven by polynomial upper bounds for the runtime and memory usage and further experimentally evaluated. Note, that the corresponding proofs are given in [10] and will only be sketched in this paper due to page limitations.
This paper is structured as follows. In Section 2, the necessary concepts are explained. Then, in Section 3 the verification approach of [9] is modified for its application to modulo counters and polynomial bounds for the

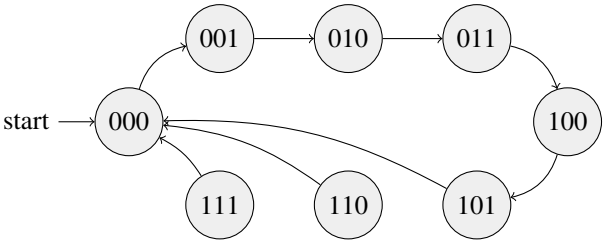

Figure 1 FSM of a 3-bit modulo-6 counter.

resources required are derived. These theoretical results are underlined with experimental results in Section 4 and, finally, the paper is concluded in Section 5.

2 Background

In the following, the used concepts are introduced. For this, let a Boolean function $f : B^n \rightarrow B^m$ be given by $f(x_0, x_1, \ldots, x_{n-1})$ with $B := \{0,1\}$. A general understanding of circuits, BDDs [1] and model checking (see Chapter 1 of [11]) is assumed.

2.1 Counter Circuits

In this paper, counter circuits are considered, which are specified below. A *full n-bit counter* (FC_n) has n flip-flops $ff_0, ff_1, \ldots, ff_{n-1}$. Initially, these flip-flops are all set to "0". With each clock signal, they count up in binary. After taking the final value "$2^n - 1$", all flip-flops are reset to "0". An *n-bit modulo-m counter* "M_mC_n" counts up in a similar manner, but the final value before the reset is given by $m-1$ with $0 < m \leq 2^n$. For this paper all remaining values are defined to reset to "0" as well.
To formally model the behavior of a counter with a *Finite State Machine* (FSM), each flip-flop ff_i can be represented with a state variable $s_i := ff_i$. The corresponding FSM for a 3-bit mod-6 counter can be seen in **Figure 1**.

2.2 PFV of Full Counter Circuits

Polynomial bounds for the resources used during the complete verification process of full counters have been proven in [9]. As mentioned above, the main challenge here is the exponential sequential depth of the circuits. The established method of *Symbolic Model Checking* (SMC) must require an exponential number of steps. For the given FSM, it starts with an empty set of reachable states Q_r and an initial state q_0. The set Q_r is then incrementally updated with all new states, which can be reached within one step. It is easy to see, that for an n-bit full counter this results in 2^n increments. Because of this, an alternative model checking variant was introduced in [9], which is defined in the following.

Definition 2.1. ***Restricted Domain Model Checking*** *(RDMC) verifies the influence of each variable. For this, $2 \cdot n$ initial sets I are created by restricting the set of state variables $S := \{s_0, s_1, \ldots, s_{n-1}\}$ by $s_i := 0$ or by $s_i := 1$ for each s_i. Then, for each set I, a set Q_r of states reachable within one step is created.*

Table 1 RDMC OF 3-BIT COUNTERS.

s_i	I	Q_r	
		FC_3	M_6C_3
$\bar{s}_2$	$\{0,1,2,3\}$	$\{1,2,3,4\}$	$\{1,2,3,4\}$
s_2	$\{\mathbf{4},5,6,7\}$	$\{\mathbf{5},6,7,0\}$	$\{\mathbf{5},\mathbf{0}\}$
$\bar{s}_1$	$\{0,1,\mathbf{4},5\}$	$\{1,2,\mathbf{5},6\}$	$\{1,2,\mathbf{5},\mathbf{0}\}$
s_1	$\{2,3,6,7\}$	$\{3,4,7,0\}$	$\{3,4,0\}$
$\bar{s}_0$	$\{0,2,\mathbf{4},6\}$	$\{1,3,\mathbf{5},7\}$	$\{1,3,\mathbf{5},\mathbf{0}\}$
s_0	$\{1,3,5,7\}$	$\{2,4,6,0\}$	$\{2,4,0\}$

The resulting 2·n sets Q_r of RDMC define the behavior of a circuit based on the intersection of sets. Whereas each state q_i can be constructed by the intersection of n initial sets I, the successor state q_i' of q_i can be constructed by the intersection of the n corresponding sets Q_r. To verify a circuit, the sets Q_r are compared to those of a golden model or the specification.

Example 2.1. *The computed sets when applying RDMC to a 3-bit full counter can be seen in **Table 1**. The $2 \cdot n = 6$ initial sets can be seen in the second column, the corresponding restricted state variables in the first column, and the corresponding sets Q_r in the third column. The state $4 = 0b100$ can be obtained by the intersection $\{4,5,6,7\} \cap \{0,1,4,5\} \cap \{0,2,4,6\}$, as marked in bold. Analogously, the intersection $\{5,6,7,0\} \cap \{1,2,5,6\} \cap \{1,3,5,7\}$ results in the successor state "5".*

2.3 Embedding

The computations of a *reversible function* $f : B^n \rightarrow B^n$ can always be inverted. This is the case if f has an equal number of inputs and outputs n and if f is bijective. These two conditions can be obtained for an irreversible function $g : B^n \rightarrow B^m$, e.g. by adding outputs called *garbage outputs* until g is bijective and by adding inputs called *constant inputs* until the number of inputs and outputs is equal. This process is called *embedding* (see Chapter 2 and 3 of [12]).

3 PFV of Modulo Counter Circuits

An immediate application of RDMC to a modulo counter, to obtain polynomial runtime and memory usage as described in Section 2.2 for full counters, is not possible. This is because the final results of RDMC are not always distinctive for non-bijective functions.

Example 3.1. *Analogously to Example 2.1, the sets of verifying a 3-bit mod-6 counter can be seen in Table 1 as well. Again, the first column shows the restricted state variables and the second column the corresponding initial sets, but sets Q_r can be seen in the fourth column now. Here, the intersection of the sets Q_r for the state "4" is $\{5,0\} \cap \{1,2,5,0\} \cap \{1,3,5,0\}$, which results in $\{5,0\}$.*

To nonetheless apply RDMC to modulo counters, the concept of embedding, as introduced in Section 2.3, can

Table 2 TRUTH TABLE FOR HALF EMBEDDED 3-BIT MOD-6 COUNTER.

s_2	s_1	s_0	s'_2	s'_1	s'_0	g_1	g_0
0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1
0	1	0	0	1	1	1	0
0	1	1	1	0	0	1	1
1	0	0	1	0	1	0	0
1	0	1	**0**	**0**	**0**	**0**	**1**
1	1	0	**0**	**0**	**0**	**1**	**0**
1	1	1	**0**	**0**	**0**	**1**	**1**

be utilized. Since constant inputs are not necessary in this case, the definition is altered.

Definition 3.1. *A function* $\boldsymbol{he}(f) : B^n \to B^{m+k}$ *is the* ***half embedding*** *of a non-bijective function* $f : B^n \to B^m$*, if* $he(f)$ *results out of adding* k *garbage outputs to* f *so that it maps all* n *inputs to unique outputs. Compared to an embedding, this creates bijectivity on the relevant subset of* B^{m+k} *instead of reversibility.*

The half embedding of a function f can be computed based on RDMC. For this, RDMC is applied and the resulting sets Q_r of each state variable s_i are compared. If the intersection of the two sets Q_r corresponding to the two initial sets I obtained by restricting the same variable s_i to "0" or to "1" is not empty, a garbage output g_i with $f(g_i) := s_i$ is added. The proof for correctness of RDMC-based half embedding in [10] is sketched in the following: As stated in Definition 2.1, the two intersected sets Q_r represent the influence of the state variable s_i. This means, only if this intersection is empty, then s_i clearly separates the co-domain of f with its assignment and no additional information is necessary.

Example 3.2. *For the RDMC-based half embedding of the 3-bit mod-6 counter of Example 3.1,* g_0 *is added, because* $\{1,3,5,0\} \cap \{2,4,0\} = \{0\}$ *and* g_1 *is added, because* $\{1,2,5,0\} \cap \{3,4,0\} = \{0\}$*. The resulting truth table can be seen in* ***Table 2****. Due to the added garbage outputs, the three different predecessors of state "0" can be distinguished, as marked in bold in the truth table. That way, the intersection used to compute the successor of state "4" only contains state "5".*

The combination of RDMC-based half embedding and RDMC hence gives an approach for the verification of modulo counters, which is similar to the one used for full counters in Section 2.2.
For an overall polynomial verification procedure, further an efficient representation of the transition relation of the given FSM and of each computed set is necessary. Similar to SMC, BDDs are used as a symbolic representation. As an example, the BDD of a 3-bit mod-6 counter can be seen in **Figure 2**. Each pair consisting of a current state variable s_i and a successor state variable s'_i is ordered next to each other and all pairs are ordered in reversed order. During the half embedding, any added garbage output g_i is added after its corresponding pair.

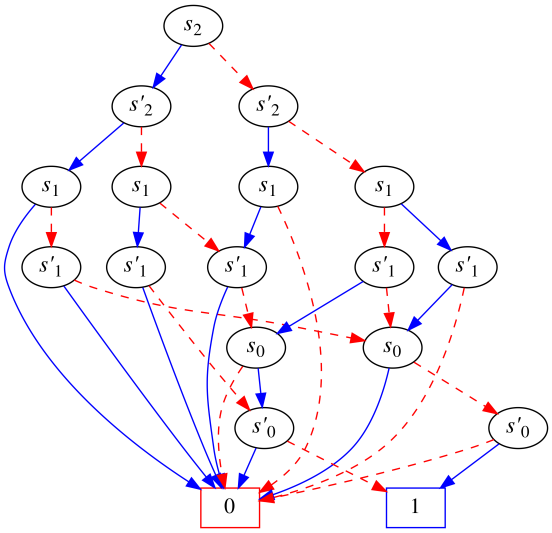

Figure 2 Transition relation of a 3-bit mod-6 counter.

Lemma 3.1. *Let BDD* T *describe the transition relation of an* n*-bit modulo-*m *counter and* $he(T)$ *the RDMC-half-embedding of* T*, both with variables ordered as reversed pairs* $(s_{n-1}, s'_{n-1}, s_n, s'_n, \ldots, s_0, s'_0)$*. Then* $|T| \leq 10 \cdot n$ *and* $|he(T)| \leq 2 \cdot (|T| + n)$ *hold.*

This is proven in [10] based on the pattern in the structure of BDD T, which is created by the specified variable order. In general, a pair-wise variable ordering can reduce the size of a BDD, because those variables strongly depend on each other and no information about these dependencies has to be stored for several variables.

Theorem 3.1. *The time and space demands of RDMC-based half embedding followed by RDMC of an* n*-bit modulo-*m *counter with variables ordered as reversed pairs are in* $\mathcal{O}(n^2)$ *and* $\mathcal{O}(n)$*, respectively.*

The proof in [10] analyzes the number of steps of RDMC and RDMC-based half embedding according to Definition 2.1 and Definition 3.1. The bottleneck here is the image computation. Further, the size of each BDD that is created is analyzed based on Lemma 3.1.

4 Experimental Results

The presented verification approach was implemented in C++, using CUDD [13] for BDD operations and a setup with a 3.6 GHz AMD Ryzen 5 CPU and 16 GB RAM. The size of all BDDs computed during verification was measured for a fixed number of variables n and all possible modulo values m with $1 < m \leq 2^n$. The results for $n = 8$ can be seen in **Figure 3**. The maximum BDD size with respect to the modulo values $1 < m \leq 256$ is marked with green triangles for RDMC-half-embedding and with blue crosses for RDMC. The size of the transition relation T is given as turquoise squares, the size of $he(T)$ as orange circles. It can be seen that these measures sizes never exceed the upper bound $2 \cdot (|T| + n)$ given by Lemma 3.1, which is marked in gray and dashed. Similar results were obtained for up to $n = 14$.

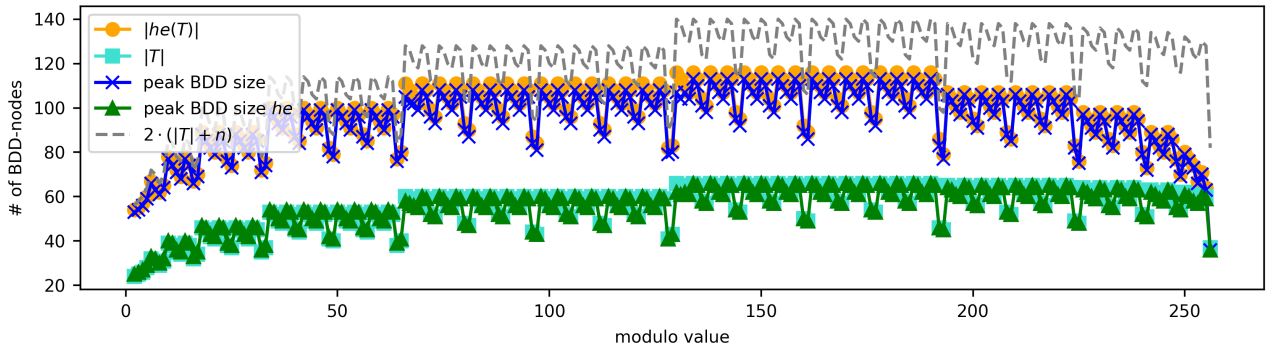

Figure 3 BDD sizes for different m during RDMC of 8-bit modulo-m counter.

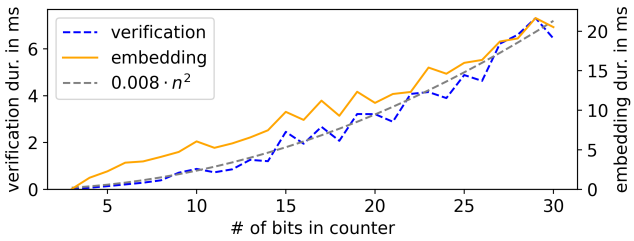

Figure 4 Duration for different n and $m = 2^{n-1} + 2$ during RDMC of n-bit modulo-m counter.

To analyze the increase in runtime for $0 < n \leq 30$, m was fixed to $2^{n-1} + 2$. **Figure 4** shows the measurements in milliseconds for RDMC-half-embedding in orange and for RDMC in blue and dashed. Both are similar to the reference function $a \cdot n^2$ with $a = 0.008$ in gray and dashed. This polynomial growth underlines the runtime predicted by Theorem 3.1. Similar results were obtained for other fixed modulo values.

5 Conclusion

To summarize, with the verification procedure proposed in this paper, consisting of RDMC-based half embedding and RDMC, we were capable of proving PFV for modulo counter circuits. The efficiency of the approach was not only theoretically proven, but further observed with experimental evaluations. With this, the strong limitation to bijective functions, given by RDMC, could be eliminated.

With research of proving PFV for sequential circuits only just beginning, of course many open challenges still remain. This e.g. includes analyzing the application of RDMC to other classes of circuits.

6 Literature

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," in *IEEE Transactions on Computers*, vol. 35, no. 8, 1986, pp. 677–691.

[2] D. Brand, "Verification of large synthesized designs," in *Int'l Conf. on CAD*, 1993, pp. 534–537.

[3] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *Int'l Symposium on Design and Diagnostics of Electronic Circuits & Systems*, 2021, pp. 99–104.

[4] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *Int'l Conf. on CAD*, 2022, pp. 1–9.

[5] R. Drechsler, A. Mahzoon, and M. Goli, "Towards polynomial formal verification of complex arithmetic circuits," in *Int'l Symposium on Design and Diagnostics of Electronic Circuits & Systems*, 2022, pp. 1–6.

[6] L. Müller and R. Drechsler, "SAT can ensure polynomial bounds for the verification of circuits with limited cutwidth," in *Euromicro Conference on Digital System Design*, 2024, pp. 57–64.

[7] M. Nadeem, C. K. Jha, and R. Drechsler, "Polynomial formal verification of approximate adders with constant cutwidth," in *IEEE European Test Symposium*, 2024, pp. 1–6.

[8] L. Weingarten, K. Datta, A. Kole, and R. Drechsler, "Complete and efficient verification for a RISC-V processor using formal verification," in *Design, Automation & Test in Europe Conference & Exhibition*, 2024, pp. 1–6.

[9] C. Dominik and R. Drechsler, "Polynomial formal verification of sequential circuits," in *Design, Automation & Test in Europe Conference & Exhibition*, 2024, pp. 1–6.

[10] C. Dominik, "Embedding sequential circuits for their polynomial formal verification," Master's thesis, University of Bremen, 2024.

[11] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Springer Cham, 2018.

[12] R. Wille and R. Drechsler, *Towards a Design Flow for Reversible Logic*. Springer Dordrecht, 2010.

[13] F. Somenzi, "CUDD: CU decision diagram package release 3.0.0," 2016. [Online]. Available: https://github.com/ivmai/cudd