SCALABLE FORMAL VERIFICATION OF FINITE FIELD ARITHMETIC CIRCUITS USING COMPUTER ALGEBRA TECHNIQUES

by

Jinpeng Lv

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

The University of Utah

August 2012

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Jinpeng Lv

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Priyank Kalla

Ganesh Goplakrishnan

Chris Myers

Kenneth Stevens

Rongrong Chen

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of Jinpeng Lv in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Priyank Kalla
Chair, Supervisory Committee

Approved for the Major Department

Gianluca Lazzi
Chair/Dean

Approved for the Graduate Council

Charles A. Wight
Dean of The Graduate School

To Ruina.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am grateful to many people, without whose support I could not have completed this Ph.D. study and dissertation. First and foremost, I would like to thank my advisor, Professor Priyank Kalla. He has been patient enough to teach me every knowledge I need to learn. I have learned many things from him, not only the knowledge itself, but also the way how to organize the knowledge and apply it to real world problems. Moreover, he is always available to discuss questions with me and provided perspectives based on his experience. I especially enjoy the brainstorming with him. Actually, the most important result of my Ph.D. research is achieved by brainstorming. Next, I would like to thank Professor Florian Enescu for his extensive help and contribution to this work. I would also like to thank the other members of my committee - Ganesh Gopalakrishnan, Chris Myers and Ken Stevens, Rongrong Chen for their help and support. Finally, I would like to thank my friends: Arun Jay, Sammer Merchant, Brandt Hammer for all the good times we spent together.

ABSTRACT

With the spread of internet and mobile devices, transferring information safely and securely has become more important than ever. Finite fields have widespread applications in such domains, such as in cryptography, error correction codes, signal processing, among many others. Therefore, dedicated hardware (VLSI) implementations of finite field arithmetic abound. In most finite field applications, the field size – and therefore the bit-width of the operands – can be very large. For example, the U.S. National Institute for Standards and Technology (NIST) recommends the use of finite fields corresponding to data-path sizes of 163-bits or more for elliptic curve cryptography. The high complexity of arithmetic operations over such large fields requires circuits to be (semi-) custom designed. This raises the potential for errors/bugs in the implementation, which can be maliciously exploited and can compromise the security of such systems. Formal verification of finite field arithmetic circuits has therefore become an imperative.

This dissertation targets the problem of *formal verification of hardware implementations of combinational arithmetic circuits over finite fields of the type* $\mathbb{F}_{2^k}$. Two specific problems are addressed: i) verifying the correctness of a custom-designed arithmetic circuit implementation against a given word-level polynomial specification over $\mathbb{F}_{2^k}$; and ii) gate-level equivalence checking of two different arithmetic circuit implementations.

In practical applications, the very large size and complexity of finite field arithmetic circuits makes design verification a very hard problem to solve. Contemporary automatic verification approaches, including those that rely on solver-based technology (such as SAT and SMT solvers), decision-diagram based methods (BDDs and BMDs), or those based on And-Invert-Graph based reductions (AIG/ABC), etc., are infeasible in proving the correctness of large-scale custom designed arithmetic circuits. In general, these techniques lack the requisite power of abstraction and the mathematical wherewithal to efficiently model and verify modulo-arithmetic circuits. In such cases, efficient symbolic reasoning is required that can model and analyze the underlying arithmetic/polynomial nature of such implementations.

This dissertation proposes polynomial abstractions over finite fields to model and represent the circuit constraints. Subsequently, decision procedures based on modern

computer algebra techniques – notably, Gröbner bases related theory and technology – are engineered to solve the verification problem efficiently. The arithmetic circuit is modeled as a polynomial system in the ring $\mathbb{F}_{2^k}[x_1, x_2, \cdots, x_d]$, and computer-algebra and algebraic-geometry based results (Hilbert's Nullstellensatz) over finite fields are exploited for verification. Two formulations are presented to address the implementation verification and the equivalence checking problems.

Gröbner basis theory is very powerful as it allows to decide many polynomial decision questions. However, the Gröbner basis computation (Buchberger's algorithm) is known to have double-exponential worst-case complexity in the input data. Therefore, straight-forward use of Gröbner basis engines for verification is infeasible for large circuits. To overcome this complexity, *we analyze the given circuit topology* to get more theoretical insights into the polynomial ideals corresponding to the circuit constraints. Based on this circuit information, we derive efficient *term orderings* to represent the polynomials. Subsequently, using the theory of Gröbner bases over finite fields, we prove that our term orderings render the set of polynomials itself a Gröbner basis – thus obviating the need for Buchberger's algorithm. We also analyze the optimality of our term orderings and identify a minimum number of computations required for verification via Gröbner basis reduction.

Using our approach, experiments are performed on a variety of custom-designed finite field arithmetic benchmark circuits. The results are also compared against contemporary methods, based on SAT and SMT solvers, BDDs, and AIG-based methods. Our tools can verify the correctness of, and detect bugs in, upto 163-bit circuits in $\mathbb{F}_{2^{163}}$; whereas contemporary approaches are infeasible beyond 48-bit circuits. Experimental results are analyzed and the advantages and limitations of our approaches are discussed. Finally, future research directions are discussed based on the work presented in this dissertation.

CHAPTER 1

INTRODUCTION

With the rapidly increasing complexity of hardware systems, verification of the correctness of designs poses serious challenges. Design flaws can be extremely costly. For example, the Intel Pentium floating point divide bug resulted in 475 million dollars of extra costs in 1993. In many safety-critical applications, such as cryptography systems, arithmetic bugs can be especially catastrophic. In [1], it is shown that incorrect (buggy) hardware can lead to full leakage of the secret key, which can compromise the security of such systems. Therefore, it is of utmost importance to verify the correctness of hardware designs.

1.1 Hardware Verification

Today, hardware verification averages about 70 percent of the overall hardware design effort and is believed to be the largest source of risk and cost. Hardware verification is becoming even more challenging as the design complexity increases.

The hardware design flow typically starts with a high-level specification or a property of the design. This specification is then translated into a register-transfer-level (RTL) description which is further optimized and translated to its corresponding netlist representation. Then the logic-level netlist is translated to a physical layout which is subsequently fabricated into integrated circuits. Fig.1.1 shows a typical design flow for realizing a hardware system. The design flow can be automated by Computer-Aided Design (CAD) tools available from both academia and industry. However, one critical question emerges: how to prove equivalent functionality between the different levels of representations? This is the objective of hardware verification. For example, after the RTL description is transformed into a gate-level netlist, it is important to ensure that its functionality remains the same. Similarly, after logic optimization is performed on the

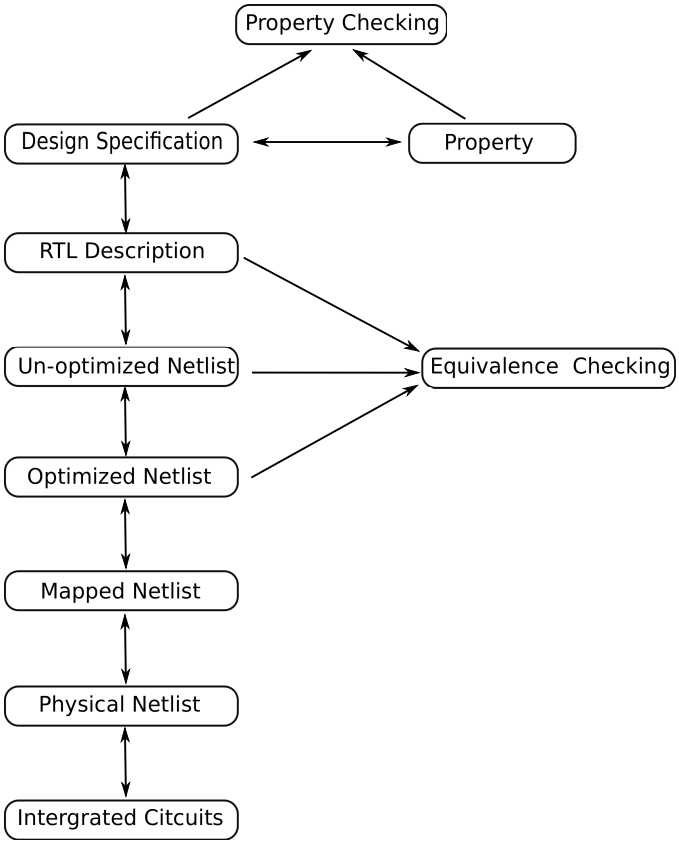

Figure 1.1. Typical circuit design and verification flow.

gate-level un-optimized netlist, it has to be ensured that the optimization process does not introduce a bug in the original design. Therefore, as shown in Fig. 1.1, verification is needed between different levels of abstractions, i.e., between design specification and the "golden model", RTL-level model and netlist-level model, and between un-optimized and optimized netlists, etc.

There are two main methodologies applied to hardware verification: simulation and formal verification. In a traditional design flow, simulation is the primary methodology for design validation. The effectiveness of simulation is achieved by exhaustive assignments of inputs to excite all possible behaviors of the system and then analyzing the output values. However, the increasing complexity of designs makes impossible for simulation to provide complete coverage.

In recent years, formal verification has emerged as an alternative technique to ensure the correctness of hardware designs, overcoming some of the limitations of simulation.

Formal verification is the process of utilizing mathematical theory to reason about the correctness of hardware designs. Formal verification in hardware usually takes one of the forms: property checking and equivalence checking. Property checking is a process of checking whether a design conforms to its given behavior or properties. Equivalence checking is conducted to prove the equivalent functionality of two given designs. Usually, equivalence checking is applied at various stages of the design cycle to verify correctness of the applied transformations. Figure 1.1 shows the role of equivalence checking in a typical hardware design flow.

Techniques utilized by property checking include model checking, theorem proving and approaches that integrate both. Equivalence checking makes use of Binary Decision Diagrams (BDDs), Satisfiability (SAT) solvers, and And-Inverter-Graph (AIG) based reductions, among others. As an emerging technique for equivalence checking, computer-algebra based decision procedures are gaining popularity. This kind of verification technique is believed to be more sophisticated in verifying arithmetic hardware designs in that they exploit the powerful applications of mathematics rather than ad-hoc techniques.

1.1.1 Property Checking

Property verification refers to proving the correspondence between designs and given properties. Usually property verification is achieved by two main formal methods: theorem proving and model checking.

Theorem proving [2] requires the existence of mathematical descriptions for both the specification and implementation, allowing these descriptions to be manipulated in a formal mathematical framework. Theorem provers apply primitive proof (mathematical) rules to a specification in order to derive new properties of a specification. Through this way, theorem proving can reduce a proof goal to simpler sub-goals that can be easily proved/disproved automatically by primitive proof steps. The benefit of this approach is its generality and completeness. However, despite several advances, generating the proof requires extensive guidance from the user. As a result, theorem proving lacks the level of automation that is desirable for a CAD framework to be practically useful. Theorem proving has gained commercial use in verifying that division and other operations are

correctly implemented in processors at AMD and Intel.

Model checking [3] is an approach to formally verifying finite-state systems. Properties about the system are modeled as temporal logic formulas, and the model defined by the system is traversed to check if the properties hold or not. Therefore, model checking consists of specifying the desired properties of the system and checking if there are violations of specified properties for all possible behaviors of the system.

Model checking is one of the most successful approaches for property verification up to date. Model checking tools [4] [5] [6] have achieved a significant level of automation and maturity and are widely in use in both academia and industry. A good aspect of model checking that is extremely important in practice is the ability to generate counterexamples. Such counterexamples provide a way to trace the incorrect behaviors (bugs). However, these tools tend to be memory intensive and are more applicable to at most medium sized designs or at the block-level, rather than at the system-level.

1.1.2 Equivalence Checking

Equivalence checking is used to formally prove that two representations of circuit designs have exactly equivalent functionality. As shown in Fig. 1.1, once a high-level representation is validated (by simulation or property checking), it is transformed into a gate-level netlist so that logic synthesis tools can be used to optimize the design according to the desired area/delay/power constraints. Then the design proceeds through a varied set of optimization and transformation operations. During various transformation stages, different implementations of the design, or parts of the design, are examined depending upon the constraints, such as area, performance, testability, etc. As the design is modified by replacing one of its components by another equivalent implementation, it needs to be verified whether or not the modified design is functionally equivalent to the original one.

Equivalence checking has important applications in arithmetic circuit verification. Hardware designs contain a large number of custom-designed circuits such as adders, multipliers, dividers, and so on. Such circuits are usually not synthesized by CAD tools because of area and performance constraints. Therefore, this raises the potential for errors/bugs in the implementation. Consequently, it remains a challenge to conduct equivalence checking for these large scale arithmetic circuits.

As an intensively investigated topic, techniques and approaches for equivalence checking have been well established. With various techniques employed for equivalence checking, BDDs and SAT-based techniques are the two dominant approaches widely used in both academia and industry. BDD-based approaches try to construct canonical representations of given circuits and conduct a linear comparison to determine whether they are equivalent or not. SAT based equivalence checking approaches try to find the unsatisfiability of a "miter" representing two designs.

There are also many promising generalizations of SAT and BDDs: Binary Moment Diagrams (BMDs) which have shown their superiority of verifying integer multipliers [7], and Satisfiability Modulo Theories (SMT) solvers which are the next generation of SAT. These approaches, to some extent, have gained some successes in equivalence checking. However, these approaches are beginning to show signs of inadequacy in two cases. First, large scale hardware designs still hinder the equivalence checking as the level of design complexity grows rapidly. For example, the verification of a 16-bit modular multiplier becomes infeasible for the current SAT/BDD based approaches. Secondly, for structurally similar circuits, this problem can be efficiently solved using the techniques of AIG based reductions [8] and subsequent use of circuit-SAT solvers [9]. However, when the circuits are functionally equivalent but structurally very dissimilar, none of the contemporary techniques, including BDDs, SAT and AIG-based approaches, are able to prove equivalence.

Ideally, approaches for equivalence checking should maintain a high-level of abstraction while still retaining sufficient information so as to not lose lower-level of functional details [10]. For instance, implementing arithmetic functions at bit-level can provide highly optimized implementations while word-level abstraction usually has much less structural information for solvers to analyze.

Arithmetic Bit Level (ABL) [11] abstraction techniques come close to achieving these requirements by extracting an arithmetic bit level representation from a given circuit. Through this way, the method can use the ABL information to prune the search space of SAT solvers. The drawback of this approach is it can only identify ABL information locally when analyzing the given circuit, which results in a exponential

blowup when looking at sophisticated circuits consisting of several arithmetic blocks.

Focus of this work: In this dissertation, we focus on equivalence checking problems for finite field arithmetic circuits. Such circuits are found in many applications such as in cryptography, coding theory, signal processing, among others. We utilize the theory of computer-algebra and algebraic-geometry, notably, Gröbner Bases related theory and technology, as the underlying verification engines. Our approach is sophisticated enough to take into account both high-level (word-level) specifications and low-level (bit-level) implementation details.

1.2 Computer Algebra Based Formal Verification

The first computer algebra based verification technique dates back to 1996 when Gröbner bases were utilized for SAT solving and formal verification [12]. Indeed, there have been many attempts to solve verification problems using Gröbner basis formulations [13] [14] [15]. The standard flow of these approaches is:

1. The verification problem is first formulated as a polynomial system.

2. The polynomial system is fed into a Gröbner basis engine to check whether the desired property is satisfied.

The critical step of this approach is the Gröbner basis computation. Unfortunately, the computation is known to have worst-case double-exponential complexity in the input data. In practice, Gröbner basis algorithms have not been capable of satisfactorily solving problems derived from real-world applications. Besides, these methods are employed for verification by modeling constraints over the Boolean level $\mathbb{Z}_2$; word-level abstractions, which can be powerfully modeled in algebra, are not utilized.

Recent advances [16] [17] [18] [19] [20] suggest a new direction of utilizing computer algebra theory to conduct hardware verification. These works show that it is feasible to overcome the complexity of Gröbner basis algorithm by efficiently engineering the integration of Gröbner bases theory and circuit analysis techniques.

1.3 Objective and Contributions of this Dissertation

This dissertation focuses on verification of hardware implementations of arithmetic circuits over finite fields of the type $\mathbb{F}_{2^k}$. Specifically, the following verification problems are addressed:

1. Formal verification of a custom-designed finite field arithmetic circuit implementation against its given word-level polynomial specification.

2. Gate-level equivalence checking of two finite field arithmetic circuit implementations.

Verification of only *combinational logic circuits* over finite fields is considered in this work. Sequential circuit verification is a very different problem for arithmetic circuits – and it is beyond the scope of this dissertation.

The *motivation* for this work stems from applications in cryptography circuits; though our techniques can be applied to verify arbitrary finite field arithmetic circuits. In cryptosystems, the datapath size (operand size) k in the circuits can very large. For example, the U.S. National Institute for Standards and Technology (NIST) recommends the use of finite fields corresponding to datapath sizes of $k = 163$-bits or more. The large size and high complexity of such circuits makes design verification quite challenging. Indeed, contemporary combinational verification techniques are unable to verify such large arithmetic circuits.

1.3.1 Contributions of this Dissertation

We propose the application of *computer-algebra techniques*, notably, *Gröbner bases* related theory and technology [21] [22], as the underlying verification framework for our applications. The advantage of using computer-algebra techniques is that it allows to integrate finite field arithmetic, circuit models and algebraic reasoning in a common verification framework. The circuits are modeled as a system of multi-variate polynomials in the field $\mathbb{F}_{2^k}$. The formal verification problem is then formulated using *Hilbert's Nullstellensatz* [23] as ideal membership testing. A Gröbner basis engine is subsequently employed as a decision procedure to solve this verification problem.

Gröbner basis theory is very powerful as it enables one to solve many polynomial decision questions. Unfortunately, the computational algorithms are known to have worst-case double-exponential complexity in the input data. Therefore, in order to make verification practical and scalable, we engineer efficient application of Gröbner basis by integrating it with circuit analysis techniques. Specifically, we analyze the topology of the given circuit and derive efficient *variable and term orders* to systematically represent and manipulate the polynomials. Subsequently, using the theory of Gröbner bases over finite fields, we prove that our term orderings impose specific constraints on the polynomials that can *obviate the need to compute a Gröbner basis*. Under this term ordering, either the polynomials themselves constitute a Gröbner basis, or the term ordering allows us to identify a minimum number of computations in the Gröbner basis algorithm that are sufficient for verification. This significantly scales verification – we are able to verify circuits for which contemporary verification methods are infeasible. To further improve our approach, we implement an efficient polynomial reduction (division) algorithm that operates on a matrix-based representation of the polynomial system.

Experiments are conducted over various custom-designed arithmetic circuits over $\mathbb{F}_{2^k}$. These include three different modulo-multiplier architectures and point-addition circuits used in elliptic curve cryptosystems. Using our approach and tools, we can verify the correctness of, and detect bugs in, up to 163-bit finite field arithmetic circuits, whereas contemporary approaches are infeasible.

1.4 Thesis Organization

The rest of this dissertation is organized as follows. Chapter 2 reviews previous approaches and highlights their drawbacks with respect to the given verification problem. Chapter 3 briefly describes the construction and properties of finite fields $\mathbb{F}_{2^k}$. Arithmetic circuit design over such fields is also reviewed to shed some light on the difficulty of the verification problem. Chapter 4 covers preliminary theoretical background related to computer-algebra, algebraic-geometry and Gröbner bases. Chapter 5 describes our approach to verify a circuit implementation against a word-level polynomial specification using ideal membership testing. We show how the Gröbner basis computation can

be obviated using efficient term orderings derived from the given circuit. Chapter 6 presents our approach to equivalence checking of two arithmetic circuit implementations. Efficient term orderings and matrix-based polynomial reduction procedures are derived. Chapter 7 describes a hierarchical verification methodology to verify arithmetic circuits over composite fields $\mathbb{F}_{(2^m)^n}$, where $k = m \cdot n$. Finally, Chapter 8 concludes the dissertation with a perspective on current and future research directions on computer algebra methods for verification.

CHAPTER 2

PREVIOUS WORK AND LIMITATIONS

Equivalence checking has been extensively investigated and many well-developed theories and techniques have been successfully applied in both academica and industry. The fundamental techniques used in equivalence checking include BDDs [24] and SAT solvers [25]. Recently, Gröbner bases based approaches are also gaining popularity. This chapter reviews widely used techniques in equivalence checking domain and discusses their limitations.

2.1 BDDs and Their Variants

Reduced Ordered Binary Decision Diagrams (ROBDDs or BDDs) are a canonical Directed Acyclic Graph (DAG) representation of a Boolean function. Circuits are usually described as a DAG. Two functionally equivalent circuits can be represented by the same BDDs. Therefore, equivalence checking between two circuits can be simply achieved by a comparison of their BDDs.

BDDs have found wide applications in many verification problems, including equivalence checking of arithmetic circuits, symbolic model checking [26] [5], among many others. However, along with the increasing complexity of designs, the size-explosion problem of BDDs becomes a bottleneck for many applications. This problem becomes especially serious when applied on designs containing large arithmetic data-path units. For example, BDD representation of multipliers requires memory that is exponential in the number of variables. As a result, BDDs fail to represent multipliers beyond 16-bit. As an attempt to control the exponential size, partitioned BDDs [27] introduce intermediate variables to represent sub-BDDs, thus partitioning the original BDD. Unfortunately, it is an intractable problem to find an optimum partition. This issue renders partitioned ROBDDs impractical for general verification problems.

Other efforts to extend the capabilities of BDDs are derived from generic Word Level Decision Diagrams (WLDDs) which are graph-based representations for functions with a Boolean domain and an integer range. These representations include ADDs [28], *BMDs [7], etc. A thorough review of WLDDs can be found in [29].

Algebraic decision diagrams (ADDs) [28] provide an efficient means for representing and performing arithmetic operations on functions from the binary domain ($\{0,1\}$) to the integer domain, i.e., $\{0,1\} \rightarrow \mathbb{Z}$. However, the mapping/decomposition at each node/variable is still binary and leads to exactly two terms. Restricting the decomposition to a binary type limits the abstraction of integer variables, as they have to be decomposed into their constituent bits. Consequently, ADDs face the same problem that BDDs do: the exponential size of the number of input bits.

BMDs [7] and their variants, such as HDDs [30], K*BMD [31], among others, perform a moment-based decomposition of a linear function. BMDs represent binary variables as $(0,1)$ integers instead of Boolean variables. Moment diagrams provide a concise representation of integer-valued functions defined over vectors of bits, or words, such as $X = 2^{n-1}x_{n-1} + \ldots + 2x_1 + x_0$, for an n-bit word X, where each x_i is a binary variable. BMDs are linear in size for integer multiplier circuits, as shown in Figure 2.1. The multiplicative constants of this representation reside in the terminal nodes. Moreover, the constants can also be represented as multiplicative terms and assigned to the edges of the graph, giving a rise to the Multiplicative Binary Moment Diagram (*BMD) [7]. Several rules for manipulating edge weights are imposed on the graph to

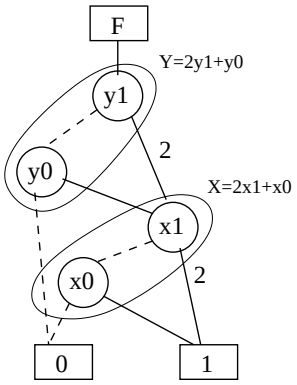

Figure 2.1. BMD for $F = x * y$; x, y are 2-bit wide, F is 4-bits wide.

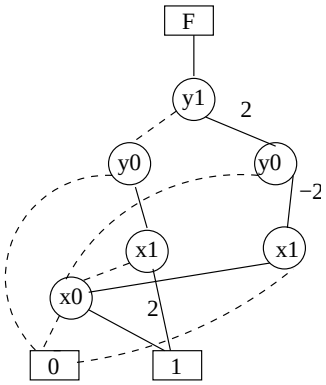

Figure 2.2. BMD for $F = x * y$; x, y, F are all 2-bits wide.

ensure canonicity.

One of the main limitations of BMDs is that performing some arithmetic operations on functions represented by BMDs is very expensive. For example, for an n-bit vector X, the BMD for X^k requires $O(n^k)$ nodes. In addition, BMDs for modular operations on bit-vectors are distorted, losing the compactness of word-level expression. One such example is depicted in Figure 2.2.

Taylor Expansion Diagrams (TEDs) [32] [33] [34] [35] are derived from Taylor series and canonical DAG representations for functions that can be abstracted as polynomials. TEDs represent bit-vectors ($x_0, x_1, \ldots, x_{n-1}$) as algebraic symbols ($X[0 : n-1]$), raising the abstraction from bits (Boolean) to words (integers). Let $f(x, y, \ldots)$ be a real differentiable function. Using the Taylor series expansion with respect to a variable x, the function f can be represented as

$$\begin{aligned} f(x, y, \ldots) \;=\; & f(x = 0, y, \ldots) \;+\; x \cdot f'(x = 0, y, \ldots) \;+ \\ & (1/2)x^2 \cdot f''(x = 0, y, \ldots) \;+ \cdots \end{aligned} \tag{2.1}$$

The derivatives of f at $x = 0$ are independent of x, and can be further decomposed w.r.t the remaining variables, one variable at a time. This resulting recursive decomposition can be represented using a non-binary tree called the TED, with memory requirements much smaller than other representations. TEDs are applicable to modeling, symbolic simulation and equivalence verification, provided that a polynomial abstraction is feasible. For binary operations, the diagram reduces to a *BMD, inheriting all its limitations.

Besides, TEDs cannot model modulo operations over bit-vectors. Therefore TEDs are incapable of solving the equivalence problems presented in this dissertation.

2.2 SAT solvers and SMT solvers

The SAT problem is a decision-problem. In principle, any decidable decision problem can be modeled in terms of SAT, and because of this, SAT solvers are used in an enormous variety of applications.

The objective of SAT solvers is to find variable assignments such that the given constraints (formulas) can be satisfied. If this is not possible, SAT solvers have to prove that no assignments satisfy the constraints (UNSAT).

Solving SAT-instances of any useful size was not possible until the introduction of the Davis-Putnam (DP) [36] algorithm. The DP algorithm works by eliminating variables through deriving new constraints from the original constraints containing the variables. Still, this has its limitations: though the variable is eliminated, the cost of elimination can be large because of the clauses needed to represent the variable in its absence. As a result, the algorithm did not see much use, but was used as a stepping stone for a more versatile technique based on searching.

The foundation of nearly all modern SAT solvers lies in the DPLL approach [25]. DPLL algorithm adopts a technique called backtracking search, whereby variables are recursively assigned, simplifying the formula at each step, building candidates to the solutions, abandoning each partial solution that is not possibly be completed to a valid solution (backtracking). DPLL algorithm also utilizes rules such as unit-propagation and pure-literal elimination to reduce formula size and reduce the number of decisions needed. However, in essence, DPLL algorithm is an exhaustive search for satisfying assignment.

Based on the basic DPLL framework, many improvements have been proposed. A major advance is conflict driven clause learning [37]. Conflict driven clause learning takes a strategy that new clauses are learned from conflicts during backtrack search and the structure of conflicts is exploited during clause learning. With this technique, the size of problem search space is greatly reduced and SAT solvers achieve the performance improvement by orders of magnitude. However, there are still many problems that are in-

tractable for SAT solvers, such as problems from cryptography domain where the designs often involve tens of millions of variables. One major drawback that limits the capacity of SAT solvers is the lack of ability for word-level reasoning. To resolve this limitation, Satisfiability modulo theories (SMT) are proposed and have gained significant popularity since 2003. The SMT problem is to decide the satisfiability of a formula expressed in a first-order background theory, such as linear inequalities, bit vectors, linear arithmetic and uninterpreted functions, etc. In fact, SMT can be considered as an extension of SAT to first-order logic. In other words, SMT solvers first apply highly optimized decision procedures for different first-order theories and then check the satisfiability using SAT solvers. For example, $X > Y \wedge Y = Z$ is first interpreted into $X > Z$ and then $X > Z$ is fed into a SAT solver to check the satisfiability.

For our problems of interest, bit-vector (BV) theories have been shown to be useful and important for hardware equivalence checking. In our case, equivalence checking problems are first compiled into the formula. Then decision procedures for bit-vector theories, such as term rewriting techniques, are applied on the compiled formula to obtain further optimization. Next, the optimized formula is bit-blasted to an equisatisfiable Boolean formula. Finally, an integrated SAT solver is used to enumerate assignments to the Boolean formula to find a satisfying assignment.

One advantage of bit-vector theories in SMT is that all problems are described and operated upon word-level (bit-vector), proving to be effective for computationally intensive designs, such as arithmetic circuits. For example, at word level, a 32-bit multiplication can be represented as one term with two 32-bit words, while at bit-level, it is represented as thousands of Boolean variables. Moreover, some instances can be fully decided on the word-level, thus achieving a high performance.

As mentioned above, SMT formulas obviously provide a much richer modeling language than what is possible with Boolean SAT formulas, even allowing word-level representations of datapath operations. Solvers based on these theories [38] [39] [40] [41] have improved abilities to represent arithmetic computations, but ultimately rely on SAT tools to solve the verification instance, making them prone to the same limitations, as shown in our experiments. For equivalence checking of gate-level circuits, word-level

information is not available. Then SMT solvers have no benefits as they have to rely on SAT solvers to solve the bit-level verification instance.

2.2.1 Circuit Based Solvers

The above SAT and SMT solvers do not take into consideration circuit topology, thus inefficient in verifying circuit designs. Instead, circuit based solvers, such as C-SAT [9] [42], focus specifically on the mechanics of checking the equivalence of pairs of combinational circuits. The main strategy utilized by C-SAT solver is signal correlation guided learning, which attempts to identify common sub-circuit structure. In other words, an internal node in the first circuit may be equivalent to an internal node in the second circuit, thus combining the identical sub-circuit as one node. Though this way, if two circuits are structurally similar, the original problem becomes a problem with much smaller space. To identify the common sub-circuits, a technique called *structural hashing* [8] is used. This is achieved by random simulation: first sending random vectors through the two circuits and then collecting pairs of candidate equivalent nodes. Practical use [8] has shown that this technique can detect potentially many, high probability, candidate equivalent nodes.

AIG [43], on the other hand, is a pseudo canonical representation of a circuit. One good property of AIGs is the operations based on AIG are fast, such as adding nodes, merging nodes. By representing the circuit with AIGs, many equivalent nodes over a large circuit can be identified quickly.

When coupled with AIG as the circuit representation and techniques used in C-SAT, circuit based SAT solvers can achieve remarkable speedups in solving a wide variety of circuit equivalence checking problems.

When two circuits are structurally very dissimilar, structural hashing is able to identify the common sub-circuits, thus reducing the problem size. However, these techniques are infeasible when verifying structurally dissimilar circuits. For example, in our experiments, we have shown that equivalence checking of Mastrovito versus Montgomery multipliers using ABC [8] and C-SAT [9] is infeasible beyond 16-bit circuits.

2.3 Computer Algebra Based Approaches

Computer algebra based approaches were first proposed in 1996 for SAT solving and formal verification [12] [13]. The principle idea of these approaches is to reason about the existence of solutions in polynomial domain: verification problems are first formulated as polynomials; then the polynoial system is fed into a Gröbner basis engine to check the existence of solutions. There have been many attempts to solve verification problems using this Gröbner basis formulations [15]. Instead of analyzing the entire problems for proof-refutation, the work of [14] utilized Gröbner bases to preprocess SAT instance to obtain additional information about the problem. This information is then fed back into the SAT solver, thus benefiting the SAT solving.

One limitation of these approaches is that the Gröbner basis computation which is known to have worst-case double-exponential complexity in the input data. Besides, in practice, the implementations of Gröbner basis algorithm have not been capable of satisfactorily solving problems derived from real-world applications.

Recent advances [16] [18] suggest a new direction of utilizing computer algebra theory to conduct hardware verification. It is feasible to overcome the complexity of Gröbner basis algorithm by efficiently engineering Gröbner bases theory and integration of circuit analysis techniques.

The work described in [16] addresses verification of finite precision integer datapath circuits using the concepts of Gröbner bases over the ring $\mathbb{Z}_{2^k}$. They model the circuit constraints by way of arithmetic-bit-level (ABL) polynomials ($\{G\}$), and formulate the verification test as an equivalent variety subset problem. To solve this, first they derive a term order that already makes $\{G\}$ a Gröbner basis. Then they compute a normal form f of the specification g w.r.t. $\{G\}$. If f is a vanishing polynomial over $\mathbb{Z}_{2^k}$ [44], circuit correctness is established. In [18], the authors further show that the vanishing polynomial test can be omitted by formulating the problem directly over $Q := \mathbb{Z}_{2^k}[X]/\langle x^2 - x : x \in X\rangle$.

However, such approaches are effective only over ring $\mathbb{Z}_{2^k}$ while our problems are derived from finite fields $\mathbb{F}_{2^k}$. The mathematical theories differ significantly in these two domains. Therefore, these approaches cannot be applied for our problems.

2.4 Verification of Finite Field Applications

There has not been much research by the design verification community to verify finite field applications. The following works that specifically targeted automated decision procedures for verification of finite field applications: [45] [46] [47].

The theorem-proving approach of [45] verifies a finite field $\mathbb{F}_{2^k}$ implementation against a given polynomial specification. They devise a decision procedure based on polynomial division, variable elimination, term re-writing, etc., and demonstrate a correctness proof of a sub-block of a Reed-Solomon decoder. Their decision procedures were partly built upon BDDs (requiring decision over $\mathbb{F}_2$), and that is infeasible for large circuits.

The work of [46] solves similar problems as those of [45]. However, they make use of OKFDDs [48] to canonically represent the circuit constraints. Moreover, instead of verifying circuits over $\mathbb{F}_{2^k}$ directly, [46] verifies the circuits over its equivalent composite field $\mathbb{F}_{(2^m)^n}$ representation, where a *non-prime* $k = m \cdot n$. Their approach has no benefit if k is prime – say, when $k = 163$ for elliptic curves. Moreover, the size-explosion of FDDs limits their approach to 16-bit ($\mathbb{F}_{2^{16}}$) circuits, as shown in their experiments.

MODDs [49] were proposed as a canonical representation of the characteristic function of a circuit over finite field $\mathbb{F}_{2^k}$. However, as each node in the DAG may have up to k children, MODDs have been shown to be exponential in the number of variables, thus infeasible beyond 32-bit circuits.

None of the above approaches provide a scalable and efficient solution to the problem of verification of large finite field arithmetic circuits.

CHAPTER 3

PRELIMINARIES

This chapter gives an account of basic communicative algebra objects, such as modular arithmetic, groups, rings, fields and polynomials. Emphasis is placed on finite fields and hardware design over such fields as these applications are the focus of this dissertation. The material is referred from [50] [51] [52] for finite field concepts and [53] [54] [55] [56] [57] for hardware design over finite fields.

3.1 Rings, Fields and Polynomials

Definition 3.1 *An* **abelian group** *is a set* $\mathbb{S}$ *and a binary operation* $'+'$ *satisfying:*

- *Closure Law: For every* $a, b \in \mathbb{S}, a + b \in \mathbb{S}$.
- *Associative Law: For every* $a, b, c \in \mathbb{S}, a + (b + c) = (a + b) + c$.
- *Commutativity: For every* $a, b \in \mathbb{S}, a + b = b + a$.
- *Existence of Identity: There is an identity element* $0 \in \mathbb{S}$ *such that for all* $a \in \mathbb{S}$; $a + 0 = a$.
- *Existence of Inverse: If* $a \in \mathbb{S}$*, then there is an element* $a^{-1} \in \mathbb{S}$ *such that* $a+a^{-1} = 0$.

The set of integers $\mathbb{Z}$, for instance, forms an abelian group under addition.

Definition 3.2 *Given two binary operations* $'+'$ *and* $'\cdot'$ *on the set* $\mathbb{R}$ *as well as two distinguished elements* $0, 1 \in \mathbb{R}$*, the system* $\mathbb{R}$ *is called a* **ring** *if the following properties hold:*

- $\mathbb{R}$ *forms an abelian group under the '+' operation with additive identity element* 0.

- *Distributive Laws: For all* $a, b, c \in \mathbb{R}$, $a \cdot (b + c) = a \cdot b + a \cdot c$.

- *Associative Law of Multiplication: For every* $a, b, c \in \mathbb{R}$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

If there is an identity element $1 \in \mathbb{R}$ such that for all $a \in \mathbb{R}$, $a \cdot 1 = a = 1 \cdot a$, then $\mathbb{R}$ is said to be a **ring with unity**.

The ring $\mathbb{R}$ is **commutative** if the following law also holds:

- *Commutative Law of Multiplication: For every* $a, b \in \mathbb{R}$, $a \cdot b = b \cdot a$.

Henceforth, we consider only commutative rings with unity, as defined above. The set of integers, $\mathbb{Z}$, and the set of rational numbers, $\mathbb{Q}$, are examples of commutative rings with unity.

Definition 3.3 *The* **modular number system** *with base* n *is a set of positive integers* $Z_n = \{0, 1, \ldots, n-1\}$*, with the two operations* $'+'$ *and* $'.'$ *satisfying the properties below:*

$$(a + b) \pmod{n} \equiv (a \pmod{n} + b \pmod{n}) \pmod{n} \tag{3.1}$$

$$(a \cdot b) \pmod{n} \equiv (a \pmod{n} \cdot b \pmod{n}) \pmod{n} \tag{3.2}$$

$$(-a) \pmod{n} \equiv (n - a) \pmod{n} \tag{3.3}$$

Example 3.1 *The set* $Z_8 = \{0, 1, \ldots, 7\}$ *denotes the modular number system with base* 8*. Examples of some operations performed* $\pmod 8$ *are:*

$$\begin{array}{rclrrcl} 3+6 &=& 9 \pmod 8 = 1 & \qquad 2 \cdot 4 &=& 8 \pmod 8 = 0 \\ 5+7 &=& 12 \pmod 8 = 4 & \qquad 3 \cdot 5 &=& 15 \pmod 8 = 7 \\ (-3) &=& 8-3 \pmod 8 = 5 & \qquad 3 \cdot (-3) &=& (3 \cdot 5) \pmod 8 = 7 \end{array}$$

The modular number system $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, where n is a natural number, forms a commutative ring with the identity elements 0 and 1. This type of a ring is a *finite integer ring*, where addition and multiplication are defined *modulo n* $\pmod n$. Many hardware and software applications perform bit-vector arithmetic. Arithmetic over

k-bit vectors manifests itself as algebra over the finite inter ring $\mathbb{Z}_{2^k}$, as a k-bit vector represents integer values from $\{0, ..., 2^k - 1\}$.

Example 3.2 *Consider the following hardware description given in Verilog. It takes as inputs two* 4*-bit vectors, and computes the sum which is also represented with a* 4*-bit wide vector. Therefore, addition is performed modulo* 2^4.

```
module Adder(A, B, sum);

    input [3:0] A;
    input [3:0] B;
    output [3:0] sum;
    reg [3:0] sum;

    always @ (A or B)
    begin
        sum <= A + B;
    end

endmodule
```

This code exemplifies arithmetic computations over the ring Z_{2^k} *implemented at bit-vector level.*

Definition 3.4 *A* **field** $\mathbb{F}$ *is a commutative ring with unity, where every non-zero element in* $\mathbb{F}$ *has a multiplicative inverse; i.e.* $\forall\ a \in \mathbb{F} - \{0\}$, $\exists\ \hat{a} \in \mathbb{F}$ *such that* $a \cdot \hat{a} = 1$.

A field is defined over a ring with an extra condition: the presence of a multiplicative inverse for all non-zero elements. Therefore, a field must be a ring while a ring is not necessarily a field. For example, the set $\mathbb{Z}_{2^k} = \{0, 1, \cdots, 2^k - 1\}$ forms a finite ring. However, $\mathbb{Z}_{2^k}$ is not a field because not every element in $\mathbb{Z}_{2^k}$ has a multiplicative inverse.

In general, fields can be infinite, or contain a finite number of elements. For example, fractions $\mathbb{Q}$, complex numbers $\mathbb{C}$, are infinite fields. In our applications, we focus on finite fields which are described later in Section 3.2.

Definition 3.5 *Let* $\mathbb{R}$ *be a ring. A* **polynomial** *over* $\mathbb{R}$ *in the indeterminate* x *is an expression of the form:*

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k = \sum_{i=0}^{k} a_i x^i, \forall a_i \in \mathbb{R}. \tag{3.4}$$

The constants a_i are the coefficients and k is the degree of the polynomial. For example, $4x^2 + 6x$ is a polynomial in x over $\mathbb{Z}$, with coefficients 4 and 6 and degree 2.

Definition 3.6 *The system consisting of the set of all polynomials in the indeterminate x with coefficients in the ring $\mathbb{R}$, where addition and multiplication are defined accordingly, forms a ring called the* **ring of polynomials** *$\mathbb{R}[x]$. Similarly, $\mathbb{R}[x_1, x_2, \cdots, x_n]$ represents the ring of multivariate polynomials with coefficients in $\mathbb{R}$.*

For example, $\mathbb{Z}_{2^3}[x]$ stands for the system of all polynomials in x with coefficients in $\mathbb{Z}_{2^3}$; $4x^2 + 6x$ is an instance of a polynomial belonging to $\mathbb{Z}_{2^3}[x]$.

3.2 Finite Fields

Finite fields find widespread applications in computer engineering, such as in error correcting codes, elliptic curve cryptography, digital signal processing, testing of VLSI circuits, among others. We describe the relevant finite field concepts [50] [51] [52] and hardware designs over such fields [53] [54] [55] [56] [57].

Definition 3.7 *A* **finite field***, also called a Galois field, is a field with a finite number of elements. The number of elements q of the finite field is a power of a prime integer – i.e. $q = p^k$, where p is a prime integer, and $k \geq 1$. Finite fields are denoted as $\mathbb{F}_q$ or $\mathbb{F}_{p^k}$.*

Definition 3.8 *The* **characteristic** *of a finite field $\mathbb{F}$ with unity element 1 is the smallest integer n such that $1 + \cdots + 1$ (n times$) = 0$.*

Lemma 3.1 *The characteristic of a finite field $\mathbb{F}_{p^k}$ is the prime integer p.*

Lemma 3.2 *The finite integer ring $\mathbb{Z}_n$ forms a finite field if and only if n is prime. Such fields are customarily denoted as $\mathbb{Z}_p = \mathbb{F}_p$.*

Example 3.3 *Consider the field $\mathbb{Z}_5$. The additive and multiplicative inverses of each element in $\mathbb{Z}_5$ (except 0) are also elements in $\mathbb{Z}_5$, as shown in Table 3.1. In contrast, $\mathbb{Z}_4$*

Table 3.1. Additive and multiplicative inverses in $\mathbb{Z}_5$.

element	additive inverse	multiplicative inverse
0	0	undefined
1	4	1
2	3	3
3	2	2
4	1	4

is not a field, as 2 *does not have a multiplicative inverse in* $\mathbb{Z}_4$.

While $\mathbb{Z}_{2^k}$ is not a field, there do exist fields $\mathbb{F}_{p^k}$ with non-prime cardinality. Such fields are called extension fields. We are interested in extension fields $\mathbb{F}_{p^k}$, where $p = 2$ and $k > 1$. As these are algebraic extensions of the binary field $\mathbb{F}_2$, they are generally termed as *binary extension fields* $\mathbb{F}_{2^k}$. Such fields are most widely used in digital hardware applications as the computation can be universally encoded in binary form for practical reasons.

3.2.1 Construction of Finite Fields $\mathbb{F}_{2^k}$

To construct and describe the properties of finite fields $\mathbb{F}_{2^k}$, the concept of **irreducible polynomials** is required:

Definition 3.9 *A polynomial* $P(x) \in \mathbb{F}_2[x]$ *is* **irreducible** *if* $P(x)$ *is nonconstant with degree* k *and it cannot be factored into a product of polynomials of lower degree in* $\mathbb{F}_2[x]$.

Therefore, a polynomial with degree k is irreducible over $\mathbb{F}_2$ if and only if it has no roots in $\mathbb{F}_2$. For example, x^2+x+1 is an irreducible polynomial, because $x^2+x+1=0$ has no roots in $\mathbb{F}_2$. Irreducible polynomials of any arbitrary degree always exist in $\mathbb{F}_2[x]$.

To construct $\mathbb{F}_{2^k}$, we take the polynomial ring $\mathbb{F}_2[x]$ and an irreducible polynomial $P(x) \in \mathbb{F}_2[x]$ of degree k, and construct $\mathbb{F}_{2^k} \equiv \mathbb{F}_2[x] \pmod{P(x)}$. Let α be a root of $P(x)$, i.e., $P(\alpha) = 0$. Note that $P(x)$ is irreducible in $\mathbb{F}_2[x]$, however, the root lies in the algebraic extension $\mathbb{F}_{2^k}$. Any element $A \in \mathbb{F}_{2^k}$ can therefore be represented as:

$$A = \sum_{i=0}^{k-1} (a_i \cdot \alpha^i) = a_0 + a_1 \cdot \alpha + \cdots + a_{k-1} \cdot \alpha^{k-1} \tag{3.5}$$

where $a_i \in \mathbb{F}_2$ are the coefficients and $P(\alpha) = 0$. The degree of any element A in $\mathbb{F}_{2^k}$ is always less than k. This is because A is always computed modulo $P(x)$, and $P(x)$ has degree k. The remainder ($(\mathrm{mod}\ P(x))$) can be of degree at most $k-1$. For this reason, the field $\mathbb{F}_{2^k}$ can be viewed as a k-dimensional vector space over $\mathbb{F}_2$. The equivalent bit vector representation for element A is given below:

$$A = (a_{k-1} a_{k-2} \cdots a_0) \tag{3.6}$$

The example below explains the construction of the finite field $\mathbb{F}_{2^4}$.

Example 3.4 *Let us construct* $\mathbb{F}_{2^4}$ *as* $\mathbb{F}_2[x]\ (\mathrm{mod}\ P(x))$*, where* $P(x) = x^4 + x^3 + 1 \in \mathbb{F}_2[x]$ *is an irreducible polynomial of degree* $k = 4$*. Let* α *be the root of* $P(x)$*, i.e.* $P(\alpha) = 0$*.*

Any element $A \in \mathbb{F}_2[x]\ (\mathrm{mod}\ x^4 + x^3 + 1)$ *has a representation of the type:* $A = a_3x^3 + a_2x^2 + a_1x + a_0$ *(degree* < 4*) where the coefficients* $a_3, \ldots, a_0$ *are in* $F_2 = \{0, 1\}$*. Since there are only* 16 *such polynomials, we obtain* 16 *elements in the field* $\mathbb{F}_{2^4}$*. Each element in* $\mathbb{F}_{2^4}$ *can then be viewed as a* 4*-bit vector over* $\mathbb{F}_2$*:* $\mathbb{F}_{2^4}$=$\{(0000), (0001), \ldots$ (1110),$(1111)\}$*. If* α *is the root of* $P(x)$*, then each element also has an exponential representation; all three representations are shown in Table 3.2. For example, consider the element* α^{12}*. Computing* $\alpha^{12}\ (\mathrm{mod}\ \alpha^4 + \alpha^3 + 1) = \alpha + 1 = (0011)$*; hence we have the three equivalent representations.*

There may exist more than one irreducible polynomials with degree k. In such cases, any degree k irreducible polynomial can be used for field construction. For example, both $x^3 + x^2 + 1$ and $x^3 + x + 1$ are irreducible in $\mathbb{F}_2$ and either one can be used to construct $\mathbb{F}_{2^3}$. This is due to the following result:

Theorem 3.1 *There exist a* **unique** *field* $\mathbb{F}_{p^k}$*, for any prime* p *and any positive integer* k*.*

Theorem 3.1 implies that finite fields with the same number of elements are isomorphic to each other up to the labeling of the elements.

Table 3.2. Bit-vector, Exponential and Polynomial representation of elements in $\mathbb{F}_{2^4} = \mathbb{F}_2[x] \pmod{x^4 + x^3 + 1}$

$a_3a_2a_1a_0$	Exponential	Polynomial	$a_3a_2a_1a_0$	Exponential	Polynomial
0000	0	0	1000	α^3	α^3
0001	1	1	1001	α^4	$\alpha^3 + 1$
0010	α	α	1010	α^{10}	$\alpha^3 + \alpha$
0011	α^{12}	$\alpha + 1$	1011	α^5	$\alpha^3 + \alpha + 1$
0100	α^2	α^2	1100	α^{14}	$\alpha^3 + \alpha^2$
0101	α^9	$\alpha^2 + 1$	1101	α^{11}	$\alpha^3 + \alpha^2 + 1$
0110	α^{13}	$\alpha^2 + \alpha$	1110	α^8	$\alpha^3 + \alpha^2 + \alpha$
0111	α^7	$\alpha^2 + \alpha + 1$	1111	α^6	$\alpha^3 + \alpha^2 + \alpha + 1$

Lemma 3.3 *Let A be any element in $\mathbb{F}_q$, then $A^{q-1} = 1$.*

As a consequence of Lemma 3.3, the following is a very important result that we will use to investigate solutions to polynomial equations in $\mathbb{F}_q$.

Theorem 3.2 $[Generalized\ Fermat's\ Little\ Theorem]$ *Given a finite field $\mathbb{F}_q$, each element $A \in \mathbb{F}_q$ satisfies:*

$$\begin{aligned} A^q &\equiv A \\ A^q - A &\equiv 0 \end{aligned} \tag{3.7}$$

As a polynomial extension of the above consequence, let $x^q - x$ be a polynomial in $\mathbb{F}_q[x]$. Every element $A \in \mathbb{F}_q$ is a solution to $x^q - x = 0$. Therefore, $x^q - x$ always *vanishes* in $\mathbb{F}_q$, and such polynomials are called **vanishing polynomials** of the field $\mathbb{F}_q$.

Example 3.5 *Given $\mathbb{F}_{2^2} = \{0, 1, \alpha, \alpha + 1\}$ with $P(x) = x^2 + x + 1$, where $P(\alpha) = 0$.*

$$\begin{aligned} 0^{2^2} &= 0 \\ 1^{2^2} &= 1 \\ \alpha^{2^2} &= \alpha \pmod{\alpha^2 + \alpha + 1} \\ (\alpha + 1)^{2^2} &= \alpha + 1 \pmod{\alpha^2 + \alpha + 1} \end{aligned}$$

3.2.2 Hardware Implementations of Arithmetic Operations Over $\mathbb{F}_{2^k}$

In some cases, finite field (primitive) computations such as ADD, MUL, etc., are implemented in hardware, and algorithms are then implemented in software (e.g. crypto-processors [58] [59]). In other cases, the entire design can be implemented in hardware – such as a one-shot Reed-Solomon encoder-decoder chip [60] [61], or the point multiplication circuitry [62] used in elliptic curve cryptosystems. Therefore, there has been a lot of research in VLSI implementations of finite field arithmetic. We describe the design of such primitive computations below to shed some light on the architectures and their design and verification complexity.

Addition in $\mathbb{F}_{2^k}$ is performed by correspondingly adding the polynomials together, and reducing the coefficients of the result modulo the characteristic 2.

Example 3.6 *Given* $A = \alpha^3 + \alpha^2 + 1 = (1101)$ *and* $B = \alpha^2 + 1 = (0101)$ *in* $\mathbb{F}_{2^4}$,

$$A + B = (\alpha^3 + \alpha^2 + 1) + (\alpha^2 + 1) = (\alpha^3) + (\alpha^2 + \alpha^2) + (1 + 1) = \alpha^3 = (1000).$$

Example 3.7 *A* 4*-bit adder in* $\mathbb{F}_{2^4}$ *is given in Figure 3.1. It takes as inputs two* 4*-bit vectors:* $A = (a_3a_2a_1a_0)$, $B = (b_3b_2b_1b_0)$ *and computes the result* $Z = (z_3z_2z_1z_0)$. *Note an adder circuit is trivial and only consists of XOR gates.*

Conceptually, the multiplication $Z = A \times B \pmod{P(x)}$ in $\mathbb{F}_{2^k}$ consists of two steps. In the first step, the multiplication $A \times B$ is performed, and in the second step, the

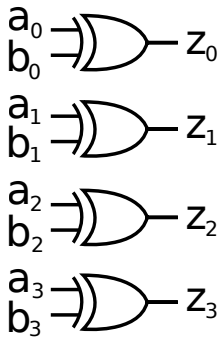

Figure 3.1. 4-bit adder over $\mathbb{F}_{2^4}$.

result is reduced modulo the irreducible polynomial $P(x)$. Multiplication procedure is shown in Example 3.8.

Example 3.8 *Consider the field $\mathbb{F}_{2^4}$. We take as inputs: $A = a_0 + a_1 \cdot \alpha + a_2 \cdot \alpha^2 + a_3 \cdot \alpha^3$ and $B = b_0 + b_1 \cdot \alpha + b_2 \cdot \alpha^2 + b_3 \cdot \alpha^3$, along with the irreducible polynomial $P(x) = x^4 + x^3 + 1$. We have to perform the multiplication $Z = A \times B \pmod{P(x)}$. The coefficients of $A = \{a_0, \ldots, a_3\}, B = \{b_0, \ldots, b_3\}$ are in $\mathbb{F}_2 = \{0, 1\}$. Multiplication can be performed as shown below:*

			a_3	a_2	a_1	a_0
$\times$			b_3	b_2	b_1	b_0
			$a_3 \cdot b_0$	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_0 \cdot b_0$
		$a_3 \cdot b_1$	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_0 \cdot b_1$	
	$a_3 \cdot b_2$	$a_2 \cdot b_2$	$a_1 \cdot b_2$	$a_0 \cdot b_2$		
$a_3 \cdot b_3$	$a_2 \cdot b_3$	$a_1 \cdot b_3$	$a_0 \cdot b_3$			
s_6	s_5	s_4	s_3	s_2	s_1	s_0

The result $Sum = s_0 + s_1 \cdot \alpha + s_2 \cdot \alpha^2 + s_3 \cdot \alpha^3 + s_4 \cdot \alpha^4 + s_5 \cdot \alpha^5 + s_6 \cdot \alpha^6$,

$$
\begin{aligned}
s_0 &= a_0 \cdot b_0 \\
s_1 &= a_0 \cdot b_1 + a_1 \cdot b_0 \\
s_2 &= a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 \\
s_3 &= a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_2 + a_3 \cdot b_1 \\
s_4 &= a_1 \cdot b_3 + a_2 \cdot b_1 + a_3 \cdot b_1 \\
s_5 &= a_2 \cdot b_3 + a_3 \cdot b_2 \\
s_6 &= a_3 \cdot b_3
\end{aligned}
$$

Here the multiply "$\cdot$" and add "$+$" operations are performed modulo 2, so they can be implemented in a circuit using AND and XOR gates. Note that unlike integer multipliers, there are no carry-chains in the design, as the coefficients are always reduced modulo $p = 2$. However, the result is yet to be reduced modulo the primitive polynomial $P(x) = x^4 + x^3 + 1$. This is shown below, where higher degree coefficients are reduced $\pmod{P(x)}$.

s_3	s_2	s_1	s_0	
s_4	0	0	s_4	$s_4 \cdot \alpha^4 \pmod{P(\alpha)} = s_4 \cdot (\alpha^3 + 1)$
s_5	0	s_5	s_5	$s_5 \cdot \alpha^5 \pmod{P(\alpha)} = s_5 \cdot (\alpha^3 + \alpha + 1)$
s_6	s_6	s_6	s_6	$s_6 \cdot \alpha^6 \pmod{P(\alpha)} = s_6 \cdot (\alpha^3 + \alpha^2 + \alpha + 1)$
z_3	z_2	z_1	z_0	

The final result (output) of the circuit is: $Z = z_0 + z_1\alpha + z_2\alpha^2 + z_3\alpha^3$*; where* $z_0 = s_0 + s_4 + s_5 + s_6;\ \ z_1 = s_1 + s_5 + s_6;\ \ z_2 = s_2 + s_6;\ \ z_3 = s_3 + s_4 + s_5 + s_6$.

The above multiplier design is called the *Mastrovito multiplier* [53] which is the most straightforward way to design a multiplier over $\mathbb{F}_{2^k}$. A logic circuit for a 4-bit *Mastrovito* multiplier over *finite field* $\mathbb{F}_{2^4}$ is illustrated in Fig.7.1.

Modular multiplication is at the heart of many public-key cryptosystems, such as Elliptic Curve Cryptography (ECC) [63]. Due to the very large field size (and hence the datapath width) used in these cryptosystems, the above *Mastrovito* multiplier architecture is inefficient, especially when exponentiation and repeat multiplications are performed. Therefore, efficient hardware and software implementations of modular multiplication

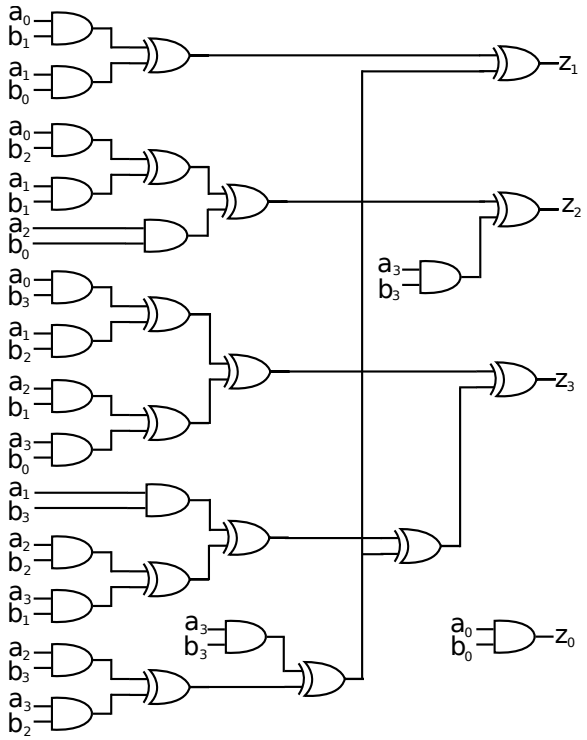

Figure 3.2. Mastrovito multiplier over $\mathbb{F}_{2^4}$.

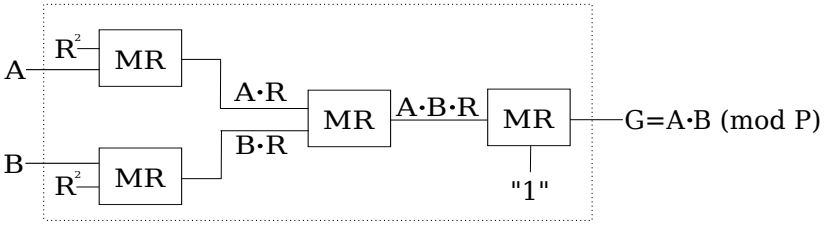

Figure 3.3. *Montgomery* multiplier over $\mathbb{F}_{2^k}$

algorithms are used to overcome the complexity of such operations. These include the Montgomery reduction [54] [55] and the Barrett reduction [57].

Montgomery Reduction: Montgomery reduction (MR) computes:

$$G = MR(A, B) = A \cdot B \cdot R^{-1} \pmod{P(x)} \tag{3.8}$$

where A, B are k-bit inputs, $R = \alpha^k$, R^{-1} is multiplicative inverse of R in $\mathbb{F}_{2^k}$, and $P(x)$ is the irreducible polynomial for $\mathbb{F}_{2^k}$. Since Montgomery reduction cannot directly compute $A \cdot B$, to compute $A \cdot B \pmod{P(x)}$, we need to pre-compute $A \cdot R$ and $B \cdot R$, as shown in Figure 3.3.

Each MR block in Figure 3.3 represents a Montgomery reduction step which is a hardware implementation of the algorithm shown in Algorithm 1.

Algorithm 1: Montgomery Reduction Algorithm [55]

Input: $A(x), B(x) \in \mathbb{F}_{2^k}$; irreducible polynomial $P(x)$.
Output: $G(x) = A(x) \cdot B(x) \cdot x^{-k} \pmod{P(x)}$.
$G(x)$:=0
for *(*$i = 0$*;* $i \leq k-1$*;* ++*i)* **do**
    $G(x) := G(x) + A_i \cdot B(x)$ /*A_i is the i^{th} bit of A*/;
    $G(x) := G(x) + G_0 \cdot P(x)$ /*G_0 is the lowest bit of G*/;
    $G(x) := G(x)/x$ /*Right shift 1 bit*/;
end

The design of Fig. 3.3 is an overkill to compute just $A \cdot B \pmod{P(x)}$. However, when these multiplications are performed repeatedly, such as in iterative squaring, then the Montgomery approach speeds-up the computation. As shown in [56], the critical

path delay and gate counts of a squarer designed using the Montgomery approach are much smaller than the traditional approaches.

Barrett Reduction: Barrett reduction is the other widely used multiplier design method adopted in cryptography system designs. Similar to Montgomery reduction, the traditional Barrett reduction, proposed in [64], needs a pre-computed value of the reciprocal/inverse of modulus $P(x)$. This pre-computation requires extra computational time and memory space. To overcome this limitation, the recent approach of [57] avoids such a pre-computation of inverses and therefore greatly simplifies the hardware design implementation. This algorithmic computation is shown in Algorithm 2.

Algorithm 2: Barrett Reduction Without Pre-Computation Algorithm [57]

Input: $R(x) \in \mathbb{F}_{2^k}$; irreducible polynomial $P(x) = x^n + \sum_{i=0}^{l} m_i \cdot x^i$ satisfying $l = \lfloor \frac{n}{2} \rfloor, m_i \in \{0, 1\}$.
Output: $G(x) = R(x) \pmod{P(x)}$.
$Q_1(x) = \frac{R(x)}{x^n}$ `/*Right shift` n `bit*/;`
$Q_2(x) = P(x) \cdot Q_1(x)$;
$Q_3(x) = \frac{Q_2(x)}{x^n}$;
$G_1(x) = R(x) \pmod{x^n}$ `/*Keep the lower` n `bits of` $R(x)$`*/;`
$G_2(x) = P(x) \cdot Q_3(x) \pmod{x^n}$;
$G(x) = G_1(x) + G_2(x)$;

Based on Barrett reduction, a multiplier can be designed with two simple steps: multiplication $R = A \times B$ and a subsequent Barrett reduction $G = R \pmod{P}$. This is shown in Figure 3.4. As we can see, a Barrett multiplier is similar to a Mastrovito multiplier except for the reduction step.

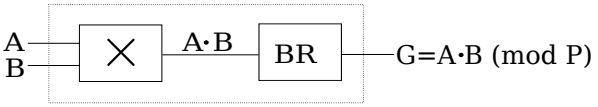

Figure 3.4. Barrett multiplier over $\mathbb{F}_{2^k}$.

One of the most influential applications of finite fields is in elliptic curve cryptography (ECC). ECC is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The main operations of encryption, decryption and authentication in ECC rely on *point multiplications*. Point multiplication involves a series of addition and doubling of points on the elliptic curve. A drawback of traditional point multiplication is that each point addition and doubling involves a multiplicative inverse operation over finite fields. Representing the points in projective coordinate systems [62] eliminates the need for multiplicative inverse operation and therefore increases the efficiency of point multiplication operation. In our experiments, we have verified custom designs based on L*ó*pez-Dahab (LD) coordinate system [65].

Example 3.9 *Consider point addition in LD projective coordinate. Given an elliptic curve:* $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$ *over* $\mathbb{F}_{2^k}$*, where* X, Y, Z *are* k*-bit vectors that are elements in* $\mathbb{F}_{2^k}$ *and similarly,* a, b *are constants from the field. Let (*X_1*,* Y_1*,* Z_1*) + (*X_2*,* Y_2*,* 1*) = (*X_3*,* Y_3*,* Z_3*) represent point addition over the elliptic curve. Then* X_3*,* Y_3*,* Z_3 *can be computed as follows:*

$$
\begin{aligned}
A &= Y_2 \cdot Z_1^2 + Y_1 \\
B &= X_2 \cdot Z_1 + X_1 \\
C &= Z_1 \cdot B \\
D &= B^2 \cdot (C + aZ_1^2) \\
Z_3 &= C^2 \\
E &= A \cdot C \\
X_3 &= A^2 + D + E \\
F &= X_3 + X_2 \cdot Z_3 \\
G &= X_3 + Y_2 \cdot Z_3 \\
Y_3 &= E \cdot F + Z_3 \cdot G
\end{aligned}
$$

Example 3.10 *Consider point doubling in projective coordinate system. Given an elliptic curve:* $Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$. *Let 2(*$X_1$, Y_1, Z_1*) = (*X_3, Y_3, Z_3*), then*

$$X_3 = X_1^4 + b \cdot Z_1^4$$
$$Z_3 = X_1^2 \cdot Z_1^2$$
$$Y_3 = bZ_1^4 \cdot Z_3 + X_3 \cdot (aZ_3 + Y_1^2 + bZ_1^4)$$

In the above examples, polynomoial multiplication and squaring operations are implemented in hardware using Montgomery or Barrett reductions over finite fields $\mathbb{F}_{2^k}$.

The field size for such applications is generally very large; as discussed before, for ECC, in $\mathbb{F}_{2^k}$, $k = 163$ or larger. Such large size and complicated arithmetic nature of such circuits clearly shows the complexity of the formal verification problem. Contemporary techniques lack the requisite power of abstraction to model and verify such large systems. For this reason, we propose polynomial abstractions over finite fields to model and verify such circuits using computer algebra techniques. This is the subject of subsequent chapters of this dissertation.

CHAPTER 4

COMPUTER ALGEBRA FUNDAMENTALS

This chapter reviews preliminary fundamental concepts of commutative and computer algebra that are utilized in our work. The concepts of polynomial ideals, varieties and Gröbner bases are described with regard to their algorithmic computation. Finally, the results of Hillbert's Nullstellensatz are described which are employed for verification over finite fields in subsequent chapters. The material is mostly referred from the textbooks [23] [22].

4.1 Monomials and Their Orderings

Definition 4.1 *A* **monomial** *in $x_1, x_2, \cdots, x_d$ is a product of this form:*

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \cdots \cdot x_d^{\alpha_d}, \tag{4.1}$$

where $\alpha_i \geq 0, i \in \{1, \cdots, d\}$. The total degree of the monomial is $\alpha_1 + \cdots + \alpha_d$.

For simplicity, we will denote a monomial $x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \cdots \cdot x_d^{\alpha_d} = x^{\alpha}$, where $\alpha = (\alpha_1, \cdots, \alpha_d)$, i.e., $\alpha \in \mathbb{Z}_{\geq 0}^d$.

Definition 4.2 *A* **multivariate polynomial** *f in variables $x_1, x_2, \ldots, x_d$ with coefficients in any given field $\mathbb{K}$ is a finite linear combination (with coefficients in $\mathbb{K}$) of monomials:*

$$f = \sum_{\alpha} a_\alpha \cdot x^\alpha, \;\; a_\alpha \in \mathbb{K}$$

The set of all polynomials in $x_1, x_2, \ldots, x_d$ with coefficients in field $\mathbb{K}$ is denoted by $\mathbb{K}[x_1, x_2, \ldots, x_d]$.

Definition 4.3 *Let $f = \sum_\alpha a_\alpha x^\alpha$ be a polynomial in $\mathbb{K}[x_1, x_2, \ldots, x_d]$.*

1. *We refer to the constant $a_\alpha \in \mathbb{K}$ as the coefficient of the monomial $a_\alpha x^\alpha$.*

2. *If $a_\alpha \neq 0$, we call $a_\alpha x^\alpha$ a term of f.*

As an example, $2x^2 + y$ is a polynomial with two terms $2x^2$ and y, with 2 and 1 as coefficients respectively. In contrast, $x + y^{-1}$ is not a polynomial because the exponent of y is less than 0.

An important fact of polynomials is that a polynomial is a sum of terms and these terms have to be arranged unambiguously so that they can be manipulated in a consistent manner. Therefore, we need establish the concept **monomial ordering (or term ordering)**. A term ordering, represented by $>$, defines how terms in a polynomial are ordered. Term-orderings are totally ordered, i.e. anti-symmetric, transitive, total, with constant terms last in the ordering. More formally, we have the following definitions:

Definition 4.4 *Let $\mathbb{T}^d = \{x^\alpha : \alpha \in \mathbb{Z}^d_{\geq 0}\}$ be the set of all monomials in $x_1, \ldots, x_d$. A* **monomial order** $>$ *on $\mathbb{T}^d$ is a total well-ordering satisfying:*

- *For any $x^\alpha \in \mathbb{T}^d$, $x^\alpha > 1$*
- *For all α, β, γ, $x^\alpha > x^\beta \Rightarrow x^\alpha \cdot x^\gamma > x^\beta \cdot x^\gamma$*

A total-order ensures that there is no ambiguity with respect to where a term is found in the term-ordering. Total orderings for monomials come in different forms, notably **lexicographic orderings** (lex), and its variants: **degree-lexicographic ordering** (deglex) and **reverse degree-lexicographic ordering** (revdeglex).

A **lexicographic ordering** (lex) is a total-ordering $>$ such that variables in the terms are lexicographically ordered. Higher variable-degrees take precedence over lower degrees (e.g. $a^3 = aaa$).

Definition 4.5 Lexicographic order: *Let $x_1 > x_2 > \cdots > x_d$ lexicographically. Also let $\alpha = (\alpha_1, \ldots, \alpha_d)$; $\beta = (\beta_1, \ldots, \beta_d) \in \mathbb{Z}^d_{\geq 0}$. Then we have:*

$$x^\alpha > x^\beta \iff \begin{cases} \textit{Starting from the left, the first co-ordinates of } \alpha_i, \beta_i \\ \textit{that are different satisfy } \alpha_i > \beta_i \end{cases} \tag{4.2}$$

A **degree-lexicographic ordering** (deglex) is a total-ordering $>$ such that the total degree of a term takes precedence over the lexicographic ordering. A **degree-reverse-**

lexicographic ordering (degrevlex) is the same as a deglex ordering, however terms are lexed in reverse.

Definition 4.6 Degree Lexicographic order: *Let $x_1 > x_2 > \cdots > x_d$ lexicographically. Also let $\alpha = (\alpha_1, \ldots, \alpha_d);\ \beta = (\beta_1, \ldots, \beta_d) \in \mathbb{Z}^d_{\geq 0}$. Then we have:*

$$x^\alpha > x^\beta \iff \begin{cases} \sum_{i=1}^d \alpha_i > \sum_{i=1}^d \beta_i & \text{or} \\ \sum_{i=1}^d \alpha_i = \sum_{i=1}^d \beta_i \text{ and } x^\alpha > x^\beta & \text{w.r.t. lex order} \end{cases} \tag{4.3}$$

Definition 4.7 Degree Reverse Lexicographic order: *Let $x_1 > x_2 > \cdots > x_d$ lexicographically. Also let $\alpha = (\alpha_1, \ldots, \alpha_d);\ \beta = (\beta_1, \ldots, \beta_d) \in \mathbb{Z}^d_{\geq 0}$. Then we have:*

$$x^\alpha > x^\beta \iff \begin{cases} \sum_{i=1}^d \alpha_i > \sum_{i=1}^d \beta_i \text{ or} \\ \sum_{i=1}^d \alpha_i = \sum_{i=1}^d \beta_i \text{ and the first co-ordinates} \\ \alpha_i, \beta_i \text{ from the right, which are different, satisfy } \alpha_i < \beta_i \end{cases} \tag{4.4}$$

As a consequence of these term orderings, we have the following relations, where $a > b > c$.

$$\text{lex:} a^2b > a^2 > abc > ab > ac^2 > ac > b^2c > b^2 > bc^3 > 1 \tag{4.5}$$

$$\text{deglex:} bc^3 > a^2b > abc > ac^2 > b^2c > a^2 > ab > ac > b^2 > 1 \tag{4.6}$$

$$\text{degrevlex:} bc^3 > a^2b > abc > b^2c > ac^2 > a^2 > ab > b^2 > ac > 1 \tag{4.7}$$

The difference between the *lex* and two *deg-* orderings is obvious, while the difference between the two degree-based orderings can be seen by considering from which direction the term is lexed, e.g. $ac^2 > b^2c$ (deglex, left-to-right) versus $b^2c > ac^2$ (degrevlex, right-to-left).

Example 4.1 *Let $f = 2x^2yz + 3xy^3 - 2x^3$. Effects of different term orderings on f are shown below:*

- *lex $x > y > z$: $f = -2x^3 + 2x^2yz + 3xy^3$*
- *deglex $x > y > z$: $f = 2x^2yz + 3xy^3 - 2x^3$*
- *degrevlex $x > y > z$: $f = 3xy^3 + 2x^2yz - 2x^3$*

Definition 4.8 *The* **leading term** *is the first term in a term-ordered polynomial. Likewise, the leading coefficient is the coefficient of the leading term. Finally, a leading power product is the leading term lacking the coefficient. We use the following notation:*

$$lt(f) \quad \text{— Leading Term} \tag{4.8}$$

$$lc(f) \quad \text{— Leading Coefficient} \tag{4.9}$$

$$lm(f) \quad \text{— Leading Monomial} \tag{4.10}$$

Example 4.2

$$f = 3a^2b + 2ab + 4bc \tag{4.11}$$

$$lt(f) = 3a^2b \tag{4.12}$$

$$lc(f) = 3 \tag{4.13}$$

$$lm(f) = a^2b \tag{4.14}$$

4.2 Varieties and Ideals

In verification applications, it is often required to analyze (the presence or absence of) solutions to a given system of constraints. In our applications, these constraints are polynomials and their solutions are described as **varieties**.

Definition 4.9 *Let $\mathbb{K}$ be a field, and let $f_1, \ldots, f_s \in \mathbb{K}[x_1, x_2, \ldots, x_d]$. We call $V(f_1, \ldots, f_s)$ the* **affine variety** *defined by $f_1, \ldots, f_s$ as:*

$$V(f_1, \ldots, f_s) = \{(a_1, \ldots, a_d) \in \mathbb{K}^d : f_i(a_1, \ldots, a_d) = 0, \forall i, 1 \leq i \leq s\}. \tag{4.15}$$

$V(f_1, \ldots, f_s) \in \mathbb{K}^d$ is **the set of all solutions** of the system of equations: $f_1(x_1, \ldots, x_d) = \cdots = f_s(x_1, \ldots, x_d) = 0$.

Example 4.3 *Given $\mathbb{R}[x, y]$, $V(x^2 + y^2) = \{(0, 0)\}$. Similarly, in $\mathbb{R}[x, y]$, $V(x^2 + y^2 - 1) = \{all\ points\ on\ the\ circle : x^2 + y^2 - 1 = 0\}$. However, varieties depend on which field we are operating on. For the same polynomial $x^2 + 1$, we have:*

- *In $\mathbb{R}[x]$, $V(x^2 + 1) = \emptyset$.*

- *In* $\mathbb{C}[x]$, $V(x^2+1) = \{(\pm i)\}$.

The above example shows the variety can be infinite, finite (non-empty set) or empty. It is interesting to note that we will be operating over finite fields $\mathbb{F}_q$, and any finite set of points is a variety. Consider the points $\{(a_1, \ldots, a_d) : a_1, \ldots, a_d \in \mathbb{F}_q\}$ in $\mathbb{F}_q^d$. Any single point is a variety of some polynomial system: e.g. $(a_1, \ldots, a_d)$ is a variety of $x_1 - a_1 = x_2 - a_2 = \cdots = x_d - a_d = 0$. Moreover, **finite unions and finite intersections** of varieties are also varieties. Let $U = V(f_1, \ldots, f_s)$ and $W = V(g_1, \ldots, g_t)$. Then:

- $U \cap W = V(f_1, \ldots, f_s, g_1, \ldots, g_t)$
- $U \cup W = V(f_i g_j : 1 \leq i \leq s, 1 \leq j \leq t)$

Another important concept related to varieties is that the variety depends not just on the given system of polynomial equations,but rather on the **ideal** generated by the polynomials.

Definition 4.10 *A subset* $I \subset \mathbb{K}[x_1, x_2, \ldots, x_d]$ *is an* **ideal** *if it satisfies:*

- $0 \in I$
- I *is closed under addition:* $x, y \in I \Rightarrow x + y \in I$
- *If* $x \in \mathbb{K}[x_1, x_2, \ldots, x_d]$ *and* $y \in I$*, then* $x \cdot y \in I$ *as well as* $y \cdot x \in I$.

Any ideal is generated by its *basis* or *generators*.

Definition 4.11 *Let* $f_1, f_2, \ldots, f_s$ *be the given elements of* $\mathbb{K}[x_1, x_2, \ldots, x_d]$*. Let* I *be an ideal in* $\mathbb{K}[x_1, x_2, \ldots, x_d]$*. If:*

$$I = \{g_1 f_1 + g_2 f_2 + \ldots + g_s f_s : g_1, \ldots, g_s \in \mathbb{K}[x_1, x_2, \ldots, x_d]\} \tag{4.16}$$

then, $f_1, \ldots, f_s$ *are called the* **basis (or generators)** *of the ideal* I *and correspondingly* I *is denoted as* $I = \langle f_1, f_2, \ldots, f_s \rangle$.

Example 4.4 *The set of even integers, which is a subset of the ring of integers* Z*, forms an ideal of* Z*. This can be seen from the following;*

- 0 *belongs to the set of even integers.*
- *The sum of two even integers* x *and* y *is always an even integer.*
- *The product of any integer* x *with an even integer* y *is always an even integer.*

Example 4.5 *Given* $\mathbb{R}[x, y]$, $I = \langle x, y\rangle$ *is an ideal containing all polynomials generated by* x *and* y*, such as* $x^2 + y$, $x \cdot y + x$. $J = \langle x^2, y^2\rangle$ *is an ideal containing all polynomials generated by* x^2 *and* y^2*, such as* $x^2 + y^2$, $x^2 \cdot y^2 + x^{10}$. *Notice* $I \neq J$ *because* $x + y$ *can only be generated by* I.

Any ideal may have many different bases. For instance, it is possible to have different sets of polynomials $\{f_1, \ldots, f_s\}$ and $\{g_1, \ldots, g_t\}$ that may generate the same ideal, i.e., $\langle f_1, \ldots, f_s\rangle = \langle g_1, \ldots, g_t\rangle$. Since variety depends on the ideal, these sets of polynomials have the same solutions.

Proposition 4.1 *If* $f_1, \ldots, f_s$ *and* $g_1, \ldots, g_t$ *are bases of the same ideal in* $\mathbb{K}[x_1, \ldots, x_d]$*, so that* $\langle f_1, \ldots, f_s\rangle = \langle g_1, \ldots, g_t\rangle$*, then* $V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$.

Example 4.6 *Consider the two bases* $F_1 = \{(2x^2 + 3y^2 - 11, x^2 - y^2 - 3\}$ *and* $F_2 = \{x^2 - 4, y^2 - 1\}$. *These two bases generate the same ideal, i.e.,* $\langle F_1\rangle = \langle F_2\rangle$. *Therefore, they represent the same variety, i.e.,*

$$V(F_1) = V(F_2) = \{\pm 2, \pm 1\}. \tag{4.17}$$

An important fundamental problem that we need to solve is one of ideal membership testing.

Definition 4.12 *Let* $f, f_1, \ldots, f_s$ *be polynomials in* $\mathbb{K}[x_1, \ldots, x_d]$. *Let ideal* $I = \langle f_1, \ldots, f_s\rangle \subset \mathbb{K}[x_1, \ldots, x_d]$. *If* f *can be written as* $f = f_1 h_1 + \cdots + f_s h_s$*, then we say* f *is a member of the ideal* I.

Our verification problems are formulated as ideal membership testing. For this purpose, we require a decision procedure to unequivocally decide ideal membership. Gröbner basis provides such a decision procedure, and this is described in the next section.

4.3 Gröbner Bases

As mentioned above, different generating sets may constitute the same ideal. However, some generating sets may be better than others – that is they may be a better representation of the ideal. A **Gröbner basis** is one such ideal representation that has many important properties that allow to solve many polynomial decision questions. By analyzing the Gröbner basis, one can deduce the presence or absence of solutions (varieties), find the dimension of the varieties, and also deduce ideal membership. A Gröbner basis, in essence, is a canonical representation of an ideal. Buchberger's work [21] laid the foundation for computing a Gröbner basis of an ideal. This section provides a synopsis of some of these concepts.

Among many equivalent definitions of Gröbner bases, we start with the definition that can best describe the properties of Gröbner bases:

Definition 4.13 *A set of non-zero polynomials $G = \{g_1, \ldots, g_t\}$ contained in an ideal I, is called a* **Gröbner basis** *for I if and only if for all $f \in I$ such that $f \neq 0$, there exists $i \in \{1, \ldots, t\}$ such that $lm(g_i)$ divides $lm(f)$.*

$$G = \textit{GröbnerBasis}(I) \iff \forall f \in I : f \neq 0, \exists g_i \in G : lm(g_i) \mid lm(f) \tag{4.18}$$

Given a set of polynomials $F = \{f_1, \ldots, f_s\}$ that generate ideal $I = \langle f_1, \ldots, f_s \rangle$, Buchberger gives an algorithm to compute a Gröbner basis $G = \langle g_1, \ldots, g_t \rangle$. This algorithm relies on the notions of S-polynomials and polynomial reduction, which are described below.

Definition 4.14 *For a field $\mathbb{K}$, $f, g \in \mathbb{K}[x_1, \ldots, x_d]$, $L = lcm\,(lt(f), lt(g))$, an* **S-polynomial** *$Spoly(f, g)$ is defined as:*

$$Spoly(f, g) = \frac{L}{lt(f)} \cdot f - \frac{L}{lt(g)} \cdot g \tag{4.19}$$

Note lcm denotes least common multiple.

Definition 4.15 *The* **reduction** *of a polynomial f, by another polynomial g, to a reduced polynomial r is denoted:*

$$f \xrightarrow{g} r$$

Reduction is carried out using multivariate, polynomial long division.

For sets of polynomials, the notation

$$f \xrightarrow{F}_{+} r$$

represents the reduced polynomial r resulting from f as reduced by a set of non-zero polynomials $F = \{f_1, \ldots, f_s\}$. The polynomial r is considered **reduced** *if $r = 0$ or no term in r is divisible by a $lm(f_i), \forall f_i \in F$.*

For all intents and purposes, the reduction process $f \xrightarrow{F}_{+} r$, of dividing a polynomial f by a set of polynomials of F, can be modeled as repeated long-division of f by each of the polynomials in F until no further reductions can be made—the result of which is r, as shown in Algorithm 3.

Algorithm 3: Polynomial Division

Input: $f, f_1, \ldots, f_s$
Output: $r, a_1, \ldots, a_s$, such that $f = a_1 \cdot f_1 + \cdots + a_s \cdot f_s + r$.
$a_1 = a_2 = \cdots = a_s = 0$; $r = 0$;
$p := f$;
while $p \neq 0$ **do**
    i=1;
    divisionmark = false;
    while $i \leq s$ *&& divisionmark = false* **do**
        if f_i *can divide* p **then**
            $a_i = a_i + lt(p)/lt(f_i)$;
            $p = p - lt(p)/lt(f_i) \cdot f_i$;
            divisionmark = true;
        else
            i=i+1;
        end
    end
    if *divisionmark = false* **then**
        $r = r + lt(p)$;
        $p = p - lt(p)$;
    end
end

The division algorithm keeps cancelling the leading terms of polynomials until no more leading terms can be further cancelled. So the key step is $p = p - lt(p)/lt(f_i) \cdot f_i$, as the following example shows.

Example 4.7 *Given $f_1 = y^2 - x$ and $f_2 = y - x$ in $\mathbb{Q}[x, y]$ with $deglex$: $y > x$. Then $f_1/f_2 = f_1 - lt(f_1)/lt(f_2) \cdot f_2 = y^2 - x - (y^2/y) \cdot (y - x) = y \cdot x - x$. Then $y \cdot x - x$ can be further divided by f_2: $(y \cdot x - x)/f_2 = x^2 - x$ which is the final result.*

We now present Buchberger's Algorithm [21] for computing Gröbner bases.

Algorithm 4: Buchberger's Algorithm

Input: $F = \{f_1, \ldots, f_s\}$, such that $I = \langle f_1, \ldots, f_s \rangle$
Output: $G = \{g_1, \ldots, g_t\}$, a Gröbner basis of I
$G := F$;
repeat
  $G' := G$;
  for *each pair* $\{f_i, f_j\}, i \neq j$ *in* G' **do**
    $Spoly(f_i, f_j) \xrightarrow{G'}_+ r$;
    if $r \neq 0$ **then**
      $G := G \cup \{r\}$;
    end
  end
until $G = G'$;

For Gröbner basis computation, a monomial (term) ordering is fixed to ensure that polynomials are manipulated in a consistent manner. Buchberger's algorithm then takes pairs of polynomials (f_i, f_j) in the basis G and combines them into "S-polynomials" (Spoly(f_i, f_j)) to cancel leading terms. The S-polynomial is then reduced (divided) by all elements of G to a remainder r, denoted as $S(f_i, f_j) \xrightarrow{G}_+ r$. Multivariate polynomial division is used for this reduction step. This process is repeated for all unique pairs of polynomials, including those created by newly added elements, until no new polynomials are generated; ultimately constructing the Gröbner basis.

Example 4.8 *Consider the ideal $I \subset \mathbb{Q}[x, y]$, $I = \langle f_1, f_2 \rangle$, where $f_1 = yx - y,\ f_2 = y^2 - x$. Assume a degree-lexicographic term ordering with $y > x$ is imposed.*

First, we need to compute $Spoly(f_1, f_2) = x \cdot f_2 - y \cdot f_1 = y^2 - x^2$. *Then we conduct a polynomial reduction* $y^2 - x^2 \xrightarrow{f_2} x^2 - x \xrightarrow{f_1} x^2 - x$. *Let* $f_3 = x^2 - x$. *Then* G *is updated as* $\{f_1, f_2, f_3\}$. *Next we compute* $Spoly(f_1, f_3) = 0$. *So there is no new polynomial generated. Similarly, we compute* $Spoly(f_2, f_3) = x \cdot y^2 - x^3$, *followed by* $x \cdot y^2 - x^3 \xrightarrow{f_1} y^2 - x^3 \xrightarrow{f_2} x - x^3 \xrightarrow{f_2} 0$. *Again, no polynomial is generated. Finally,* $G = \{f_1, f_2, f_3\}$.

Gröbner basis now gives a decision procedure to test for membership in an ideal.

Theorem 4.1 *Let* $G = \{g_1, \cdots, g_t\}$ *be a Gröbner basis for an ideal* $I \subset \mathbb{K}[x_1, \cdots, x_d]$ *and let* $f \in \mathbb{K}[x_1, \ldots, x_d]$. *Then* $f \in I$ *if and only if the remainder on division of* f *by* G *is zero.*

In other words,

$$f \in I \iff f \xrightarrow{G}_+ 0 \tag{4.20}$$

Example 4.9 *Consider Example 4.8. Let* $f = y^2x - x$ *be another polynomial. Note that* $f = yf_1 + f_2$, *so* $f \in I$. *If we divide* f *by* f_1 *first and then by* f_2, *we will obtain a zero remainder. However, since the set* $\{f_1, f_2\}$ *is not a Gröbner basis, we find that the reduction* $f \xrightarrow{f_2} x^2 - x \xrightarrow{f_1} x^2 - x \neq 0$; *i.e. dividing* f *by* f_2 *first and then by* f_1 *does not lead to a zero remainder. However, if we compute the Gröbner basis* G *of* I, $G = \{x^2 - x, yx - y, y^2 - x\}$, *dividing* f *by polynomials in* G *in any order will always lead to the zero remainder. Therefore, one can decide ideal membership unequivocally using the Gröbner basis.*

Definition 4.16 *A* **minimal Gröbner basis** *for a polynomial ideal* I *is a Groebner basis* G *for* I *such that*

- $lc(g_i) = 1, \forall g_i \in G$
- $\forall g_i \in G$, $lt(g_i) \notin \langle lt(G - \{g_i\})\rangle$

A **minimal** Gröbner basis is a Gröbner basis such that no leading term of any element in G divides another in G. A minimal Gröbner basis can be computed by removing any polynomial whose leading term can be divided by another in a given Gröbner basis.

A minimal Gröbner basis can be further reduced.

Definition 4.17 *A reduced Gröbner basis for a polynomial ideal I is a Gröbner basis $G = \{g_1, \ldots, g_t\}$ such that:*

- $lc(g_i) = 1, \forall g_i \in G$
- $\forall g_i \in G$, *no monomial of* g_i *lies in* $\langle lt(G - \{g_i\})\rangle$

G is a reduced Gröbner basis when no monomial of any element in G divides the leading term of another element.

For a given monomial ordering, the reduced Gröbner basis is a canonical representation of the ideal, as given by Proposition 4.2 below.

Proposition 4.2 *Let $I \neq \{0\}$ be a polynomial ideal. Then, for a given monomial ordering, I has a unique reduced Gröbner basis.*

4.4 Hillbert's Nullstellensatz

In this section, we further describe some correspondence between ideals and varieties in the context of algebraic geometry. The celebrated results of Hillbert's Nullstellensatz establish such correspondences, and these results, together with Gröbner bases, provide a basis for our verification solutions.

Definition 4.18 *A field $\overline{\mathbb{K}}$ is an algebraically closed field if every polynomial in one variable with degree at least 1, with coefficients in $\overline{\mathbb{K}}$, has a root in $\overline{\mathbb{K}}$.*

In other words, any non-constant polynomial equation over $\overline{\mathbb{K}}[x]$ always has at least one root in $\overline{\mathbb{K}}$. Every field $\mathbb{K}$ is contained in an algebraically closed one $\overline{\mathbb{K}}$. For example, the field of reals $\mathbb{R}$ is not an algebraically closed field, because $x^2 + 1 = 0$ has no root in $\mathbb{R}$. However, $x^2 + 1 = 0$ has roots in the field of complex numbers $\mathbb{C}$, which is an algebraically closed field. In fact, $\mathbb{C}$ is the algebra closure of $\mathbb{R}$. Every algebraically closed field is an infinite field.

Theorem 4.2 [**Weak Nullstellensatz**] *Let $I \subset \overline{\mathbb{K}}[x_1, x_2, \cdots, x_d]$ be an ideal satisfying $V(I) = \emptyset$. Then $I = \overline{\mathbb{K}}[x_1, x_2, \cdots, x_d]$, Or equivalently,*

$$V(I) = \emptyset \iff I = \overline{\mathbb{K}}[x_1, x_2, \cdots, x_d] = \langle 1 \rangle \tag{4.21}$$

Corollary 4.1 *Let $I = \langle f_1, \ldots, f_s \rangle \subset \overline{\mathbb{K}}[x_1, x_2, \cdots, x_d]$. Let G be the reduced Gröbner basis of I. Then $V(I) = 0 \iff G = \{1\}$.*

The *Weak Nullstellensatz* offers a way to evaluate whether or not the system of multivariate polynomial equations (ideal I) has common solutions in $\overline{\mathbb{K}}^d$. For this purpose, we only need to check if the ideal is generated by the unit element, i.e., $1 \in I$. This approach can be used to evaluate the feasibility of constraints in our verification problems. Another interesting result that we will employ is one of **Strong Nullstellensatz** to describe which we need the concepts of "ideals of varieties" and radicals.

Let $\mathbb{K}$ be any field and let $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{K}^d$ be a point, and $f \in \mathbb{K}[x_1, \ldots, x_d]$ be a polynomial. We say that f *vanishes* on $\mathbf{a}$ if $f(\mathbf{a}) = 0$, i.e., $\mathbf{a}$ is in the variety of f.

Definition 4.19 *For any variety V of $\mathbb{K}^d$, the ideal of polynomials that vanish on V, called the vanishing ideal of V, is defined as $I(V) = \{f \in \mathbb{F}[x_1, \ldots, x_d] : \forall \mathbf{a} \in V, f(\mathbf{a}) = 0\}$.*

Proposition 4.3 *If a polynomial f vanishes on a variety V, then $f \in I(V)$.*

Example 4.10 *Let ideal $J = \langle x^2, y^2 \rangle$. Then $V(J) = \{(0, 0)\}$. All polynomials in J will obviously agree with the solution and vanish on this variety. However, the polynomials x, y are not in J but they also vanish on this variety. Therefore, $I(V(J))$ is the set of all polynomials that vanish on $V(J)$, and the polynomials x, y are members of $I(V(J))$.*

Definition 4.20 *Let $J \subset \mathbb{K}[x_1, \ldots, x_d]$ be an ideal. The radical of J is defined as $\sqrt{J} = \{f \in \mathbb{K}[x_1, \ldots, x_d] : \exists m \in \mathbb{N}, f^m \in J\}$.*

Example 4.11 *Let $J = \langle x^2, y^2 \rangle \subset \mathbb{K}[x, y]$. Note neither x nor y belongs to J, but they belong to $\sqrt{J}$. Similarly, $x \cdot y \notin J$, but since $(x \cdot y)^2 = x^2 \cdot y^2 \in J$, therefore, $x \cdot y \in \sqrt{J}$.*

When $J = \sqrt{J}$, then J is said to be a *radical ideal*. Moreover, $I(V)$ is a radical ideal. The strong Nullstellensatz establishes the correspondence between radical ideals and varieties.

Theorem 4.3 *(Strong Nullstellensatz [22]) Let $\overline{\mathbb{K}}$ be an algebraically closed field, and let J be an ideal in $\overline{\mathbb{K}}[x_1, \ldots, x_d]$. Then we have $I(V_{\overline{\mathbb{K}}}(J)) = \sqrt{J}$.*

4.5 Concluding Remarks

For verification, we have to analyze constraints corresponding to the circuit functionality. Solutions to these constraints are viewed as varieties and the constraints themselves are analyzed as polynomial ideals. Since Nullstellensatz defines the correspondences between ideals and varieties, the verification problems are modeled using Nullstellensatz. These are subsequently solved using Gröbner basis techniques. While Nullstellensatz applies over algebraically closed fields, and finite fields are not algebraically closed, our approach requires modifications to suit our problems, as described in the subsequent chapters.

CHAPTER 5

IMPLEMENTATION VERIFICATION USING IDEAL MEMBERSHIP TESTING

This chapter describes our approach to the problem of formal verification of hardware implementations of arithmetic circuits over finite fields of the type $\mathbb{F}_{2^k}$, using a computer-algebra/algebraic-geometry based approach. Given a specification polynomial f and a circuit C, we have to prove that the circuit C correctly implements f. Otherwise, we have to generate a counter example that excites the bug in the design. The arithmetic circuit is modeled as a polynomial system in $\mathbb{F}_{2^k}[x_1, x_2, \cdots, x_d]$ and the verification problem is formulated using Strong Nullstellensatz over finite fields as a membership test in a corresponding (radical) ideal. This requires the computation of a Gröbner basis, which is computationally expensive. To overcome this limitation, we analyze the circuit topology and derive a term order to represent the polynomials. Subsequently, using the theory of Gröbner bases over finite fields, we prove that this term order renders the set of polynomials itself a Gröbner basis of this (radical) ideal – thus significantly enhancing verification efficiency. Using our approach, we can verify the correctness of, and detect bugs in, up to 163-bit circuits in $\mathbb{F}_{2^{163}}$, corresponding to the NIST-specified ECC standard. In contrast, contemporary approaches, including SAT, SMT, BDD, AIG based techniques, are infeasible.

5.1 Problem Statement

The following is our problem statement:

- Given a finite field $\mathbb{F}_{2^k}$, i.e. given k (datapath size), along with the corresponding irreducible polynomial $P(x)$. Let $P(\alpha) = 0$, i.e. α be the root of $P(x)$.

- Given a word-level specification polynomial $S = \mathcal{F}(A^1, A^2, \ldots, A^n) \pmod{P(x)}$, where each A^i represents a word-level k-bit input; $S, A^1, A^2, \ldots, A^n \in \mathbb{F}_{2^k}$; $\mathcal{F}$ is a function describing the input-output relation.

- Given a gate-level combinational circuit C. The bit-level primary inputs of the circuit are $\{a_0^j, a_1^j, \ldots, a_{k-1}^j\}$, for $j = 1, \ldots, n$; the primary outputs are $\{z_0, \ldots, z_{k-1}\} = Z$. Here $a_i^j, z_i \in \mathbb{F}_2, i = 0, \ldots, k-1$.

- The word-level and bit-level correspondences are the following:
  $A^1 = a_0^1 + a_1^1\alpha + \cdots + a_{k-1}^1\alpha^{k-1} = (a_{k-1}^1 \cdots a_1^1 a_0^1)$,
  $\vdots$
  $A^n = a_0^n + a_1^n\alpha + \cdots + a_{k-1}^n\alpha^{k-1} = (a_{k-1}^n \cdots a_1^n a_0^n)$,
  and the primary outputs are related as:
  $Z = z_0 + z_1\alpha + z_2\alpha^2 + \cdots + z_{k-1}\alpha^{k-1} = (z_{k-1} \cdots z_2 z_1 z_0)$.

Our goal is to formally prove that $\forall A^j, Z \in \mathbb{F}_{2^k}$, the circuit output Z correctly implements the specification $S = \mathcal{F}(A^1, A^2, \ldots, A^n) \pmod{P(x)}$ over $\mathbb{F}_{2^k}$. Otherwise, we have to produce a counter-example that excites the bug in the design.

Example 5.1 *Consider the verification problem instance for a multiplier circuit over* $\mathbb{F}_{2^k}$.

- *Given the finite field* $\mathbb{F}_{2^k}$ *and the corresponding irreducible polynomial* $P(x)$. *Let* $P(\alpha) = 0$.

- *Given a word-level multiplier specification polynomial* $S = A \cdot B \pmod{P(x)}$, *where* $A, B, S \in \mathbb{F}_{2^k}$ *(*k*-bit vectors). Function* $\mathcal{F}$ *corresponds to multiplication operation:* $A \cdot B \pmod{P}$.

- *Given a gate-level combinational circuit. The bit-level primary inputs of the circuit are* $\{a_0, \ldots, a_{k-1}, b_0, \ldots, b_{k-1}\}$, *and* $\{z_0, \ldots, z_{k-1}\}$ *are the primary outputs; here* $a_i, b_i, z_i \in \mathbb{F}_2, i = 0, \ldots, k-1$. *Therefore,* $A = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{k-1}\alpha^{k-1}$, $B = b_0 + b_1\alpha + b_2\alpha^2 + \cdots + b_{k-1}\alpha^{k-1}$ *and* $Z = z_0 + z_1\alpha + \cdots + z_{k-1}\alpha^{k-1}$.

We need to check whether the circuit implementation matches the specification, i.e., whether $S = Z, \forall a_i, b_i$.

Our approach is generic enough to verify the implementation of any combinational finite field arithmetic circuit against the given polynomial specification. Without loss of generality and for the purpose of exposition of our proposed approach, we use finite field multiplier circuits for our verification objective, as they form the core of most computations and are notoriously hard to verify.

5.2 Verification Setup and Polynomial Modeling

Our verification setup is depicted in Fig. 5.1. Given the specification polynomial $S = A \cdot B \pmod{P(x)}$, and the circuit implementation with A, B as inputs and Z as output, we want to verify the property $S = Z$ over $\mathbb{F}_{2^k}$.

Specification: Given two k-bit inputs in bit-vector form $A = (a_{k-1}a_{k-2}\cdots a_1a_0)$ and $B = (b_{k-1}b_{k-2}\cdots b_1b_0)$, the specification can be modeled in polynomial forms in $\mathbb{F}_{2^k}$ as follows:

$$\begin{aligned} A =& a_0 + a_1 \cdot \alpha + \cdots + a_{k-1} \cdot \alpha^{k-1} \\ B =& b_0 + b_1 \cdot \alpha + \cdots + b_{k-1} \cdot \alpha^{k-1} \\ S =& A \cdot B \pmod{P(x)} \end{aligned}$$

Implementation: Given a gate-level circuit netlist, we map the gate-level Boolean operators (AND, OR, NOT, XOR) to polynomials over $\mathbb{F}_2 (\subset \mathbb{F}_{2^k})$ using the following

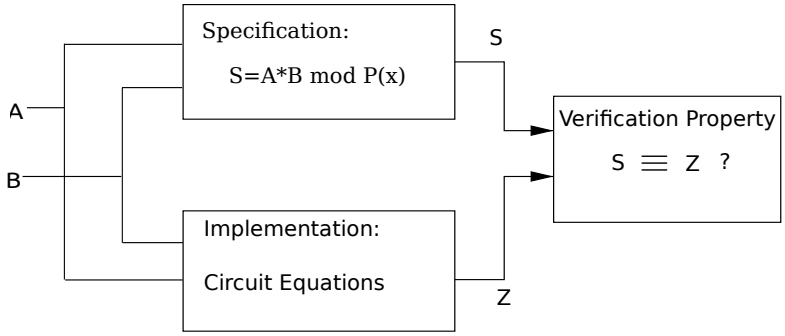

Figure 5.1. The verification setup

one-to-one mapping over $\mathbb{B} \rightarrow \mathbb{F}_2$:

$$\begin{aligned} \neg a &\rightarrow a + 1 \pmod 2 \\ a \vee b &\rightarrow a + b + a \cdot b \pmod 2 \\ a \wedge b &\rightarrow a \cdot b \pmod 2 \\ a \oplus b &\rightarrow a + b \pmod 2 \end{aligned} \tag{5.1}$$

where $a, b \in \mathbb{F}_2 = \{0, 1\}$. Note that the equation $c = \mathcal{F}(a, b)$ is written in polynomial form as $c - \mathcal{F}(a, b) = c + \mathcal{F}(a, b)$, as $-1 \equiv +1 \pmod 2$.

Example 5.2 *Consider the equation with Boolean operators:*

$$z = a \oplus (b \vee c).$$

The equation modeled over $\mathbb{F}_2$ is:

$$z + a + b + c + b \cdot c = 0$$

The left-hand side expression is a polynomial in $\mathbb{F}_2[a, b, c, z] \subset \mathbb{F}_{2^k}[a, b, c, z]$:

$$z + a + b + c + b \cdot c$$

Therefore, we can transform the entire circuit implementation as polynomials over $\mathbb{F}_{2^k}$. Let Z symbolically denote the word-level result of the implementation, i.e. the output of the circuit.

The Verification Property: The property $S = Z$ is modeled as a polynomial f : $S + Z = 0$ over $\mathbb{F}_{2^k}$.

Overall, our verification constraints can be modeled as a polynomial system as follows:

$$
\left.\begin{array}{r}
f_1(x_1, x_2, \cdots, x_d) = 0 \\
f_2(x_1, x_2, \cdots, x_d) = 0 \\
\vdots \\
f_Z : Z + z_0 + z_1 \cdot \alpha, \cdots, z_{k-1} \cdot \alpha^{k-1} = 0
\end{array}\right\} \quad \textit{Circuit implementation}
$$

$$
\left.\begin{array}{r}
f_A : A + a_0 + a_1 \cdot \alpha + \cdots + a_{k-1} \cdot \alpha^{k-1} = 0 \\
f_B : B + b_0 + b_1 \cdot \alpha + \cdots + b_{k-1} \cdot \alpha^{k-1} = 0 \\
f_{spec} : S + A \cdot B = 0
\end{array}\right\} \quad \textit{Word-level specification}
$$

$$
\left.\begin{array}{r}
f : S + Z = 0
\end{array}\right\} \quad \text{Property: } S = Z \text{ ?}
$$

Example 5.3 *Consider a 2-bit multiplier over* $\mathbb{F}_{2^2}$ *with* $P(x) = x^2 + x + 1$*, given in Fig. 5.2. Variables* a_0, a_1, b_0, b_1 *are primary inputs,* z_0, z_1 *are primary outputs, and* c_0, c_1, c_2, c_3, r_0 *are intermediate variables. The gate* $\otimes$ *corresponds to AND-gate, i.e. bit-level multiplication modulo 2. The gate* $\oplus$ *corresponds to XOR-gate, i.e. addition modulo 2.*

The circuit can be described using the following Boolean equations:

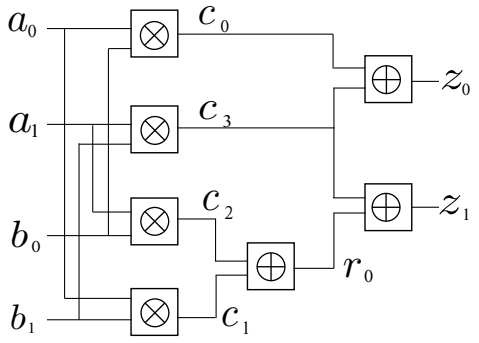

Figure 5.2. A 2-bit multiplier over $\mathbb{F}(2^2)$.

$$c_0 = a_0 \wedge b_0,$$
$$c_1 = a_0 \wedge b_1,$$
$$c_2 = a_1 \wedge b_0,$$
$$c_3 = a_1 \wedge b_1,$$
$$r_0 = c_1 \oplus c_2,$$
$$z_0 = c_0 \oplus c_3,$$
$$z_1 = r_0 \oplus c_3,$$

With the mapping rules given in Equation 5.1, the above equations are transformed into the following polynomials:

$$c_0 + a_0 \cdot b_0,$$
$$c_1 + a_0 \cdot b_1,$$
$$c_2 + a_1 \cdot b_0,$$
$$c_3 + a_1 \cdot b_1,$$
$$r_0 + c_1 + c_2,$$
$$z_0 + c_0 + c_3,$$
$$z_1 + r_0 + c_3,$$

Therefore, our overall polynomial system is:

$$
\left.\begin{array}{r}
f_1 : c_0 + a_0 \cdot b_0 \\
f_2 : c_1 + a_0 \cdot b_1 \\
f_3 : c_2 + a_1 \cdot b_0 \\
f_4 : c_3 + a_1 \cdot b_1 \\
f_5 : r_0 + c_1 + c_2 \\
f_6 : z_0 + c_0 + c_3 \\
f_7 : z_1 + r_0 + c_3 \\
f_Z : Z + z_0 + z_1 \cdot \alpha
\end{array}\right\} \quad \textit{Circuit constraints}
$$

$$
\left.\begin{array}{r}
f_A : A + a_0 + a_1 \cdot \alpha \\
f_B : B + b_0 + b_1 \cdot \alpha \\
f_{spec} : S + A \cdot B
\end{array}\right\} \quad \textit{specification}
$$

$$
\left.\begin{array}{r}
f : S + Z
\end{array}\right\} \quad \textit{Property to verify: } S = Z\ ?
$$

With the polynomial model given above, we formulate our problem as **(radical) ideal membership testing**, which is described next.

5.3 Verification Formulation as Ideal Membership Testing

To formulate our verification test, we first analyze the circuit and model the Boolean gate-level operators as polynomials over $\mathbb{F}_2$ $(\subset \mathbb{F}_{2^k})$, as given by the mappings of Equations 5.1. To this set we then append the polynomials corresponding to the word-level specification. Let $\{f_1, f_2, \ldots, f_s\}$ denote this set of polynomials derived from both *specification* and *implementation*. Let $\{x_1, x_2, \ldots, x_d\}$ denote all the variables in the polynomial system. As a consequence, $\{f_1, f_2, \ldots, f_s\} \in \mathbb{F}_{2^k}[x_1, \ldots, x_d]$. Let $J = \langle f_1, \ldots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ denote the ideal generated by these polynomials. Our verification property $S = Z$ is also modeled as a polynomial $f : S+Z \in \mathbb{F}_{2^k}[x_1, \ldots, x_d]$.

To prove that the specification polynomial (f) matches the implementation ($J = \langle f_1, \ldots, f_s \rangle$), we need to check whether $f : S + Z = 0$ *agrees* with all the solutions of J over the field $\mathbb{F}_{2^k}$. In computer algebra terminology, we need to check *whether or not f vanishes on the variety* $V_{\mathbb{F}_{2^k}}(J)$, where $V_{\mathbb{F}_{2^k}}(J)$ denotes the variety of ideal J over the given field $\mathbb{F}_{2^k}$. This is because for all points (solutions) $p \in V_{\mathbb{F}_{2^k}}(J)$, if $f(p) = 0$, then

$f : S + Z = 0 \implies S = Z$. On the other hand, if $f(p) \neq 0$ for some point p, then p corresponds to the bug in the design.

Now if f vanishes on $V_{\mathbb{F}_{2^k}}(J)$, according to Proposition 4.3, we know that f should be a member of the radical ideal $I(V_{\mathbb{F}_{2^k}}(J))$. Therefore, our verification test can be modeled as membership testing of f in the (radical) ideal $I(V_{\mathbb{F}_{2^k}}(J))$. To solve this problem, we need to first derive the generators of $I(V_{\mathbb{F}_{2^k}}(J))$ (note that we are only given the generators of J), and then perform the ideal membership testing using the Gröbner basis algorithm.

5.3.1 Generating $I(V_{\mathbb{F}_{2^k}}(J))$

Strong Nullstellensatz establishes correspondences between ideals and their radicals. As given in Theorem 4.3, $I(V_{\overline{\mathbb{K}}}(J)) = \sqrt{J}$, where the variety V is taken over the algebraically closed field $\overline{\mathbb{K}}$. Finite fields are, however, *not* algebraically closed, as shown by the following result from [50]:

Theorem 5.1 *Given finite fields $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^m}$ such that n divides m. Then $\mathbb{F}_{2^n} \subset \mathbb{F}_{2^m}$.*

Therefore, $\mathbb{F}_2 \subset \mathbb{F}_{2^2} \subset \mathbb{F}_{2^4} \subset \mathbb{F}_{2^8} \subset \ldots$; and $\mathbb{F}_2 \subset \mathbb{F}_{2^3} \subset \mathbb{F}_{2^6} \ldots$; and so on. The algebraic closure of $\mathbb{F}_{2^k}$ is known to be an infinite field obtained as the union of all such finite fields.

Therefore, Nullstellensatz needs to be suitably modified for application over finite fields. We re-visit the notion of vanishing polynomials for this purpose.

Over the finite field $\mathbb{F}_{2^k}$, any element A satisfies the property $A^{2^k} - A = 0$. Therefore, polynomial $x^{2^k} - x$ vanishes at all points in $\mathbb{F}_{2^k}$, and $x^{2^k} - x$ is called the vanishing polynomial of the field. As a consequence, the variety $V(x^{2^k} - x) = \mathbb{F}_{2^k}$. Over multivariate polynomial ring $\mathbb{F}_{2^k}[x_1, \ldots, x_d]$, $V(x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d)$ is $\mathbb{F}_{2^k}^d$.

In the sequel, we use the following notation: Let $J_0 = \langle x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$ denote the ideal of vanishing polynomials over $\mathbb{F}_{2^k}$. Also, if $J = \langle f_1, \ldots, f_s \rangle$ then the sum of ideals $J + J_0 = \langle f_1, \ldots, f_s,\ x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$. Let $\overline{\mathbb{F}_{2^k}}$ denote the algebraic closure of $\mathbb{F}_{2^k}$.

Lemma 5.1 *Let $J \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ be any ideal and let $J_0 = \langle x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$. Then $V_{\mathbb{F}_{2^k}}(J) = V_{\overline{\mathbb{F}_{2^k}}}(J + J_0)$.*

Proof. Since $\overline{\mathbb{F}_{2^k}} \supset \mathbb{F}_{2^k}$, we have :

$$\begin{aligned} V_{\mathbb{F}_{2^k}}(J) &= V_{\overline{\mathbb{F}_{2^k}}}(J) \cap \mathbb{F}_{2^k}^d \\ &= V_{\overline{\mathbb{F}_{2^k}}}(J) \cap V_{\mathbb{F}_{2^k}}(J_0) \\ &= V_{\overline{\mathbb{F}_{2^k}}}(J) \cap V_{\overline{\mathbb{F}_{2^k}}}(J_0) \\ &= V_{\overline{\mathbb{F}_{2^k}}}(J + J_0) \end{aligned}$$

■

As a consequence of the above lemma, variety of any ideal J over a finite field $\mathbb{F}_{2^k}$ can be equivalently analyzed over its algebraic closure $\overline{\mathbb{F}_{2^k}}$ by just appending to J all the vanishing polynomials J_0. These vanishing polynomials do not change the zero-set of J but allow the same analysis over the algebraic closure.

Lemma 5.2 $I(V_{\mathbb{F}_{2^k}}(J)) = I(V_{\overline{\mathbb{F}_{2^k}}}(J + J_0)) = \sqrt{J + J_0}$.

Proof. As shown above, $V_{\mathbb{F}_{2^k}}(J) = V_{\overline{\mathbb{F}_{2^k}}}(J + J_0)$. Therefore, $I(V_{\mathbb{F}_{2^k}}(J)) = I(V_{\overline{\mathbb{F}_{2^k}}}(J + J_0))$. According to Strong Nullstellensatz, $I(V_{\overline{\mathbb{F}_{2^k}}}(J + J_0)) = \sqrt{J + J_0}$. Thus:

$$I(V_{\mathbb{F}_{2^k}}(J)) = I(V_{\overline{\mathbb{F}_{2^k}}}(J + J_0)) = \sqrt{J + J_0} \tag{5.2}$$

■

Lemma 5.3 *Let J be any arbitrary polynomial ideal in $\mathbb{F}_{2^k}[x_1, \ldots, x_d]$ and J_0 be the corresponding vanishing ideal. Then $J + J_0$ is radical. In other words, $\sqrt{J + J_0} = J + J_0$.*

Proof. This is a well known result, a proof of which is given in [66]. ■

Putting together the above results, we finally arrive at the following application of Nullstellensatz over finite fields.

Theorem 5.2 [**Strong Nullstellensatz in Finite Fields**] *Let* $J \subset \mathbb{F}_{2^k}[x_1, x_2, \cdots, x_n]$ *be an ideal and* J_0 *be the ideal of vanishing polynomials. Then,*

$$I(V_{\mathbb{F}_{2^k}}(J)) = J + J_0 = J + \langle x_1^{2^k} - x_1, x_2^{2^k} - x_2, \cdots, x_d^{2^k} - x_d \rangle \tag{5.3}$$

Proof. Combining Lemma 5.2 and Lemma 5.3,

$$I(V_{\mathbb{F}_{2^k}}(J)) = I(V_{\overline{\mathbb{F}_{2^k}}}(J + J_0)) = \sqrt{J + J_0} = J + J_0 \tag{5.4}$$

where $J_0 = \langle x_1^{2^k} - x_1, x_2^{2^k} - x_2, \cdots, x_d^{2^k} - x_d \rangle$. ■

Overall Verification Problem Formulation: Through Strong Nullstellensatz over finite fields, given an ideal J, we can directly construct ideal $I(V_{\mathbb{F}_{2^k}}(J)) = J + J_0$. For our verification problem, we take the polynomials $\{f_1, \ldots, f_s\}$ representing the circuit constraints and the specification polynomials to generate ideal J. Then we append the vanishing polynomials $\{x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ of ideal J_0. Our verification problem can now be formulated as testing whether the verification property polynomial f is in $J + J_0$. If $f \in (J + J_0)$, correctness of the circuit is established. Otherwise, there is a bug in the design. To test if $f \in (J + J_0)$, it is required to compute a Gröbner basis G of the ideal $J + J_0$. Then, we reduce f w.r.t. G: i.e., $f \xrightarrow{G}_+ r$. If $r = 0$, then the circuit is correct, otherwise there is a bug in the design.

Example 5.4 *Let us re-consider Example 5.3. First, polynomials are extracted from the circuit implementation and the specification, as shown in Example 5.3. These polynomials represent the ideal* J*. Along with the ideal* $J_0 = \langle x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$*, the following polynomials represent* $J + J_0$ *for the multiplier circuit.*

$$
\left.\begin{aligned}
f_1 &: c_0 + a_0 \cdot b_0 \\
f_2 &: c_1 + a_0 \cdot b_1 \\
f_3 &: c_2 + a_1 \cdot b_0 \\
f_4 &: c_3 + a_1 \cdot b_1 \\
f_5 &: r_0 + c_1 + c_2 \\
f_6 &: z_0 + c_0 + c_3 \\
f_7 &: z_1 + r_0 + c_3 \\
f_Z &: Z + z_0 + z_1 \cdot \alpha
\end{aligned}\right\} \quad implementation\ (\subset J)
$$

$$
\left.\begin{aligned}
f_A &: A + a_0 + a_1 \cdot \alpha \\
f_B &: B + b_0 + b_1 \cdot \alpha \\
f_{spec} &: S + A \cdot B = 0
\end{aligned}\right\} \quad specification\ (\subset J)
$$

$$
\left.\begin{aligned}
a_0^2 - a_0,\ a_1^2 - a_1,\ b_0^2 - b_0,\ b_1^2 - b_1 \\
c_0^2 - c_0,\ c_1^2 - c_1,\ c_2^2 - c_2,\ c_3^2 - c_3 \\
r_0^2 - r_0,\ z_0^2 - z_0,\ z_1^2 - z_1 \\
A^4 - A,\ B^4 - B,\ Z^4 - Z,\ S^4 - S
\end{aligned}\right\} \quad \textit{vanishing polynomials}(J_0)
$$

Now we need to compute the Gröbner basis G of this ideal $J + J_0$. Once the computation of G is completed, we simply need a polynomial reduction to test whether $f : S + Z$ can be reduced by G. In other words, we need test whether $S + Z \xrightarrow{G}_+ 0$?

While our approach seems reasonably simple, the complexity of Gröbner basis computation can make verification infeasible.

Complexity of Gröbner basis over finite fields: For our specific problem of computing a Gröbner basis for $J + J_0$ over $\mathbb{F}_q$, the following result is known [66]:

Theorem 5.3 *Let $I = \langle f_1, \ldots, f_s,\ x_1^q - x_1, \ldots, x_d^q - x_d \rangle \subset \mathbb{F}_q[x_1, \ldots, x_d]$ be an ideal over any finite field $\mathbb{F}_q$. The time and space complexity of Buchberger's algorithm to compute a Gröbner basis of I is bounded by $q^{O(d)}$ assuming that the length of input $f_1, \ldots, f_s$ is dominated by $q^{O(d)}$.*

In our case $q = 2^k$, and when k and d are large, this complexity makes verification infeasible. In what follows, we show that a variable/term order can be derived by analyzing the circuit topology which makes the set of polynomials $\{f_1, \ldots, f_s, x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ itself a Gröbner basis of $J + J_0$ – obviating the need to apply Buchberger's algorithm.

5.4 Obviating Buchberger's Algorithm

Just as variable orderings play a critical role in constructing BDDs and solving SAT feasibly, the Gröbner basis computation is also highly susceptible to the term orderings imposed on the polynomials. Therefore a key step to improve/avoid the high complexity of Gröbner basis computation is to derive a "good" term order.

Buchberger's work [21] initially laid the foundation for computing Gröbner's bases. Subsequently, many improvements were introduced to improve the efficiency of Buchberger's algorithm. Two of the most important improvements are the chain and product criteria. For our particular circuit verification application, we exploit the product criteria.

Lemma 5.4 *[Product Criterion [67]] Let $\mathbb{F}$ be any field, and $f, g \in \mathbb{F}[x_1, \cdots, x_d]$ be polynomials. If the equality $lm(f) \cdot lm(g) = LCM(lm(f), lm(g))$ holds, then $Spoly(f, g) \xrightarrow{G}_+ 0$.*

The above result states that when the leading monomials of f, g are relatively prime, then $Spoly(f, g)$ always reduces to 0 modulo G. Thus $Spoly(f, g)$ need not be considered in Buchberger's algorithm. Modern computer algebra engines perform this check to avoid unnecessary $Spoly(f, g)$ computations. If we could analyze the given circuit and derive a term order such that every polynomial pair (f, g) in the generating set has relatively prime leading monomials, then for all S-polynomials, the subsequent reduction would not add any new polynomials in the basis. In other words, $Spoly(f, g) \xrightarrow{G}_+ 0$ for all pairs f, g. Consequently, the polynomials $\{f_1, \ldots, f_s\}$ extracted from the circuit (corresponding ideal J) and represented using such a term order would itself constitute a Gröbner basis of J. In [16], the authors derive exactly such a term order, and a similar concept can be applied in our case.

Note that in our case:

- since the circuit constraints $\{f_1, \ldots, f_s\}$ are modeled as polynomials in $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$, they contain only multi-linear monomial terms;
- the output of a gate is uniquely computed, and it always appears as a "single variable term" in the polynomials;
- the circuit is acyclic;

Let x_i be the output variable of any gate H_i in the circuit, and let $x_{p_1}, \ldots, x_{p_j}$ denote variables that are the inputs to the gate H_i. If we can represent the polynomials f_i such that $x_i >$ every monomial in the variables $x_{p_1}, \ldots, x_{p_j}$, then all $(f_i, f_j), i \neq j$ have relatively prime leading monomials and $\{f_1, \ldots, f_s\}$ is a Gröbner basis.

Proposition 5.1 *Let C be any arbitrary combinational circuit. Let $\{x_1, \ldots, x_d\}$ denote the set of all variables (signals) in the circuit, i.e. the primary input, intermediate and primary output variables. Perform a* **reverse topological traversal** *of the circuit and order the variables such that $x_i > x_j$ if x_i appears earlier in the reverse topological order. Impose a lex term order to represent the Boolean expression for each gate as a polynomial f_i; then $f_i = x_i + tail(f_i)$. Then the set of all polynomials $\{f_1, \ldots, f_s\}$ forms a Gröbner basis, as $lt(f_i)$ and $lt(f_j)$ for $i \neq j$ are relatively prime.*

Example 5.5 *Consider the circuit of Figure 5.2, reproduced below. Variables a_0, a_1, b_0, b_1 are primary inputs, z_0, z_1 are primary outputs, and c_0, c_1, c_2, c_3, r_0 are intermediate variables.*

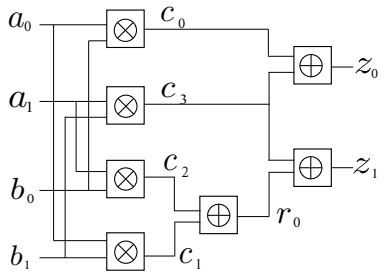

Figure 5.3. A 2-bit multiplier over $\mathbb{F}(2^2)$. The gate $\otimes$ corresponds to AND-gate, i.e. bit-level multiplication modulo 2. The gate $\oplus$ corresponds to XOR-gate, i.e. addition modulo 2.

We perform a reverse topological traversal of the circuit. Starting from the primary outputs, traverse the circuit to the primary inputs, and order the gates according to the their (reverse) topological levels. The primary outputs z_0, z_1 are both at level-0, variables r_0, c_0, c_3 are at level-1, c_1, c_2 are at level-2, and the primary inputs a_0, a_1, b_0, b_1 are at level-3. We order the variables $\{z_0 > z_1\} > \{r_0 > c_0 > c_3\} > \{c_1 > c_2\} > \{a_0 > a_1 > b_0 > b_1\}$. Using this variable order, we impose a lex term order on the monomials. Then the polynomials of J all have relatively prime leading terms, as shown below:

$$c_0 + a_0 \cdot b_0,\ lm = c_0;$$
$$c_1 + a_0 \cdot b_1,\ lm = c_1;$$
$$c_2 + a_1 \cdot b_0,\ lm = c_2;$$
$$c_3 + a_1 \cdot b_1,\ lm = c_3;$$
$$r_0 + c_1 + s_2,\ lm = r_0;$$
$$z_0 + c_0 \cdot c_3,\ lm = z_0;$$
$$z_1 + r_0 \cdot c_3,\ lm = z_1$$

In our overall problem formulation, we also have variables $A, B, S, Z \in \mathbb{F}_{2^k}$. They can also be accommodated in this term order by imposing $S > Z > A > B > z_0 > z_1 > r_0 > c_0 > c_3 > c_1 > c_2 > a_0 > a_1 > b_0 > b_1$.

Thus, using the result of Proposition 5.1, the set of polynomials $\{f_1, \ldots, f_s\}$ is a Gröbner basis for J. Note that $\{x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ is a Gröbner basis for J_0. However, we have to compute a Gröbner basis of $J + J_0 = \langle f_1, \ldots, f_s,\ x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$. Not all polynomials pairs in $\{f_1, \ldots, f_s,\ x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ have relatively prime leading monomials.

Consider an arbitrary polynomial $f_i \in J$. Using our term order, we have $f_i = x_i + \text{tail}(f_i)$; i.e. the leading monomial of f_i is a single variable term x_i. Clearly, the pair $(x_i + \text{tail}(f_i),\ x_i^{2^k} - x_i),\ f_i \in J,\ x_i^{2^k} - x_i \in J_0$ do not have relatively prime leading monomials. In fact, the pairs $(x_i + \text{tail}(f_i), x_i^{2^k} - x_i)$ are the only ones to be considered for Gröbner basis computation, as all other pairs have relatively prime leading terms.

This motivated us to investigate further the question "what is the result of the reduction $\text{Spoly}(x_i + \text{tail}(f_i), x_i^{2^k} - x_i) \xrightarrow{J,J_0}_+ r$?". We state and prove the following:

Theorem 5.4 *Let $q = 2^k$, and let $\mathbb{F}_q[x_1, \ldots, x_d]$ be a ring on which we have a monomial order $>$. Let I be a subset of $\{1, \ldots, d\}$. For all $i \in I$, let $f_i = x_i + P_i$ (where $P_i = tail(f_i)$) such that all indeterminates x_j that appear in P_i satisfy $x_i > x_j$. Then the set $G = \{f_i : i \in I\} \cup \{x_1^q - x_1, \ldots, x_d^q - x_d\}$ is a Gröbner basis.*

Proof. According to Buchberger's Theorem (Theorem 1.7.4 in [22]), we need to show that for all $f, g \in G$, $Spoly(f, g) \xrightarrow{G}_+ 0$. Let $G_1 = \{f_i : i \in I\}$. Lemma 5.4 shows that if $f, g \in G$, have relatively prime leading terms, then $Spoly(f, g) \xrightarrow{G}_+ 0$. So the only case where Lemma 5.4 does not apply is when $f = x_i + P_i$ and $g = x_i^q - x_i$. Then $Spoly(f, g) = x_i^{q-1} f - g = P_i x_i^{q-1} + x_i$. In what follows, it is important to note that the indeterminates appearing in P_i are all less than x_i.

First of all, $P_i x_i^{q-1} + x_i - P_i x_i^{q-2}(x_i + P_i) = P_i^2 x_i^{q-2} + x_i$, which shows that $P_i x_i^{q-1} + x_i \xrightarrow{x_i+P_i} P_i^2 x_i^{q-2} + x_i$.

Next, $P_i^2 x_i^{q-2} + x_i - P_i^2 x_i^{q-3}(x_i + P_i) = P_i^3 x_i^{q-3} + x_i$. Continuing in this fashion, we get $P_i^{q-1} x_i + x_i - P_i^{q-1}(x_i + P_i) = x_i + P_i^q$, and finally $x_i + P_i^q - (x_i + P_i) = P_i^q - P_i$. Hence,

$$P_i x_i^{q-1} + x_i \xrightarrow{x_i+P_i} P_i^2 x_i^{q-2} + x_i \xrightarrow{x_i+P_i} P_i^3 x_i^{q-3} + x_i \xrightarrow{x_i+P_i} \cdots$$

$$\cdots \xrightarrow{x_i+P_i} P_i^q + x_i \xrightarrow{x_i+P_i} P_i^q - P_i.$$

Over the finite field $\mathbb{F}_q$, $P_i^q - P_i$ is a vanishing polynomial. Therefore, $P_i^q - P_i \in I(V(J_0)) = \langle x_1^q - x_1, \ldots, x_d^q - x_d \rangle$. By Lemma 5.4, $G_0 = \{x_1^q - x_1, \ldots, x_d^q - x_d\}$ is Gröbner basis. Therefore $P_i^q - P_i \xrightarrow{G_0}_+ 0$ which gives that $P_i^q - P_i \xrightarrow{G}_+ 0$, as $G_0 \subset G$.

In conclusion, $\forall f, g \in G,\ Spoly(f, g) \xrightarrow{G}_+ 0$ and hence G is a Gröbner basis. ■

As a consequence of Theorem 5.4, the Gröbner basis G for our verification instance (ideal $J + J_0$) can be obtained directly by construction using a reverse topological traversal of the circuit. While G is indeed a Gröbner basis, it is neither *minimal* nor *reduced*. We now show that this basis can actually be made *minimal* by considering the vanishing ideal of only the primary inputs of the given circuit.

Corollary 5.1 *Let $q = 2^k$ and $\mathbb{F}_q[x_1, \ldots, x_d]$ be the ring on which we impose the monomial order $>$ obtained via Proposition 5.1. Let I be a subset of $\{1, \ldots, d\}$. For all $i \in I$, let $f_i = x_i + P_i$ (where $P_i = tail(f_i)$) such that all indeterminates x_j that appear in P_i satisfy $x_i > x_j$. Let X_{PI} denote the set of all primary input variables of the circuit. Then the set $G = \{f_i : i \in I\} \cup \{x_{pi}^2 - x_{pi}\}$ is a* **minimal** *Gröbner basis, where $x_{pi} \in X_{PI}$.*

Proof. According to the Definition 4.16 of a minimal Gröbner basis, two conditions have to be satisfied: i) all polynomials in the basis are monic, i.e their leading coefficient is 1; and ii) leading monomial of any polynomial does not divide the leading monomial of any other polynomial in the basis. We have already shown that G is a Gröbner basis. Moreover, in $\mathbb{F}_{2^k}$, the coefficient of every non-zero term is always 1. Therefore, all polynomials are monic.

Furthermore, our ideal basis G consists of two sets of polynomials: i) polynomials derived from the circuit which are of the form $f_i = x_i + \text{tail}(f_i)$; and ii) the vanishing polynomials $x_i^{2^k} - x_i$ for $i = 1, \ldots, d$. Our term order ensures that in $f_i = x_i + \text{tail}(f_i)$, x_i corresponds to either the primary output variables or the intermediate variables. Primary input variables ($x_i \in X_{PI}$) will never occur as leading terms of f_i because a primary input is not an output of any gate in the circuit. Therefore, $\forall x_i \in (\{x_1, \ldots, x_d\} - \{X_{PI}\})$, there always exists f_i with $lm(f_i) = x_i$ which will divide the vanishing polynomial $x_i^{2^k} - x_i$. In such cases, $x_i^{2^k} - x_i,\ x_i \notin X_{PI}$ can be removed from the basis. By eliminating all vanishing polynomials corresponding to non-primary-input variables, we will obtain $G = \{f_i : i \in I\} \cup \{x_{pi}^{2^k} - x_{pi}\}$ as a minimal Gröbner basis, where $x_{pi} \in X_{PI}$.

Finally, since $x_{pi} \in \mathbb{F}_2 \subset \mathbb{F}_{2^k}$, $x_i^2 - x_i = 0$, we obtain $G = \{f_i\} \cup \{x_{pi}^2 - x_{pi}\}$ as the minimal Gröbner basis. ■

While we can obtain a minimal Gröbner basis G directly by construction, unfortunately, we *cannot* obtain a *reduced* Gröbner basis without actually performing the reduction. This is because in a reduced Gröbner basis, the tail (tail(f_i)) of every polynomial f_i is also reduced w.r.t. $lt(f_j)$, for all $i \neq j$. However, a reduced Gröbner basis computation is not necessary for ideal membership testing.

5.5 Our Overall Approach

We setup the verification problem in $\mathbb{F}_{2^k}[x_1, \ldots, x_d]$, on which we impose the monomial order $>$ as derived above. We extract the set of polynomials $G_1 = \{f_1, \ldots, f_s\}$ from the circuit. We generate the set $G_0 = \{x_{pi}^{2^k} - x_{pi}\} \forall x_{pi} \in X_{PI}$. Then the set $G = G_1 \cup G_0$ is forms a minimal Gröbner basis of the ideal $J + J_0 = \langle f_1, \ldots, f_s, x_{pi}^{2^k} - x_{pi} \rangle$. We take our specification polynomial f and compute $f \xrightarrow{G}_+ r$. If $r = 0$, then $f \in J + J_0$ and the circuit is correct; otherwise, if $r \neq 0$, then we have a bug in the design. Moreover, *if $r \neq 0$, then the monomial order ensures that r contains only the primary input variables*. To show this, assume that $r \neq 0$ and r contains either an intermediate or a primary output variable x_j. As there always exists a polynomial f_j in G with $lm(f_j) = x_j$, r can be further reduced by f_j. Continuing in this fashion, all the terms with non-primary-input (intermediate or primary output) variables can be eliminated. Finally, *in the presence of a bug, any assignment to the (primary-input) variables that makes $r \neq 0$, provides a counter-example for debugging*. A SAT or SMT-solver can find such an assignment in no time as r is simplified by Gröbner basis reduction. Our results therefore obviate the need to construct a Gröbner basis, and the verification can be performed only by reduction: $f \xrightarrow{G}_+ r$.

Our overall approach is described in Algorithm 5. It first inputs the given circuit implementation as Boolean equations. Each equation then is transformed to polynomials G_1 using Equations 5.1. All polynomials are then normalized into a sum-of-term form using the distributive law: $A \cdot (B + C) = A * B + A * C$. Subsequently, our verification problem is formulated as a radical ideal membership testing. We conduct a reverse topology traversal of the circuit to generate the variable ordering. Then, we append vanishing polynomials $G_0 = \{x^2 + x\}$ for all $x \in$ primary inputs. Finally, we compute the reduction of f (property polynomial) modulo $G_1 \cup G_0$. If the reduction result is $r = 0$, the circuit is correct. If there are bugs in implementation, then the result r is a polynomial that encodes *all* input vector assignments that excite the bug(s) in the design.

Algorithm 5: Proposed Verification Algorithm

Input: Circuit Implementation Equations Z.
Specification Polynomial S.
Output: True if $S = Z$. Bug polynomial r if $S \neq Z$.
for *(i=0; i < number of eqns ; i++)* **do**
`/*Each equation is transformed to polynomials */;`
poly[i] = Eqn-to-Poly(eqn[i]);
`/*Each equation is transformed to sum-of-term form */;`
newpoly[i] = Sum-of-term(poly[i]);
end
`/*Obtain circuit-based variable order*/;`
ordered_var=T_Traversal(newpoly);
for *var* $\in \{PI\}$ **do**
`/*appending vanishing polynomials*/;`
vanpoly[i]=$x^2 + x$;
end
r=reduce(S,Z,vanpoly,ordered_var);
if *r*=$\{0\}$ **then**
return True;
else
return Bug polynomial r;
end

5.6 Experimental Results

Our algorithm is implemented in $C++$ with calls to the SINGULAR computer algebra tool [v. 3-1-2] [68] to perform polynomial reductions. Our experiments are conducted on a desktop with 2.40 GHz Intel Core(TM)2 Quad CPU and 8 GB memory running 64-bit Linux.

We conducted verification experiments on several large custom-designed circuits, including Mastrovito multipliers, Montgomery multipliers, Barrett multipliers and ECC point addition and point doubling circuits. The designs are given in equation (EQN) format and then translated to different formats: CNF, SMTLIB, BLIF, Polynomials that are used by SAT, SMT, BDD/AIG based solvers, and Singular, respectively. All our circuit benchmarks have been made available to the larger verification community through the SMT-LIB benchmark suite [69].

5.6.1 Evaluation of SAT, SMT, BDD, AIG Based Methods

We evaluated the performance of many SAT solvers [70] [71] [72] [73], SMT solvers [38] [74] [75] [39] [40] [76] [77] [8] and BDD based techniques [78], on our benchmarks. For these experiments, using the conventional equivalence checking approach, we created a "miter" circuit to compare the specification against the implementation. The implementation was given as a Montgomery multiplier as a gate-level netlist. Since BDD/SAT/AIG based approaches cannot operate upon word-level representations directly, the specification is given as a Mastrovito-style gate-level circuit implementation. For SMT experiments, the designs were modeled at bit-vector level using quantifier-free bit-vector (QF-BV) theories, maintaining a bit-vector-level abstraction whenever possible. Table 5.1 shows that none of BDDs, AIG/ABC, SAT or SMT solvers can verify the correctness of circuits beyond 16-bits.

Table 5.1. Runtime for verification of Montgomery versus Mastrovito multipliers over $\mathbb{F}_{2^k}$ for BDDs, SAT, SMT-solver and AIG/ABC based methods. TO = timeout of 10hrs. Time is given in seconds.

	Word size of the operands k-bits		
Solver	8	12	16
MiniSAT	22.55	TO	TO
CryptoMiniSAT	7.17	16082.40	TO
PrecoSAT	7.94	TO	TO
PicoSAT	14.85	TO	TO
Yices	10.48	TO	TO
Beaver	6.31	TO	TO
CVC	TO	TO	TO
Z3	85.46	TO	TO
Boolector	5.03	TO	TO
Sonolar	46.73	TO	TO
SimplifyingSTP	14.66	TO	TO
ABC	242.78	TO	TO
BDD	0.10	14.14	1899.69

5.6.2 Evaluation of Our Approach

Our approach takes as inputs a gate-level circuit implementation and word-level specification. Note the difference in the input requirements between our approach and SAT/BDD/SMT/AIG based approaches. Our approach only requires a word-level specification while SAT/BDD/SMT/AIG based approaches require an inherently large gate-level specification. Therefore, there is an inherent advantage of our method in that it maintains a high-level abstraction whenever possible.

Verification using Gröbner Basis Computations in SINGULAR**:** Conceptually, our approach requires to first compute a Gröbner basis and then conduct a polynomial reduction (ideal membership testing). If we use SINGULAR to compute a Gröbner basis using our term order derived from Proposition 5.1, but without deducing the results of Theorem 5.4 and Corollary 5.1, we can verify the correctness of only up to 48-bit multipliers. Beyond that, the Gröbner basis engine runs into memory explosion. This result is shown in Table 5.2.

Evaluation of Our Approach: Our approach only requires a polynomial reduction (division) for the verification test: $S + Z \xrightarrow{G_1, G_0}_{+} r$ and to check if $r = 0$? For this polynomial reduction, we use the REDUCE command in SINGULAR. Results for verification of Mastrovito multipliers using our term ordering and only this reduction are shown in Table 5.3. With our approach, we can verify the correctness of up to 163-bit Mastrovito multipliers. We also experimented with bug-catching in incorrect designs; the bugs were introduced by arbitrarily swapping the wires (variables) x_i with x_j, for some gate $i \neq j$. In such cases, we obtained a non-zero r. We used a SAT-solver to find a SAT assignment

Table 5.2. Verification of Mastrovito multipliers by computing Gröbner bases using SINGULAR. MO=out of 8G memory. Time is given in seconds.

Size	16	32	48	64	96	128	160	163
#variables	323	1155	2499	4355	9603	16899	26243	27224
#polynomials	609	2241	4897	8577	19009	33537	52161	54117
#terms	2415	9439	21071	37311	83615	148351	231519	240261
Time	0.94	93.80	1174.27	MO	MO	MO	MO	MO

to $r \neq 0$. These run times are shown in Table 5.3.

Table 5.3. Runtime for verifying bug-free and buggy Mastrovito multipliers using our approach. TO = timeout of 10hrs. Time is given in seconds.

method	16	32	48	64	96	128	160	163
#variables	323	1155	2499	4355	9603	16899	26243	27224
#polynomials	291	1091	2403	4227	9411	16643	25923	26989
#terms	1793	7169	16129	28673	64513	114689	179201	185984
Bug-free	0.04	1.41	24.00	112.13	758.82	3054	9361	16170
Bugs	0.04	1.43	25.11	114.86	788.65	3061	9384	16368

Results of the verification of Montgomery multipliers are shown in Table 5.4. Montgomery multipliers are significantly larger than Mastrovito multipliers. If we represent a polynomial for every gate in the design, then we create too many variables (d) in the system, exceeding SINGULAR'S capacity ($d \leq 32767$). For this reason, we partition the circuit, and construct the polynomials for each circuit partition – and we ensure that our term ordering constraint is not violated. With such efforts, we are able to verify Montgomery multipliers up to 128-bit datapaths, beyond which we still exceed SINGULAR'S capacity. Similarly, results for the verification of Barrett multipliers are shown in Table 5.5.

Table 5.6 and Table 5.7 show the results of verifying ECC point addition and point doubling circuits, respectively. There are several representation systems for ECC point addition and point doubling. We choose L*ó*pez-Dahab coordinate system [79] to repre-

Table 5.4. Runtime for verifying bug-free and buggy Montgomery multipliers using our approach. TO = timeout of 10hrs. Time is given in seconds.

method	16	32	48	64	96	128
#variables	319	1194	2280	4395	6562	14122
#polynomials	287	1130	2184	4267	6370	13866
#terms	2262	10741	18199	40021	55512	134887
Bug-free	0.03	1.50	11.03	27.70	1802.75	10919.35
Bugs	0.03	1.52	11.10	28.18	1812.15	11047.10

Table 5.5. Runtime for verifying bug-free and buggy Barrett multipliers using our approach. TO = timeout of 10hrs. Time is given in seconds.

method	16	32	48	64	96	128	160	163
#variables	305	1103	2389	4146	9216	16072	24643	26847
#polynomials	276	1041	2263	4004	8986	15008	24318	25746
#terms	1777	6757	15228	26452	60824	107454	16386	174571
Bug-free	0.03	1.31	22.12	103.30	724.14	2865	9024	14048
Bugs	0.03	1.32	23.06	106.02	734.63	2947	9207	14836

sent point addition and point multiplication. We custom designed these circuits, where the polynomial computations were implemented using Mastrovito multipliers. Our approach is able to verify up to 163-bit ECC operations, whereas SAT, SMT, BDD and AIG-based techniques cannot even verify 16-bit ECC circuits.

Table 5.6. Verification of ECC point addition. Run-time given is seconds. TO = timeout of 24hrs.

Size	16	32	48	64	96	128	160	163
#variables	548	1615	3623	6854	13986	28468	30237	31384
#polynomials	10812	30826	86482	123544	288720	509660	604740	646129
Runtime	0.26	4.82	118	557	3598	15346	47290	81016

Table 5.7. Verification of ECC point doubling. Run-time given is seconds. TO = timeout of 24hrs.

Size	16	32	48	64	96	128	160	163
#variables	528	1598	3321	6409	12230	26493	29015	30442
#polynomials	4640	14523	42324	61274	142733	243452	297465	313145
Runtime	0.10	2.21	54	263	1532	8012	21493	36439

5.7 Conclusions

This chapter has presented a formal approach to model and verify multiplier circuits over finite fields $\mathbb{F}_{2^k}$ using a computer-algebra based approach. We show how the verification test can be formulated as membership testing of the specification polynomial f in a (radical) ideal $J + J_0 = \langle f_1, \ldots, f_s, x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d \rangle$, where $J = \langle f_1, \ldots, f_s \rangle$ corresponds to the ideal generated by polynomials extracted from the circuit, and $J_0 = \langle x_i^{2^k} - x_i \rangle$ corresponds to the ideal of vanishing polynomials of the field. By analyzing the circuit topology, we derive a monomial order that makes the set $\{f_1, \ldots, f_s, x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ itself a Gröbner basis of $J + J_0$. Subsequently, the verification can be formulated by simply carrying out the reduction $f \overset{J,J_0}{\rightarrow}_+ r$. Using our approach, we are able to verify the correctness of up to 163-bit multipliers and ECC point addition circuits over $\mathbb{F}_{2^{163}}$, whereas conventional techniques based on SAT, SMT, BDD and AIG-based solvers are infeasible. A conference paper based on this approach was presented in [20], and a journal version of this paper has been submitted for review.

CHAPTER 6

GATE-LEVEL EQUIVALENCE CHECKING OF ARITHMETIC CIRCUITS OVER FINITE FIELDS

This chapter describes our approach to equivalence checking of two combinational circuits designed for finite field computations. Combinational equivalence checking is a fundamental problem in hardware verification, and it has been widely investigated over the years. Canonical decision diagrams (BDDs and their variants), implication-based methods, SAT solvers, and And-Invert-Graph (AIG) based reductions, are among the many techniques employed for this purpose. When one circuit is synthesized from the other, this problem can be efficiently solved using AIG-based reductions (e.g., the ABC tool [8]) and circuit-SAT solvers (e.g., CSAT [9]). Synthesized circuits generally contain many sub-circuit equivalences which AIG and CSAT based tools can identify and exploit for verification. However, when the circuits are functionally equivalent but structurally very dissimilar, none of the contemporary techniques, including ABC and CSAT, offer a practical solution. Particularly, for *custom-designed arithmetic circuits* , this problem largely remains unsolved today. Since these custom designed circuits are prevalent in industry, it is therefore imperative to develop scalable methods to verify such circuits.

Focusing on finite field arithmetic circuits, we utilize computer algebra techniques and formulate the equivalence verification problem as a *Weak Nullstellensatz proof*, and solve it using Gröbner bases. This requires the computation of a reduced Gröbner basis, which can be expensive for large circuits. To overcome this complexity, we again wish to exploit the circuit topology-based term orderings (as described in the previous chapter) for polynomial manipulation. Unfortunately, unlike in the previous case, the set of polynomials corresponding to this verification instance (the miter circuit) does not constitute a Gröbner basis. However, using Gröbner bases theory, we identify *a*

minimum number of S-polynomial computations that are necessary and sufficient to prove or disprove equivalence. Experiments demonstrate the effectiveness and efficiency of our approach – we can verify 128-bit structurally very dissimilar implementations, while none of the contemporary methods are feasible.

6.1 Problem Statement and Modeling

In this application, we are given two combinational arithmetic circuits C_1 and C_2, as gate-level flattened netlists. We have to prove or disprove their functional equivalence.

Our approach is generic enough to perform equivalence checking of any arbitrary combinational arithmetic circuit over $\mathbb{F}_{2^k}$. However, without loss of generality, we will again consider finite field multiplier circuits as examples to explain our approach.

Our problem can be formally described as:

- Given a finite field $\mathbb{F}_{2^k}$, i.e. given k (datapath size), along with the corresponding irreducible polynomial $P(x)$. Let $P(\alpha) = 0$, i.e. α be the root of $P(x)$.

- Given two k-bit combinational circuits C_1 and C_2. The common primary inputs of both circuits are $\{a_0, \ldots, a_{k-1},\ b_0, \ldots, b_{k-1}\}$. The primary outputs of C_1 are $\{x_0, \ldots, x_{k-1}\}$; The primary outputs of C_2 are $\{y_0, \ldots, y_{k-1}\}$, where $a_i, b_i, x_i, y_i \in \mathbb{F}_2, i = 0, \ldots, k-1$.

- The word-level representation of inputs is $A = a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1}$, and $B = b_0 + b_1\alpha + \cdots + b_{k-1}\alpha^{k-1}$. Correspondingly, the outputs are $X = x_0 + x_1\alpha + \cdots + x_{k-1}\alpha^{k-1}$ and $Y = y_0 + y_1\alpha + \cdots + y_{k-1}\alpha^{k-1}$.

Our goal is to formally prove that $\forall a_i, b_i \in \mathbb{F}_2 \subset \mathbb{F}_{2^k}$, the outputs X and Y of circuits C_1 and C_2 are equal to each other, i.e., $X = Y$ always holds. Otherwise, there must exist a bug in one of the given circuits.

The equivalence verification setup is shown in Fig. 6.1. Given circuits C_1 and C_2, we want to prove that for all possible inputs, the output X of circuit C_1 is always equal to the output Y of circuit C_2 . This can be, conversely, modeled as proving that $X \neq Y$ has no solutions. Such a setup is called a “miter” circuit, and proving infeasibility of the miter is

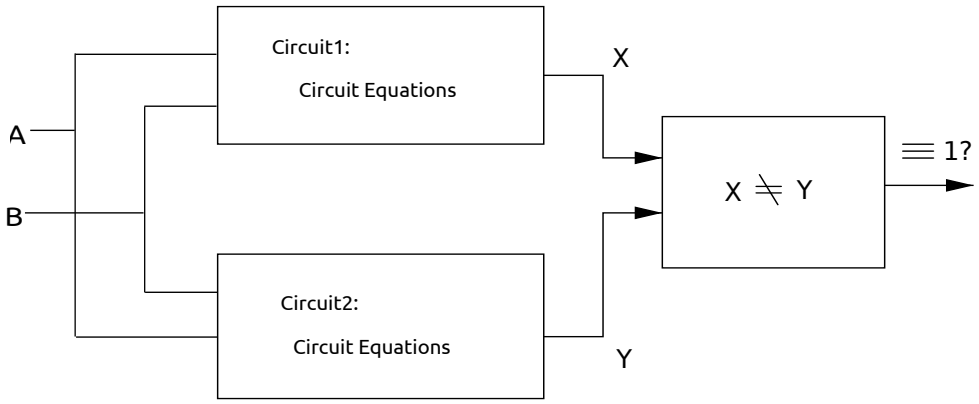

Figure 6.1. The equivalence checking setup: miter.

a standard practice in combinational circuit verification. This is mostly because it enables the use of *constraint-solvers* (such as SAT solvers) to prove/disprove equivalence.

The constraints for circuits C_1 and C_2 are modeled as polynomials over $\mathbb{F}_{2^k}$ using Equations 5.1. The $X \neq Y$ constraint corresponding to the miter is also modeled as a polynomial in $\mathbb{F}_{2^k}$ as follows:

$$t(X - Y) = 1, \text{where } t \text{ is a free variable in } \mathbb{F}_{2^k} \tag{6.1}$$

The correctness of the above constraint modeling can be shown as follows:

- When $X = Y, X - Y = 0$, so $t \cdot 0 = 1$ has no solutions, and the miter is infeasible.
- When $X \neq Y, (X - Y) \neq 0$. Over any field, every non-zero element has a multiplicative inverse. Let $t^{-1} = (X - Y)$. Then $t \cdot t^{-1} = 1$ will always have a solution over $\mathbb{F}_{2^k}$.

The above $t(X - Y) = 1$ model for the miter can also be employed over $\mathbb{F}_2$, i.e. the Boolean ring. Since 1 is the only non-zero element in $\mathbb{F}_2, t = 1$, and the $X \neq Y$ constraint is specified as $X + Y + 1 = 0 \pmod 2$.

Overall, the entire miter circuit can be modeled as a polynomial system over $\mathbb{F}_{2^k}$:

$$
\left.\begin{array}{r}
f_1^1(x_1, x_2, \cdots, x_d) \\
\vdots \\
f_A : A + a_0 + a_1\alpha + \cdots + a_{k-1}\alpha^{k-1} \\
f_X : X + x_0 + x_1 \cdot \alpha + \cdots + x_{k-1} \cdot \alpha^{k-1}
\end{array}\right\} \quad \textit{Circuit } 1
$$

$$
\left.\begin{array}{r}
f_1^2(x_1, x_2, \cdots, x_d) = 0 \\
\vdots \\
f_B : B + b_0 + b_1\alpha + \cdots + b_{k-1}\alpha^{k-1} \\
f_Y : Y + y_0 + y_1 \cdot \alpha + \cdots + y_{k-1} \cdot \alpha^{k-1}
\end{array}\right\} \quad \textit{Circuit } 2 \qquad (6.2)
$$

$$
\left.\begin{array}{r}
f_m : t \cdot (X - Y) + 1 = 0
\end{array}\right\} \quad \text{Miter:} X \neq Y
$$

Subsequently, we need to check whether or not there are any solutions to the set of polynomials in Equations 6.2. The following example illustrates our polynomial system modeling.

Example 6.1 *Consider two functionally equivalent circuits over* $\mathbb{F}_{2^2}$. *The miter is shown in Figure 6.2.*

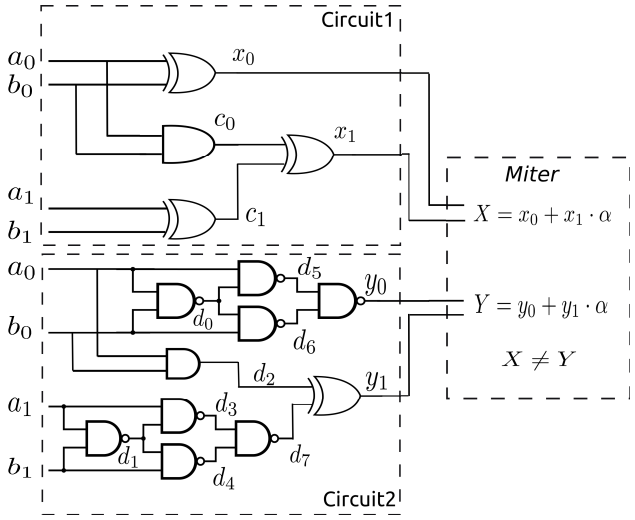

Figure 6.2. Miter for 2-bit circuit equivalence.

The miter is modeled as a system of polynomials, where the outputs of C_1, C_2 *are expressed at word level as:* $X + x_0 + x_1 \cdot \alpha$ *and* $Y + y_0 + y_1 \cdot \alpha$.

$$
\left.\begin{array}{r}
x_0 = a_0 \oplus b_0 \Rightarrow x_0 + a_0 + b_0 \\
c_0 = a_0 \wedge b_0 \Rightarrow c_0 + a_0 \cdot b_0 \\
c_1 = a_0 \oplus b_1 \Rightarrow c_1 + a_0 + b_1 \\
x_1 = c_0 \oplus c_1 \Rightarrow x_1 + c_0 + c_1 \\
X + x_0 + x_1 \cdot \alpha
\end{array}\right\} \quad \textit{Circuit } 1
$$

$$
\left.\begin{array}{r}
d_0 = \neg(a_0 \wedge b_0) \Rightarrow d_0 + a_0 \cdot b_0 + 1 \\
d_1 = \neg(a_1 \wedge b_1) \Rightarrow d_1 + a_1 \cdot b_1 + 1 \\
d_2 = a_0 \wedge b_0 \Rightarrow d_2 + a_0 \cdot b_0 \\
d_3 = \neg(a_1 \wedge d_1) \Rightarrow d_3 + a_1 \cdot d_1 + 1 \\
d_4 = \neg(b_1 \wedge d_1) \Rightarrow d_4 + b_1 \cdot d_1 + 1 \\
d_5 = \neg(a_0 \wedge d_0) \Rightarrow d_5 + a_0 \cdot d_0 + 1 \\
d_6 = \neg(b_0 \wedge d_0) \Rightarrow d_6 + b_0 \cdot d_0 + 1 \\
d_7 = \neg(d_3 \wedge d_4) \Rightarrow d_7 + d_3 \cdot d_4 + 1 \\
y_0 = \neg(d_5 \wedge d_6) \Rightarrow y_0 + d_5 \cdot d_6 + 1 \\
y_1 = d_2 \oplus d_7 \Rightarrow y_1 + d_2 + d_7 \\
Y + y_0 + y_1 \cdot \alpha
\end{array}\right\} \quad \textit{Circuit } 2
$$

$$
\left.\begin{array}{r}
t \cdot (X - Y) + 1 = 0
\end{array}\right\} \quad \textit{Miter:} X \neq Y \qquad (6.3)
$$

With the polynomial model given above, we formulate our problem as a **Weak Nullstellensatz** problem, which is described next.

6.1.1 Verification Problem Formulation as Weak Nullstellensatz

As described in Equation 6.2 and Example 6.1, to formulate our verification test, we first analyze the miter circuit and model the Boolean gate-level operators as polynomials over $\mathbb{F}_2$ – i.e. two sets of implementation polynomials representing C_1 and C_2, and the miter polynomials: $X \neq Y$ (X, Y are outputs of C_1 and C_2). Subsequently, we can

reason whether or not solutions exist to this polynomial system.

For this purpose, we wish to use techniques from computer algebra and algebraic geometry to reason about the solutions (variety) to the polynomial equations (ideal).

Notation: Let F_1, F_2 represent the set of polynomials generated from circuit C_1 and C_2, respectively. Let f_m represent the miter polynomial. Let $F = \{F_1, F_2, f_m\} = \{f_1, f_2, \ldots, f_s, f_m\}$ denote this set of polynomials derived from the miter circuit. Let $\{x_1, \ldots, x_d\}$ denote all variables occurring in F. Let $J = \langle F_1, F_2, f_m \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ denote the ideal generated by these polynomials. Subsequently, $V_{\mathbb{F}_{2^k}}(J)$ denotes the variety (solutions) of J over $\mathbb{F}_{2^k}$.

Our verification problem can be formulated as the evaluation:

$$V_{\mathbb{F}_{2^k}}(J) = \emptyset? \tag{6.4}$$

Weak Nullstellensatz [80] explicitly specifies the condition when a variety is empty.

Theorem 6.1 $[\mathbf{Weak\ Nullstellensatz}]$ *Let $J \subset \overline{\mathbb{K}}[x_1, x_2, \cdots, x_d]$ be an ideal satisfying $V_{\overline{\mathbb{K}}}(J) = \emptyset$. Then $I = \overline{\mathbb{K}}[x_1, x_2, \cdots, x_n] \iff \{1\} \in J$.*

Recall that a reduced Gröbner basis is a canonical representation of an ideal. We know that the unit ideal $\langle 1 \rangle$ can generate the entire set of polynomials in $\overline{\mathbb{K}}[x_1, x_2, \cdots, x_n]$. Therefore, Weak Nullstellensatz can be further described via Gröbner basis as:

Corollary 6.1 $[\mathbf{Weak\ Nullstellensatz}]$ *Let $I \subset \overline{\mathbb{K}}[x_1, x_2, \cdots, x_d]$ be an ideal satisfying $V(I) = \emptyset$. Then the Reduced GröbnerBasis(I)$= \{1\}$.*

The *Weak Nullstellensatz* now offers us a way to evaluate whether the system of multivariate polynomial equations has a common solution in $\overline{\mathbb{K}}^d$.

However, *Weak Nullstellensatz* is stated over an algebraically closed field $\overline{\mathbb{K}}$. Our problem is modeled over $\mathbb{F}_{2^k}$ which is not algebraically closed. Therefore, *Weak Nullstellensatz* is bound to fail when applied directly, without modification, to finite fields.

Let us explain why *Weak Nullstellensatz* fails when applying to the field $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$ by an example.

Example 6.2 *We are given an implementation of a circuit over* $\mathbb{F}_2 \subset \mathbb{F}_{2^k}$*:*

$$x_1 = a \vee (\neg a \wedge b) \tag{6.5}$$

Its corresponding specification is :

$$y_1 = a \vee b \tag{6.6}$$

where x_1 *and* y_1 *are symbolically different but functionally equivalent. Then we transform the circuit equations into their polynomial forms:*

$$\begin{aligned} x_1 = a \vee (\neg a \wedge b) &\mapsto x_1 + a + b \cdot (a+1) + a \cdot b \cdot (a+1) \pmod 2 \\ y_1 = a \vee b &\mapsto y_1 + a + b + a \cdot b \pmod 2 \\ x_1 \neq y_1 &\mapsto x_1 + y_1 + 1 \pmod 2 \end{aligned}$$

Then the reduced Gröbner basis of above polynomials with term ordering *lex* $x_1 > y_1 > a > b$ is:

$$\begin{aligned} a^2 \cdot b + a \cdot b + 1 \\ y_1 + a \cdot b + a + b \\ x_1 + a \cdot b + a + b + 1 \end{aligned}$$

which is not equal to $\langle 1 \rangle$, even though their variety is empty. The reason for this can be explained as follows.

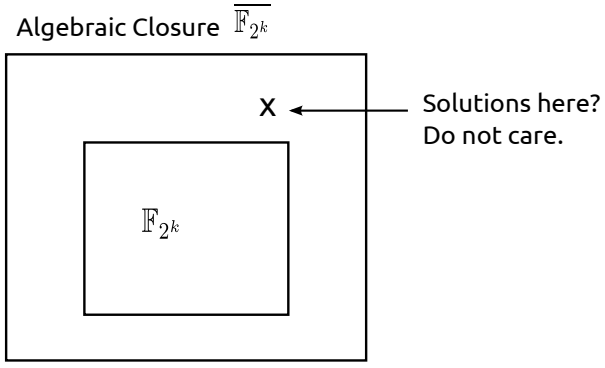

Figure 6.3. A solution (bug) in $(\overline{\mathbb{F}_{2^k}} - \mathbb{F}_{2^k})$ is a "don't care".

As shown in Figure 6.3, $\overline{\mathbb{F}_{2^k}}$ is the algebraic closure of $\mathbb{F}_{2^k}$. If there is no solution to ideal J in the algebraic closure $\overline{\mathbb{F}_{2^k}}$, then there is no solution in $\mathbb{F}_{2^k}$ either. However, what happens when there is a solution in $\overline{\mathbb{F}_{2^k}}$, i.e. $1 \notin GB(J)$? In this case, it means that there is a *non-empty set of solutions* to the polynomial system in $\overline{\mathbb{F}_{2^k}}^d$. There are two possibilities:

- The solution(s) may lie within $\mathbb{F}_{2^k}$.
- The solution(s) may lie in $\overline{\mathbb{F}_{2^k}}$, but outside $\mathbb{F}_{2^k}$, as depicted in Figure 6.3.

We are interested in finding out whether or not $X \neq Y$ over $\mathbb{F}_{2^k}$ – i.e. whether the circuit has bugs over the given field $\mathbb{F}_{2^k}$. We do not care if the solution is outside the field $\mathbb{F}_{2^k}$, in which case the bug is really a "don't care" condition (akin to a "false negative" in design verification parlance).

To address this problem, *Weak Nullstellensatz* needs to be suitably modified for application over finite fields $\mathbb{F}_{2^k}$.

Theorem 6.2 $[\textbf{Weak Nullstellensatz in } \mathbb{F}_{\mathbf{2^k}}]$
Given $f_1, f_2, \cdots, f_s \in \mathbb{F}_{2^k}[x_1, x_2, \cdots, x_d]$. Let $J = \langle f_1, f_2, \cdots, f_s \rangle \subset \mathbb{F}_{2^k}[x_1, x_2, \cdots, x_d]$ be an ideal. Let $J_0 = \langle x_1^{2^k} - x_1, x_2^{2^k} - x_2, \cdots, x_d^{2^k} - x_d \rangle$ be the ideal of vanishing polynomials in $\mathbb{F}_{2^k}$. Then $V_{\mathbb{F}_{2^k}}(J) = V_{\overline{\mathbb{F}_{2^k}}}(J + J_0) = \emptyset$, if and only if the reduced GröbnerBais$(J + J_0) = \{1\}$.

Proof. According to the definition of vanishing polynomials over $\mathbb{F}_{2^k}$, we have $V_{\overline{\mathbb{F}_{2^k}}}(J_0) = \mathbb{F}_{2^k}^d$. From Lemma 5.1, we know:

$$V_{\overline{\mathbb{F}_{2^k}}}(J + J_0) = V_{\mathbb{F}_{2^k}}(J). \tag{6.7}$$

Combining with Corollary 6.1, we conclude:

$$V_{\overline{\mathbb{F}_{2^k}}}(J + J_0) = \emptyset \Leftrightarrow \text{ reduced GröbnerBais}(J + J_0) = \{1\} \tag{6.8}$$

■

Example 6.3 *Re-visiting Example 6.2, we need to append the vanishing polynomials $a^2 - a, b^2 - b, x_1^2 - x_1, y_1^2 - y_1$ to given ideal. Now when we compute the reduced*

Gröbner basis, we get: reduced-GB$(x_1 + a + b \cdot (a+1) + a \cdot b \cdot (a+1), y_1 + a + b + a \cdot b, x_1 + y_1 + 1, a^2 - a, b^2 - b, x_1^2 - x_1, y_1^2 - y_1) = \{1\}$ *which proves* $x_1 = y_1$.

Verification Problem Formulation: Through Weak Nullstellensatz over $\mathbb{F}_{2^k}$, given an ideal $J \in \mathbb{F}_{2^k}[x_1, \ldots, x_d]$, we can determine whether the variety of J is empty by analyzing the corresponding reduced Gröbner basis of $J + J_0$.

For our verification problem, we take the polynomials $\{F_1, F_2, f_m\} = \{f_1, \ldots, f_s, f_m\}$ representing the miter circuit constraints to generate ideal J. Then we append the vanishing polynomials $\{x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ of ideal J_0. We compute the reduced Gröbner basis G of $J + J_0$ and check if G equals to the unit ideal $\{1\}$. The two circuits are functionally equivalent if and only if $G = \{1\}$.

The critical issue in the Weak Nullstellensatz formulation is the computational complexity of a Gröbner basis (as given in Theorem 5.3). To overcome this complexity, we again wish to exploit our circuit topology-based term ordering from Proposition 5.1 for polynomial representation. Note that according to the term ordering from Proposition 5.1, the set of polynomials in $\{F_1, F_2\}$ does constitute a Gröbner basis – as C_1 and C_2 are independent circuits. However, with the miter polynomial f_m, the set of polynomials $F = \{F_1, F_2, f_m\}$ does not constitute a Gröbner basis. This is because there always exists one polynomial $f_o \in F, (f_o \neq f_m)$ corresponding to the output of either C_1 or C_2 with a leading term that is not relatively prime w.r.t. the leading term of the miter polynomial f_m. Their corresponding S-polynomial computation also does not reduce to zero. This is shown in Example 6.4.

Example 6.4 *Let us re-consider Example 6.2. Based on our topological term ordering of the circuit, we impose a lex term order with:*
$x_1 > y_1 > d_4 > d_3 > d_2 > d_1 > d_0 > c_1 > c_0 > a_0 > a_1 > b_0 > b_1$,
Then the set of polynomials of the miter circuit $\{F_1, F_2, f_m\}$ *does not constitutes a Gröbner Basis. This is because the miter polynomial* $f_m : tX - tY + 1$ *and output polynomial* f_X *of circuit* C_1, $f_X : X + x_0 + x_1 \cdot \alpha$, *has a common variable* X *in their leading terms* tX *and* X, *respectively. Therefore,* $lt(f_m)$ *and* $lt(f_o)$ *are not relatively prime. Moreover,* $Spoly(f_m, f_X) \xrightarrow{F_1, F_2, f_m} r$, $r \neq 0$, *thus violating the property of a*

Gröbner basis that all S-polynomials should reduce to zero.

This suggests that we may have to compute a reduced Gröbner basis. However, in the next section, we describe our results that can identify *a minimum number of S-polynomial computations* that are sufficient and necessary to prove equivalence or to detect bugs.

6.2 Verification Uusing a Minimum Number of S-polynomial Computations

To identify a minimum number of S-polynomial computations in Buchberger's algorithm, we make use of the following lemma.

Lemma 6.1 *Let $r \in \mathbb{F}_2[x_1, \ldots, x_d]$ be a multi-linear polynomial expression; i.e. r is a nonconstant polynomial such that every monomial term in r contains variables of degree 1. Then r has a root in $\mathbb{F}_2^d$.*

Proof. Let $l(r)$ denote the number of nonzero monomials appearing in r. We will perform induction on $l(r)$. Note that in $\mathbb{F}_2$, the coefficient of all non-zero monomials is 1.

The case $l(r) = 1$ is trivial, as $r = x_1 x_2 \ldots x_t$, for some $t \leq d$. A polynomial with one monomial term always has a solution.

For the general case, $l(r) \geq 2$. Then we can always write $r = r' + M$ where M is a product of monomials. After appropriately re-labeling the variables, we can assume that x_1 divides M, i.e. x_1 appears in M. If x_1 divides r' too, then x_1 divides r as well. As a consequence, we obtain $x_1 = 0$ as a solution for $r = 0$. So, r has a root in $\mathbb{F}_2$.

If x_1 does not divide r, then it does not divide r'. So variable x_1 does not appear in r'. Then, let $r" = \mathcal{F}(0, x_2, \ldots, x_d)$. Note that $l(r") < l(r)$, as monomial M does not appear in $r"$. By induction, there is a solution $(x_2, \ldots, x_d)$ for $r" = 0$, which also gives a solution $(0, x_2, \ldots, x_d)$ for r. Thus r always has a root in F_2. ■

Now we state and prove the following theorem.

Theorem 6.3 *Let F_1, F_2 correspond to the set of polynomials derived from circuits C_1, C_2, respectively. Let f_m be the miter polynomial. Let $F = \{F_1, F_2, f_m\}$ and $J = \langle F \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ be the ideal of polynomials corresponding to the miter circuit. Im-*

pose the circuit topology-based monomial order $>$ from Proposition 5.1. Let $F_0 = \{x_1^{2^k} - x_1, \ldots, x_d^{2^k} - x_d\}$ be the vanishing polynomials of $\mathbb{F}_{2^k}$; and $J_0 = \langle F_0 \rangle$. Let $f_o \in F$ ($f_o \neq f_m$) be the only polynomial such that the leading terms of f_m, f_o are not relatively prime. Then $V_{\mathbb{F}_{2^k}}(J) = \emptyset \iff r = 1$, where r is computed as $Spoly(f_m, f_o) \xrightarrow{F,F_0}_+ r$.

Proof. Let $q = 2^k$, and let G and G_{red}, respectively, denote the Gröbner basis and the reduced Gröbner basis of $(J + J_0)$. Let T represent the set of all variables occurring in F, and let $T_{pi} \subset T$ denote the set of all primary inputs.

Our objective is to deduce whether or not the variety $V_{\mathbb{F}_{2^k}}(J) = \emptyset$, *without actually computing a reduced Gröbner basis*. Recall, according to Theorem 6.2, $V_{\mathbb{F}_q}(J) = \emptyset \iff G_{red} = \{1\}$, so we only need to check whether $G_{red} = \{1\}$? Based on our term ordering, we will try to identify the polynomials that constitute G_{red}.

In the first iteration of Buchberger's algorithm, $Spoly(f_m, f_o)$ is the only polynomial that needs to be computed and reduced to obtain r, as all other S-polynomials reduce to zero, due to Theorem 5.4. We need to consider three cases:

- Case 1: $r = 1$.
- Case 2: $r = 0$.
- Case 3: r is a non-constant multi-linear polynomial consisting of only primary input variables of the circuit.

Case 1 is the trivial case: If $r = 1$, then $1 \in G$, so $G_{red} = \{1\}$ and therefore $V(J + J_0) = \emptyset$. The miter is infeasible and the circuits are equivalent.

Case 2: When $r = 0$, no new polynomial is created in Buchberger's algorithm. Therefore $G = \{F, F_0\}$. While the set $\{F, F_0\}$ is itself a Gröbner basis, it is not reduced. So, what is the reduced basis G_{red}? We will show that $G_{red} \neq \{1\}$ and this will imply that $V(J + J_0) \neq \emptyset$.

To reduce a Gröbner basis G, we take all polynomials $f \in G$ and reduce $f \xrightarrow{G-f}_+ f'$. All such f' constitute G_{red}. We will consider such a reduction for $G = \{F, F_0\}$. For all $f_j \in F$, let $f_j = x_j + P_j$, where $P_j = \text{tail}(f_j)$ and $lm(f_j) = x_j$ where $x_j \notin T_{pi}$.

This is due to our term order where only gate outputs (x_j) appear as leading terms of all polynomials. Let v be any variable in P_j. If $v \in \{T - T_{pi}\}$ (non-primary-input), then $v = lm(f_k)$ ($k \neq j$). Thus $f_j \xrightarrow{\{F,F_0\}-f_j} f_j'$, where $f_j' = x_j + P_j'$. In such a case, P_j' contains only primary inputs. From a circuit-structure perspective, this reflects that any internal gate output x_j can be expressed in terms of primary inputs.

Similarly, $x_i^q - x_i$ with $x_i \in \{T - T_{pi}\}$ will reduce to zero, and only vanishing polynomials of primary inputs will remain in F_0. Moreover, since circuit inputs are bit-level, $x_{pi}^2 = x_{pi}$; so $x_{pi}^2 - x_{pi}$, $x_{pi} \in \{T_{pi}\}$, are the vanishing polynomials remaining in the reduced basis. Let $F' = \{x_j + P_j'\}$, where $x_j \in T$. Then, the reduced Gröbner basis G_{red} of $\{F, F_0\}$ = $reducedGB(\{F\} \cup \{x_i^q - x_i\}) = \{F'\} \cup \{x_{pi}^2 - x_{pi}\}$. Clearly, $G_{red} \neq 1$. We conclude, if $r = 0$, $G_{red} \neq \{1\}$, and $V(J + J_0) \neq \emptyset$. The miter constraints are feasible and the circuits are not equivalent.

Case 3: If r is a non-constant polynomial, then due to our term order and Corollary 5.1, r will contain only the primary input variables of the circuit. Moreover, as these variables are Boolean, $x_{pi}^2 = x_{pi}^3 = \cdots = x_{pi}$, all variables in the monomials of r have degree 1, and r is multi-linear.

After the first iteration of Buchberger's algorithm, we obtain $\{F, F_0, r\}$ in the basis. Because r contains only primary inputs, $lt(r)$ is relatively prime w.r.t. leading terms of all polynomials in F. So the Gröbner basis of $\{F, r\}$ is $\{F, r\}$ itself.

However, $\{F, r\} \cup \{F_0\}$ is *not* a Gröbner basis, because $lm(r)$ and $lm(x_k^q - x_k)$ are not relatively prime when $x_k \in T_{pi}$. Therefore, $G = GB(\{F, r\} \cup \{F_0\}) = \{F\} \cup GB(r \cup \{F_0\})$. In such a case, if we can show that $1 \notin GB(r \cup \{F_0\})$, then $1 \notin GB(\{F, F_0, r\})$.

To show $1 \notin GB(r \cup \{F_0\})$, we utilize the Weak Nullstellensatz Theorem 6.2: if $V(r \cup \{F_0\}) \neq \emptyset$, then $1 \notin GB(r \cup \{F_0\})$. In Lemma 6.1, we showed that if r is a multi-linear polynomial, it always has a root. This means that $V(r \cup \{F_0\}) \neq \emptyset$. Therefore $1 \notin GB(r \cup \{F_0\})$. This proves Case 3: if r is not 0 or 1, then $\{1\} \notin G = GB(F, F_0)$.

So, we conclude that:

$$V_{\mathbb{F}_{2^k}}(J) = \emptyset \iff r = 1. \tag{6.9}$$

■

Combining with Corollary 5.1, the above theorem can be re-stated based on a minimum Gröbner basis.

Corollary 6.2 *Let $J = \langle F \rangle \subset \mathbb{F}_{2^k}[x_1, \ldots, x_d]$ on which we impose our circuit-based monomial order $>$. Let $J_0^{PI} = \langle x_{pi}^2 - x_{pi} \rangle$, where $x_{pi} \in PI$. Let f_o, f_m be the only polynomial pair such that $lm(f_m), lm(f_o)$ are not relatively prime. Then $V_{\mathbb{F}_{2^k}}(J) = \emptyset \iff r = 1$, where r is computed as $Spoly(f_m, f_o) \xrightarrow{J, J_0^{PI}}_{+} r$.*

Theorem 6.3 and Corollary 6.2 provide the foundation of our verification formulation. We only need one S-polynomial computation to identify whether or not the two circuits are equivalent. Our overall approach is described in the following algorithm.

Algorithm 6: Our Proposed Equivalence Checking Algorithm

Input: Two Circuit Implementations with outputs X and Y (Boolean equations).
Output: 1 if $X = Y$. Bug polynomial r if $X \neq Y$.
for *(i=0; i < number of eqns; i++)* **do**
    /*Each equation is transformed to polynomials */;
    poly[i] = Eqn-to-Poly(eqn[i]);
    /*Each equation is transformed to sum-of-term*/;
    newpoly[i] = Sum-of-term(poly[i]);
end
/*Obtain circuit-based variable order*/;
ordered_var=T_Traversal(newpoly);
for $x \in \{PI\}$ **do**
    /*append vanishing polynomials*/;
    vanpoly[i]=$x^2 + x$;
end
/*Identify polynomials that need to be reduced*/;
f_o, f_m=Identify(newpoly, vanpoly);
To_Be_Reduced = Spoly(f_o, f_m);
r=reduce(To_Be_Reduced, vanpoly, ordered_var);
if $r=\{1\}$ **then**
    return 1;
else
    return Bug polynomial r;
end

Algorithm 5 first inputs the Boolean expressions of the given circuit implementation. Each expression is then transformed into polynomials F using the mappings shown in Equation 5.1. All polynomials are then normalized into a sum-of-term form using the distributive law $A(B + C) = AB + AC$. Then we perform a reverse topology traversal of the circuit to derive our variable and ordering. Then, we append vanishing polynomials $F_0 = \{x^2+x\}$ for all $x \in$ primary inputs. Subsequently, we identify the two polynomials f_m and f_o that have common variables in their leading terms. Finally, we conduct a polynomial reduction of $Spoly(f_m, f_o)$ modulo $\{F \cup F_0\}$. If the reduction result is $r = 1$, the two circuits are equivalent. If $r \neq 1$, the circuits are not equivalent. Again, any assignment to the variables that makes $r \neq 1$ provides an input vector that can be used as a counter-example for debugging.

6.3 Improving Polynomial Division using F_4-style Reduction

Through the results described above, the need for Buchberger's algorithm is obviated and verification can be performed by analyzing the result of just one S-polynomial reduction. Therefore, the most intensive computational step is that of polynomial division $Spoly(f_m, f_o) \xrightarrow{F,F_0}_+ r$. When the two circuits C_1, C_2 are very large, the polynomial set $\{F, F_0\}$ also becomes extremely large. This division procedure then becomes the bottleneck in verifying the equivalence. To further improve upon our approach, we exploit the relatively recent concept of F_4-style polynomial reduction [81], which implements polynomial division using successive row-reductions on a matrix.

Let us first describe the matrix representation for polynomial algebra operations.

Matrix representation of polynomials: Each row i of the matrix M corresponds to polynomial f_i, whereas each column j corresponds to monomial m_j. If the j^{th} entry on row i in matrix is 1, i.e. $M(i, j) = 1$, it means the j^{th} monomial is present in the i^{th} polynomial. Similarly, $M(i, j) = 0$ denotes the absence of m_j in f_i. Since we are operating in $\mathbb{F}_{2^k}$, coefficients are always $\{0, 1\}$, and no specific representation of coefficients is required. Note, however, that the entries in rows and columns have to satisfy the imposed term ordering.

Example 6.5 *Given two polynomials:* $f_1 = a_0 + a_1 \cdot b_1 + 1$ *and* $f_2 = a_0 \cdot b_0 + b_1 + 1$

Table 6.1. Matrix representation for polynomials.

	$a_0 \cdot b_0$	a_0	$a_1 \cdot b_1$	b_1	1
f_2	1	0	0	1	1
f_1	0	1	1	0	1

with term ordering lex with $a_0 > a_1 > b_0 > b_1$*. First, we sort all monomials occurring in* f_1 *and* f_2 *w.r.t. term ordering:* $a_0 \cdot b_0 > a_0 > a_1 \cdot b_1 > b_1 > 1$.

Then, we associate these sorted monomials with the columns of the matrix. The polynomials are also sorted according to the term order before they are associated with the rows of the matrix. For example, since $lm(f_2) > lm(f_1)$*,* f_2 *appears on row* 1 *and* f_1 *appears on row* 2*. The generated matrix is shown in Table 6.1.*

Polynomial reduction requires operations of addition/subtraction and cancellation of leading terms. We demonstrate how the addition/subtraction and division operations are implemented on

Matrix subtraction for polynomials: The subtraction of two polynomials can be formulated as a row-eduction in the matrix. Since coefficients of polynomials are computed (mod 2) in our case, row-reductions are also performed (mod 2).

Example 6.6 *Again consider* $f_1 = a_0 + a_1 \cdot b_1 + 1$ *and* $f_2 = a_0 \cdot b_0 + b_1 + 1$ *with lex order:* $a_0 > a_1 > b_0 > b_1$*. Let us perform* $f_1 - f_2$*:* $f_1 - f_2 = f_2 - f_1 \pmod 2 = a_0 \cdot b_0 + a_0 + a_1 \cdot b_1 + b_1$*. On the matrix, each entry on row* 2 *is subtracted from the corresponding entry on row* 1 *and the result is stored in row* 2*, as shown in Table 6.2.*

Matrix reduction for polynomials: Polynomial division is implemented as cancellation of leading terms. The reduction step in Algorithm 3 that cancels leading terms

Table 6.2. Matrix subtraction of polynomials.

	$a_0 \cdot b_0$	a_0	$a_1 \cdot b_1$	b_1	1
f_2	1	0	0	1	1
$f_2 - f_1$	1	1	1	1	0

Table 6.3. Matrix reduction for polynomials: representation.

	$a_0 \cdot b_1$	a_0	b_1	1
$b_1 \cdot f_2$	1	0	1	0
f_1	1	1	0	1

is:

$$f_1/f_2 = f_1 - \frac{lm(f_1)}{lm(f_2)} \cdot f_2 \tag{6.10}$$

In matrix representation, we create two rows, one each for f_1 and $\frac{lm(f_1)}{lm(f_2)} \cdot f_2$, and then perform subtraction on the matrix; this is shown in Example 6.7.

Example 6.7 *Given two polynomials:* $f_1 = a_0 \cdot b_1 + a_0 + 1$ *and* $f_2 = a_0 + 1$ *with term order lex:* $a_0 > a_1 > b_0 > b_1$. *Consider the polynomial reduction:*

$$f_1/f_2 = f_1 - \frac{a_0 \cdot b_1}{a_0} \cdot f_2 = f_1 - b_1 \cdot f_2$$

We create two rows in matrix for f_1 *and* $b_1 \cdot f_2$ *and insert monomials from* f_1 *and* $b_1 \cdot f_2$ *into the matrix columns, as shown in Table 6.3.*

Then we conduct $f_1 - b_1 \cdot f_2$*:*

Finally, row 2 *represents the reduction result of* $f_1/f_2 = a_0 + b_1 + 1$.

With the above basic polynomial operations formulated as matrix operations, we now describe our algorithm to create the matrix of polynomials corresponding to our verification instance (miter circuit). The algorithm is shown in Algorithm 7. The main idea behind this algorithm is to setup the rows of the matrix (polynomials) in a way that polynomial division can be subsequently performed by subtracting row i from row

Table 6.4. Matrix reduction for polynomials: subtraction.

	$a_0 \cdot b_1$	a_0	b_1	1
$b_1 \cdot f_2$	1	0	1	0
$f_1 - bf_2$	0	1	1	1

$i-1$. In the algorithm, the computation $L := L \cup \frac{mon}{lm(f_k)} \cdot f_k$ in the while-loop, actually corresponds to $\frac{lm(f_1)}{lm(f_2)} \cdot f_2$ in Equation 6.10.

Algorithm 7: Generating the Matrix for Polynomial Reduction

Input: $f, F = \{f_1, \ldots, f_s\}$ with $f_1 > f_2 > \cdots > f_s$.
Output: A matrix representing $f \xrightarrow{f_1,\ldots,f_s}_+ r$
/*Let L be the set of polynomials corresponding to rows of matrix*/;
L:={f} ;
/*The index of polynomials in F*/;
i:=1;
/*Let M_L be the set of monomials */;
M_L:={ monomials of f} ;
mon:= the i^{th} monomial of M_L;
while *mon* $\notin PrimaryInputs$ **do**
    Identify $f_k \in F$ satisfying: $lm(f_k)$ can divide mon ;
    /*add new polynomial to L as a new row in matrix*/;
    $L := L \cup \frac{mon}{lm(f_k)} \cdot f_k$;
    /*Add monomials to M_L as new columns in matrix */;
    M_L:=$M_L \cup$ {monomials of $\frac{mon}{lm(f_k)} \cdot f_k$} ;
    $i := i + 1$;
    mon:= the i^{th} monomial of M_L;
end

To better understand the algorithm, we describe the matrix construction procedure in Example 6.8.

Example 6.8 *Suppose that two functionally equivalent circuits and the miter are represented by the following polynomials at bit-level (i.e. over $\mathbb{F}_2$):*

$$
\begin{aligned}
f_m &= x + y + 1, \\
f_o &= x + n_0 + n_2, \\
f_1 &= y + n_{10}, \\
f_2 &= n_0 + i_2 \cdot i_3, \\
f_3 &= n_2 + i_0 \cdot i_1, \\
f_4 &= n_{10} + n_7, \\
f_5 &= n_7 + n_6 + n_4 \cdot i_0, \\
f_6 &= n_6 + n_5 + n_3 \cdot i_1, \\
f_7 &= n_5 + n_4 \cdot n_3, \\
f_8 &= n_4 + i_1 + i_3, \\
f_9 &= n_3 + i_0 + i_2;
\end{aligned}
$$

Note that $i_0, \ldots, i_3$ denote the primary inputs of the circuits. The circuit topology-based monomial order is derived as lex with $x > y > n_0 > n_2 > n_{10} > n_7 > n_6 > n_5 > n_4 > n_3 > i_0 > i_1 > i_2 > i_3$. All polynomials above have already been sorted (ordered) according to their leading terms in descending order. All monomials in each polynomial are also ordered.

In this case, $f = Spoly(f_m, f_o) = y + n_0 + n_2 + 1$ and $F = \{f_1, \ldots, f_9\}$. We want to show the algorithm's operation to construct a matrix for the reduction $f \xrightarrow{F}_+ r$.

Initialization:

$$
\begin{aligned}
L &:= \{f\}; \\
M_L &:= \{y, n_0, n_2, 1\}; \\
mon &:= y;
\end{aligned}
$$

Iteration $i = 1$:

$$\begin{aligned} f_k &:= f_1 = y + n_{10}; \\ L &:= \{f, f_1\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, 1\}; \\ i &:= 2; \\ mon &:= n_0 \end{aligned}$$

Iteration $i = 2$*:*

$$\begin{aligned} f_k &:= f_2 = n_0 + i_2 \cdot i_3; \\ L &:= \{f, f_1, f_2\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, i_2 \cdot i_3, 1\}; \\ i &:= 3; \\ mon &:= n_2 \end{aligned}$$

Iteration $i = 3$*:*

$$\begin{aligned} f_k &:= f_3 = n_2 + i_0 \cdot i_1; \\ L &:= \{f, f_1, f_2, f_3\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, i_0 \cdot i_1, i_2 \cdot i_3, 1\}; \\ i &:= 4; \\ mon &:= n_{10} \end{aligned}$$

Iteration $i = 4$*:*

$$\begin{aligned} f_k &:= f_4 = n_{10} + n_7; \\ L &:= \{f, f_1, f_2, f_3, f_4\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, n_7, i_0 \cdot i_1, i_2 \cdot i_3, 1\}; \\ i &:= 5; \\ mon &:= n_7 \end{aligned}$$

Iteration $i = 5$*:*

$$\begin{aligned} f_k &:= f_5 = n_7 + n_6 + n_4 \cdot i_0; \\ L &:= \{f, f_1, f_2, f_3, f_4, f_5\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_4 \cdot i_0, i_0 \cdot i_1, i_2 \cdot i_3, 1\}; \\ i &:= 6; \\ mon &:= n_6 \end{aligned}$$

Iteration $i = 6$*:*

$$\begin{aligned} f_k &:= f_6 = n_6 + n_5 + n_3 \cdot i_1; \\ L &:= \{f, f_1, f_2, f_3, f_4, f_5, f_6\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_5, n_4 \cdot i_0, n_3 \cdot i_1, i_0 \cdot i_1, i_2 \cdot i_3, 1\}; \\ i &:= 7; \\ mon &:= n_5 \end{aligned}$$

Iteration $i = 7$*:*

$$
\begin{aligned}
f_k &:= f_7 = n_5 + n_4 \cdot n_3; \\
L &:= \{f, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}; \\
M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_5, n_4 \cdot n_3, n_4 \cdot i_0, n_3 \cdot i_1, i_0 \cdot i_1, i_2 \cdot i_3, 1\}; \\
i &:= 8; \\
mon &:= n_4 \cdot n_3
\end{aligned}
$$

Iteration $i = 8$*:*

$$
\begin{aligned}
f_k &:= f_8 = n_4 + i_1 + i_3; \\
L &:= \{f, f_1, f_2, f_3, f_4, f_5, f_6, f_7, n_3 \cdot f_8\}; \\
M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_5, n_4 \cdot n_3, n_4 \cdot i_0, n_3 \cdot i_1, n_3 \cdot i_3, i_0 \cdot i_1, i_2 \cdot i_3, 1\}; \\
i &:= 9; \\
mon &:= n_4 \cdot i_0
\end{aligned}
$$

Iteration $i = 9$*:*

$$
\begin{aligned}
f_k &:= f_8 = n_4 + i_1 + i_3; \\
L &:= \{f, f_1, f_2, f_3, f_4, f_5, f_6, f_7, n_3 \cdot f_8, i_0 \cdot f_8\}; \\
M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_5, n_4 \cdot n_3, n_4 \cdot i_0, n_3 \cdot i_1, n_3 \cdot i_3, i_0 \cdot i_1, \\
&\quad\ i_0 \cdot i_3, i_2 \cdot i_3, 1\}; \\
i &:= 10; \\
mon &:= n_3 \cdot i_1
\end{aligned}
$$

Iteration $i = 10$*:*

$$\begin{aligned} f_k &:= f_9 = n_3 + i_0 + i_2; \\ L &:= \{f, f_1, f_2, f_3, f_4, f_5, f_6, f_7, n_3 \cdot f_8, i_0 \cdot f_8, i_1 \cdot f_9\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_5, n_4 \cdot n_3, n_4 \cdot i_0, n_3 \cdot i_1, n_3 \cdot i_3, i_0 \cdot i_1, \\ &\quad i_0 \cdot i_3, i_1 \cdot i_2, i_2 \cdot i_3, 1\}; \\ i &:= 11; \\ mon &:= n_3 \cdot i_3 \end{aligned}$$

Iteration $i = 11$*:*

$$\begin{aligned} f_k &:= f_9 = n_3 + i_0 + i_2; \\ L &:= \{f, f_1, f_2, f_3, f_4, f_5, f_6, f_7, n_3 \cdot f_8, i_0 \cdot f_8, i_1 \cdot f_9, i_3 \cdot f_9\}; \\ M_L &:= \{y, n_0, n_2, n_{10}, n_7, n_6, n_5, n_4 \cdot n_3, n_4 \cdot i_0, n_3 \cdot i_1, n_3 \cdot i_3, i_0 \cdot i_1, \\ &\quad i_0 \cdot i_3, i_1 \cdot i_2, i_2 \cdot i_3, 1\}; \\ i &:= 12; \\ mon &:= i_0 \cdot i_1 \end{aligned}$$

Termination: Because $i_0 \cdot i_1$ *contains variables* $\in PrimaryInputs$ *only.*

Each polynomial in L *corresponds to a row in the matrix and each monomial corresponds to a column. The generated matrix is shown in Table 6.5.*

With the generated matrix, the polynomial reduction can be formulated as a series of matrix subtractions, i.e., $Row_i - Row_{i-1}$*. After all row subtractions, the reduction result corresponds to the polynomial represented in the last row.*

Two important points to be noted:

- *All subtractions are computed modulo* 2*.*

Table 6.5. Matrix created for polynomial reduction for Example 6.8.

	y	n_0	n_2	n_{10}	n_7	n_6	n_5	$n_4 \cdot n_3$	$n_4 \cdot i_0$	$n_3 \cdot i_1$	$n_3 \cdot i_3$	$i_0 \cdot i_1$	$i_0 \cdot i_3$	$i_1 \cdot i_2$	$i_2 \cdot i_3$	1
f	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
f_1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
f_2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
f_3	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
f_4	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
f_5	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0
f_6	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0
f_7	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
$n_3 \cdot f_8$	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0
$i_0 \cdot f_8$	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0
$i_1 \cdot f_9$	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0
$i_3 \cdot f_9$	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0

- *If polynomials f_i and f_{i-1} have no common leading monomials, then they cannot conduct a reduction. Correspondingly, in the matrix, when conducting $Row_i - Row_{i-1}$, if the first non-zero entries of Row_i and Row_{i-1} are not in the same column (leading monomials), then we move on to the next row and perform $Row_{i+1} - Row_{i-1}$.*

This procedure is shown in Table 6.6 for $i_1 \cdot f_9 - i_0 \cdot f_8$: here $lm(i_1 \cdot f_9) = n_4 \cdot n_3$ while $lm(i_0 \cdot f_8) = n_4 \cdot i_0$. These leading monomial are not equal and they cannot divide each other. Thus we skip the current row ($i_1 \cdot f_9$). Instead, we move to the next row ($i_3 \cdot f_9$) and compute $i_3 \cdot f_9 - i_0 \cdot f_8$. Finally, the last entry in Table 6.6 corresponds to $r = 1$, and that denotes infeasibility of the miter circuit.

As shown in the above example, the polynomial reduction result r can be computed by successively subtracting rows i from rows $i + 1$. Finally, the last row represents r. If the last row only contains the monomial 1, the two circuits are equivalent. Otherwise, the polynomial corresponding to the last row represents the bug polynomial.

6.4 Experimental Results

The above verification approach using F_4-style reduction has been implemented in $C++$ as an efficient equivalence checking engine. Using this setup, we performed experiments to verify equivalence between different finite field multiplier implementations. Our experiments are conducted on a desktop with 2.40GHz Intel Core(TM)2 Quad CPU and 8GB memory running 64-bit Linux.

6.4.1 Equivalence Checking of Structurally Similar Circuits

To evaluate the performance of structurally similar circuits, we conduct a equivalence check between Mastrovito and Barrett multipliers. As shown in Chapter 3, Mastrovito and Barrett multipliers are somewhat structurally similar. Table 6.7 shows the results of verifying Mastrovito multipliers against Barrett multipliers. SAT solvers, ABC and CSAT can solve them reasonably fast. Singular can also verify these circuits within a matter of seconds. However, since Singular has a limitation on the number of variables it can accommodate (< 65535 variables), it cannot verify circuits larger than 96-bit

Table 6.6. Subtraction result of the matrix created for polynomial reduction.

	y	n_0	n_2	n_{10}	n_7	n_6	n_5	$n_4 \cdot n_3$	$n_4 \cdot i_0$	$n_3 \cdot i_1$	$n_3 \cdot i_3$	$i_0 \cdot i_1$	$i_0 \cdot i_3$	$i_1 \cdot i_2$	$i_2 \cdot i_3$	1
f	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
f_1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
f_2	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1
f_3	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1
f_4	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1
f_5	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
f_6	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1
f_7	0	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1
$n_3 \cdot f_8$	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1
$i_0 \cdot f_8$	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1
$\mathbf{i_1 \cdot f_9}$	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**
$i_3 \cdot f_9$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 6.7. Verification of Mastrovito multiplier vs. Barrett multiplier. TO=10hrs. $\star$=Out of variable limitation. Time is given in seconds.

Size	8	16	32	64	96	128	163
#variables	412	1445	4587	18953	42576	110543	195124
#gates	1446	6846	25846	101401	227499	403036	653021
MiniSAT	0.02	0.27	0.36	1.60	17.54	5.10	28.97
PicoSAT	0.02	0.15	0.78	3.90	6.58	41.89	130.56
PrecoSAT	0.05	0.40	1.61	22.98	91.90	90.25	187.53
CryptoMiniSAT	0.07	0.82	1.31	4.75	16.81	128.22	42.78
ABC	0.12	1.07	0.82	2.79	5.72	9.79	18.67
CSAT	0.03	3.02	0.58	0.87	1.83	5.97	5.49
Singular	0.03	0.17	0.41	1.12	$\star$	$\star$	$\star$
Ours (correct design)	0.00	0.01	0.01	0.02	0.03	0.05	0.12
Ours (buggy design)	0.00	0.02	0.02	0.02	0.04	0.06	0.13

circuits. The results also show that our approach is the most efficient in verifying circuit equivalence over finite fields.

6.4.2 Equivalence Checking of Structurally Dissimilar Circuits

As the experiments in Table 5.1 depict, given two structurally dissimilar circuits (such as a Mastrovito versus a Montogmery multiplier), none of SAT, SMT, BDD and AIG-based methods are able to verify the equivalence of circuits beyond 16-bits. The reason why ABC and CSAT are infeasible is that the structural hashing utilized by ABC and CSAT is not beneficial for structurally dissimilar circuits. It is unable to find common sub-circuit nodes as they do not really exist. Without merging internal sub-circuit equivalences, these tools are unable to reduce the size of the verification instance.

Our experiments perform verification between Montgomery multipliers on one hand, and Mastrovito and Barrett multipliers, on the other. Table 6.8 shows the runtimes of equivalence verification of Barrett versus Montgomery multipliers. Table 6.9 shows the runtimes for Mastrovito versus Montgomery multiplier verification. Singular can only verify 64-bit multipliers because of the limit on the number of variables it imposes. In contrast, our approach can successfully verify up to 128-bit multipliers with dissimilar structures. In the tables, note that the verification time for 128-bit multipliers is signifi-

Table 6.8. Verification of Barrett multiplier vs. Montgomery multiplier. TO=10hrs.$\star$=Out of variable limitation. Time is given in seconds.

Size	8	16	32	64	96	128	163
#variables	942	3426	9478	40059	98452	197841	286357
#gates	1968	8784	23548	86017	188121	330528	528903
Singular	0.05	486.74	3210.30	$\star$	$\star$	$\star$	$\star$
Ours (correct design)	0.00	0.13	3.39	125.88	1407.86	59.18	TO
Ours (buggy design)	0.00	0.13	3.41	127.03	1435.14	59.86	TO

cantly less than that of 96-bit ones. These experimental results are correct: we re-ran the experiments and also checked the circuit designs for errors – no errors were found. The reason for this anomaly may lie in the irreducible polynomials we selected to construct the circuits.

6.5 Limitation of Our Approach

While our approach is efficient verifying modulo-arithmetic circuits over finite fields $\mathbb{F}_{2^k}$, our approach cannot be applied to verify multiplier circuits over integers or over the finite ring $\mathbb{Z}_{2^k}$. This is due to the polynomial function representation of circuits over integers. The polynomial representation of circuits over finite fields has a much simpler form than that over integer rings. For example, circuits over finite fields are mainly constructed by *XOR* and *AND* gates which can be transformed into simple polynomials (mod 2):

Table 6.9. Verification of Mastrovito multiplier vs. Montgomery multiplier. TO=10hrs. Time is given in seconds.

Size	8	16	32	64	96	128	163
#variables	934	3387	9346	39654	99163	204972	294578
#gates	1958	8694	23318	86132	188526	331188	530278
Singular	0.05	446.83	3646.12	$\star$	$\star$	$\star$	$\star$
Ours (correct design)	0.00	0.12	3.29	126.01	1463.95	59.37	TO
Ours (buggy design)	0.00	0.13	3.31	127.45	1511.82	60.10	TO

$$a \wedge b \rightarrow a \cdot b \pmod 2$$
$$a \oplus b \rightarrow a + b \pmod 2$$

However, circuits over finite integer rings involve a large number of *OR* gates which are transformed into polynomials as:

$$a \vee b \rightarrow a + b + a \cdot b \pmod 2$$

Polynomial representations for *OR*-dominated functions include more monomial terms and also more occurrences of variables among the terms. This eventually results in size-explosion of the intermediate (remainder) polynomials in the reduction. Therefore our approach becomes infeasible in verifying integer arithmetic circuits over rings $\mathbb{Z}_{2^k}$. A conference paper corresponds to the initial theoretical model for this problem was published in [19] and a paper describing the efficient implementation of our approach is under submission [82].

CHAPTER 7

VERIFICATION OF COMPOSITE FIELD ARITHMETIC CIRCUITS

As an effort to reduce the high implementation costs, a methodology that designs arithmetic circuits over composite field is proposed [83], where the finite field $\mathbb{F}_{2^k}$ is decomposed as $\mathbb{F}_{(2^m)^n}$, for a $k = m \cdot n$, and the arithmetic operations are then performed over $\mathbb{F}_{(2^m)^n}$. The decomposition introduces a hierarchy (modularity) in the design by lifting the ground field from $\mathbb{F}_2$ (bits) to $\mathbb{F}_{2^m}$ (words). This results in impressive area and delay savings over large finite fields [83] [84] [85].

The hierarchy of composite field circuits also introduces a challenge to verify such problems: both word-level and bit-level information are contained in the designs, which are not able to be solved by any contemporary technique.

This chapter addresses the implementation verification of such arithmetic circuits. We formulate the verification problem as an (radical) ideal membership test at different abstraction levels and then apply approaches presented in Chapter 5 to solve it, i.e., conducting a polynomial reduction.

Our approach is based on the known field decomposition information and the circuit hierarchy. We utilize this information to:

- first verify the correctness of lower-level building-blocks (adders and multipliers) over the ground field $\mathbb{F}_{2^m}$;
- then verify the overall function at the higher-level over the extension field $\mathbb{F}_{(2^m)^n}$.

Using our approach, we are able to prove the correctness of finite field circuits for up to 1024-bit with decomposition $\mathbb{F}_{(2^{32})^{32}}$.

7.1 Circuit Designs over Composite Fields

The finite field $\mathbb{F}_{2^k}$ is a k-dimensional vector space over the sub-field $\mathbb{F}_2$. If $k = m \cdot n$, the field $\mathbb{F}_{2^k}$ can be decomposed as $\mathbb{F}_{(2^m)^n}$. Such a field representation is called a **composite field**, and it is constructed as a n-dimensional extension of the sub-field $\mathbb{F}_{2^m}$. The subfield $\mathbb{F}_{2^m}$ is called the ground field. Note that we have $\mathbb{F}_2 \subset \mathbb{F}_{2^m} \subset \mathbb{F}_{(2^m)^n}$.

According to Theorem 3.1, there exists an unique field of size p^k. This implies that $\mathbb{F}_{2^k}$ is isomorphic to $\mathbb{F}_{(2^m)^n}$ when $k = m \cdot n$, and due to this isomorphism, it is possible to derive one field representation from the other. The principle of constructing a composite field is described in [83]. Here we derive concrete steps for circuit design purpose.

Definition 7.1 *A* **primitive polynomial** $P(x)$ *is a polynomial with coefficients in* $\mathbb{F}_2$ *which has a root* $\alpha \in \mathbb{F}_{2^k}$ *such that* $\{0, 1, \alpha, \alpha^2, \cdots, \alpha^{2^k-2}\}$ *is the set of all elements in* $\mathbb{F}_{2^k}$*, where* α *is a* **primitive element** *of* $\mathbb{F}_{2^k}$.

The only difference between primitive polynomials and irreducible polynomials is whether they can generate all distinct elements of a finite field $\mathbb{F}_{2^k}$. Primitive polynomials can generate all elements with a primitive element of $\mathbb{F}_{2^k}$ while irreducible polynomials cannot generate all elements of $\mathbb{F}_{2^k}$.

Recall that to construct a finite field $\mathbb{F}_{2^k}$, we need a primitive polynomial $P(x) \in \mathbb{F}_2[x]$ of degree k. Similarly, to construct $\mathbb{F}_{(2^m)^n}$, we require a primitive polynomial, of degree n, with coefficients from the ground field $\mathbb{F}_{2^m}$. Given $\mathbb{F}_{2^k}$ and $P(x)$, the primitive polynomial of the composite field can be easily derived. We will use the following notation:

- Let $P(x)$ denote the given primitive polynomial of general field $\mathbb{F}_{2^k}$, and α be the primitive root, i.e. $P(\alpha) = 0$.
- Let $Q(x)$ denote the primitive polynomial of ground field $\mathbb{F}_{2^m}$, and β be the primitive root of $\mathbb{F}_{2^m}$, i.e. $Q(\beta) = 0$. Note that $Q(x)$ is a degree m primitive polynomial over $\mathbb{F}_2$ so it is also known.
- Let $R(x)$ denote the primitive polynomial of composite field $\mathbb{F}_{(2^m)^n}$, and γ be the primitive root, i.e. $R(\gamma) = 0$. This polynomial $R(x)$ has to be derived.

Lemma 7.1 *From [85]: Let $\mathbb{F}_{2^k}$ be decomposed as $\mathbb{F}_{(2^m)^n}$ where $k = m \cdot n$. Let γ be the primitive root of the field $\mathbb{F}_{(2^m)^n}$. Then*

$$R(x) = \prod_{i=0}^{i=n-1} (x_i + \gamma^{2^{m \cdot i}}) \tag{7.1}$$

Since $\mathbb{F}_{2^k}$ is isomorphic to $\mathbb{F}_{(2^m)^n}$, α and γ are actually the same elements. Now let us consider the representation of an element A in $\mathbb{F}_{2^k}$ and its corresponding representation in the composite field.

- Any element $A \in \mathbb{F}_{2^k}$ is represented as:

$$A = \sum_{i=0}^{i=k-1} a_i \cdot \alpha^i, a_i \in \mathbb{F}_2, \text{and } P(\alpha) = 0 \tag{7.2}$$

- The same element $A \in \mathbb{F}_{(2^m)^n}$ is represented as:

$$A = \sum_{i=0}^{i=n-1} A_i \cdot \gamma^i, A_i \in \mathbb{F}_{2^m}, \text{and } R(\gamma) = 0 \tag{7.3}$$

- Now we have to represent the element A_i from above in the ground field $\mathbb{F}_{2^m}$:

$$A_i = \sum_{j=0}^{j=m-1} a_{ij} \cdot \beta^j, a_{ij} \in \mathbb{F}_2, \text{and } Q(\beta) = 0 \tag{7.4}$$

Now we need to find the relationship between the primitive roots α and β (or between γ and β, since $\alpha = \gamma$), so as to be able to map the elements from $\mathbb{F}_{2^k}$ to $\mathbb{F}_{(2^m)^n}$. We have the following result [85]:

Theorem 7.1 *For $\gamma \in \mathbb{F}_{(2^m)^n}$, and $\beta = \gamma^\omega$, where $\omega = (2^{m \cdot n} - 1)/(2^m - 1)$, then we have $\beta \in \mathbb{F}_{2^m}$. In other words:*

$$\beta = \alpha^{(2^{m \cdot n} - 1)/(2^m - 1)} = \gamma^{(2^{m \cdot n} - 1)/(2^m - 1)} \tag{7.5}$$

The above result states the following: Since γ is a primitive root, it can be used to generate all the non-zero elements of $\mathbb{F}_{(2^m)^n}$. Moreover, β is a primitive root of the ground field $\mathbb{F}_{2^m}$, which is a sub-field of $\mathbb{F}_{(2^m)^n}$ (i.e. $\mathbb{F}_{2^m} \subset \mathbb{F}_{(2^m)^n}$); so $\beta \in \mathbb{F}_{(2^m)^n}$.

Therefore an exponent of γ can be used to generate β as $\beta = \gamma^{\omega}$, where ω is given in Theorem 7.1. Now we know all the relationships between α, β, γ, and we are ready to perform the decomposition.

Example 7.1 *As an example, let us reconsider the field $\mathbb{F}_{2^4}$ and decompose it as $\mathbb{F}_{(2^2)^2}$. Let $P(x) = x^4 + x^3 + 1$ and $P(\alpha) = 0$. We need to perform the following steps:*

1. *Derivation of $R(x)$:*

$$\begin{aligned} R(x) &= \prod_{i=0}^{i=1}(x + \gamma^{2^{2 \cdot i}}) \\ &= (x + \gamma) \cdot (x + \gamma^{2^2}) \\ &= x^2 + (\gamma^4 + \gamma) \cdot x + \gamma^5 \end{aligned} \tag{7.6}$$

   Notice that $R(\gamma) = \gamma^2 + (\gamma^4 + \gamma) \cdot \gamma + \gamma^5 = 0$.

2. *Representation of element $A \in \mathbb{F}_{(2^2)^2}$:*

$$\begin{aligned} A &= \sum_{i=0}^{i=1} A_i \cdot \gamma^i, A_i \in \mathbb{F}_{2^2} \\ &= A_0 + A_1 \cdot \gamma \end{aligned} \tag{7.7}$$

3. *Representation of A_0, A_1 in $\mathbb{F}_{2^m}$:*

$$\begin{aligned} A_0 &= a_{00} + a_{01} \cdot \beta \\ A_1 &= a_{10} + a_{11} \cdot \beta \end{aligned} \tag{7.8}$$

   where $a_{ij} \in \mathbb{F}_2$. $Q(x)$ can be any degree $m = 2$ primitive polynomial in the ground field $\mathbb{F}_{2^2}$. Let us take $Q(x) = x^2 + x + 1$.

4. *Now we can substitute A_0, A_1 into A as follows:*

$$\begin{aligned} A &= \sum_{i=0}^{i=1}(\sum_{j=0}^{j=1} a_{ij} \cdot \beta^j) \cdot \gamma^i \\ &= a_{00} + a_{01} \cdot \beta + (a_{10} + a_{11} \cdot \beta) \cdot \gamma \end{aligned} \tag{7.9}$$

where each $a_{ij} \in \mathbb{F}_2$. From Eqn. (7.5), we have: $\beta = \alpha^5 = \gamma^5$. We then substitute β and γ with α to obtain:

$$\begin{aligned} A &= \sum_{i=0}^{i=1}(\sum_{j=0}^{j=1} a_{ij} \cdot \beta^j) \cdot \gamma^i \\ &= a_{00} + a_{01} \cdot \alpha^5 + (a_{10} + a_{11} \cdot \alpha^5) \cdot \alpha \end{aligned}$$

Since $P(x) = x^4 + x^3 + 1$ with $P(\alpha) = 0$, we have

$$A \pmod{P(\alpha)} = a_{00} + a_{01} + a_{11} + (a_{01} + a_{10} + a_{11}) \cdot \alpha + a_{11} \cdot \alpha^2 + (a_{01} + a_{11}) \cdot \alpha^3 \tag{7.10}$$

5. *The same element $A \in \mathbb{F}_{2^4}$ is represented as:*

$$A = a_0 + a_1 \cdot \alpha + a_2 \cdot \alpha^2 + a_3 \cdot \alpha^3 \tag{7.11}$$

6. *Since Eqns. 7.10 and 7.11 represent the same element, we can match the coefficients of the the polynomials to obtain:*

$$\begin{aligned} a_0 &= a_{00} + a_{01} + a_{11} \\ a_1 &= a_{01} + a_{10} + a_{11} \\ a_2 &= a_{11} \\ a_3 &= a_{01} + a_{11} \end{aligned}$$

This mapping can also be reversed and represented as a matrix T:

$$\begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Now we have successfully derived the composite field representation $\mathbb{F}_{(2^2)^2}$ from $\mathbb{F}_{2^4}$. The element $A \in \mathbb{F}_{2^4}$ is represented as $A = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$, where $P(\alpha) = 0$. The same element A is represented in $\mathbb{F}_{(2^2)^2}$ as:

$$\begin{aligned} A &= A_0 + A_1 \cdot \alpha \\ A_0 &= a_{00} + a_{01} \cdot \alpha^5 \\ A_1 &= a_{10} + a_{11} \cdot \alpha^5 \\ a_{00} &= a_0 + a_3 \\ a_{01} &= a_2 + a_3 \\ a_{10} &= a_1 + a_3 \\ a_{11} &= a_2 \end{aligned}$$

In the above equations, $\alpha = \gamma$ and $R(\gamma) = 0$.

Multiplication $A \cdot B \pmod{P(x)}$ over $\mathbb{F}_{2^4}$ can now be performed over the decomposition $\mathbb{F}_{(2^2)^2}$, where $A = A_0 + A_1\gamma, B = B_0 + B_1\gamma$ and the modulus is taken over $R(\gamma)$. Such a design is shown in Figure 7.2, where $a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3$ are primary inputs. After a suitable transformation, *composite field* inputs are obtained as $a_{00}, a_{01}, a_{10}, a_{11}, b_{00}, b_{01}, b_{10}, b_{11}$. A_0, A_1, B_0, B_1 are 2-bit buses. Correspondingly, each block in Fig.7.2 internally represents a 2-bit operation: $\times$ represents 2-bit *multiplication* and $+$ represents 2-bit *addition* over the ground field. A logic circuit for a 4-bit *Mastrovito* multiplier over *finite field* $\mathbb{F}_{2^4}$ is illustrated in Fig.7.1.

Its corresponding composite field design with decomposition $\mathbb{F}_{(2^2)^2}$ is shown in Figure 7.2. Each block in Fig.7.2 represents a 2-bit operation internally, where $\times$ represents an m-bit multiplier and $+$ represents an m-bit adder.

7.2 Problem Formulation and Hierarchy Verification

Let us again take the multiplier verification problem as example. The *specification* $S = A \cdot B \pmod{P(x)}$ is already given in polynomial form (word level). The *implementation* is available at two different abstraction levels: one at the bit-level (ground field $\mathbb{F}_{2^m}$ adders and multipliers) and one at the higher-level at $\mathbb{F}_{(2^m)^n}$. Using this information,

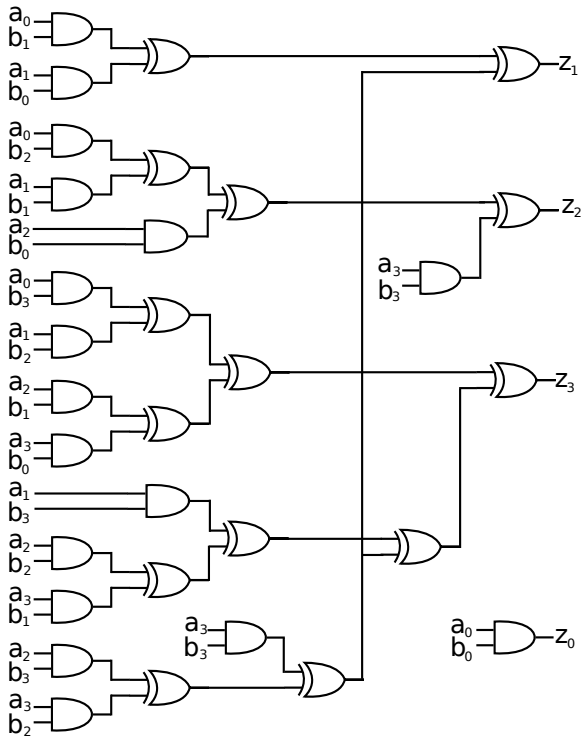

Figure 7.1. Mastrovito multiplier over $\mathbb{F}_{2^4}$.

we derive constraints (polynomials) Z corresponding to the circuit. Our verification problem is to prove/disprove that for all values of the inputs $A = \{a_0, \ldots a_{k-1}\}$, $B = \{b_0, \ldots b_{k-1}\}$, the circuit implementation Z correctly computes the multiplication S.

As we can notice from Figure 7.2, the entire composite field circuit is constructed on lower level building-blocks (adders and multipliers). Therefore, we have two verification objectives: low level circuits and higher-level interconnection of the lower-level blocks.

Verification of low level circuits over $\mathbb{F}_{2^m}$:

Low level building-blocks consist of adders and multipliers over $\mathbb{F}_{2^m}$. These circuits are implemented at gate-level and are nothing special as the regular finite field circuits we verified before. Therefore, we can simply employ the same methods described in Chapter 5 to formulate the verification test as membership testing of the property polynomial ($S + Z = 0$). When the correctness of low level circuits is certified, we can conduct the high level verification over $\mathbb{F}_{(2^m)^n}$.

Verification of higher-level interconnection over $\mathbb{F}_{(2^m)^n}$: The difficulty of verifying

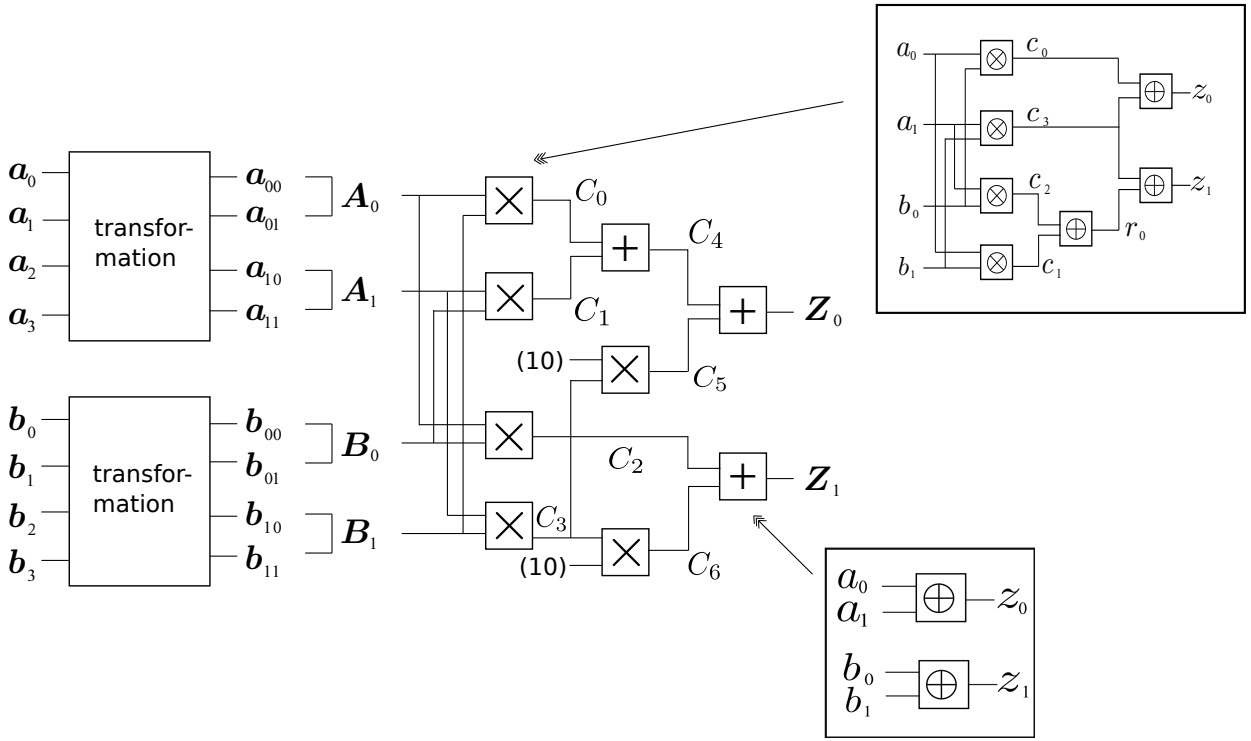

Figure 7.2. Mastrovito multiplier over $\mathbb{F}_{(2^2)^2}$

the composite field circuits lies in the verification of high level interconnection of low level building-blocks. Specifically, due to the presence of hierarchy of composite field circuits, the constraints derived from the high level interconnection contain both gate-level and word-level abstractions. For example, in Figure 7.2, the circuit hierarchy can be described as follows:

$$
\begin{aligned}
a_{00} &= a_0 + a_3 \\
a_{01} &= a_2 + a_3 \\
a_{10} &= a_1 + a_3 \\
a_{11} &= a_2 \\
A_0 &= a_{00} + a_{01} \cdot \alpha^5 \\
A_1 &= a_{10} + a_{11} \cdot \alpha^5 \\
b_{00} &= b_0 + b_3 \\
b_{01} &= b_2 + b_3 \\
b_{10} &= b_1 + b_3 \\
b_{11} &= b_2 \\
B_0 &= b_{00} + b_{01} \cdot \alpha^5 \\
B_1 &= b_{10} + b_{11} \cdot \alpha^5 \\
C_0 &= A_0 \cdot B_1 \\
C_1 &= A_1 \cdot B_0 \\
C_2 &= A_0 \cdot B_0 \\
C_3 &= A_1 \cdot B_1 \\
C_4 &= C_0 + C_1 \\
C_5 &= C_3 \cdot \alpha^5 \\
C_6 &= C_3 \cdot \alpha^5 \\
Z_0 &= C_4 + C_5 \\
Z_1 &= C_2 + C_6
\end{aligned}
\tag{7.12}
$$

where $a_0, \ldots, a_3, b_0, \ldots, b_3$ are variables in $\mathbb{F}_2$ (bits) while $A_0, A_1, B_0, B_1, C_1, \ldots, C_6$, Z_0, Z_1 are variables in $\mathbb{F}_{2^2}$ (words). Therefore bit-level variables and word-level variables co-exist in the design. As far as we know, there are no techniques that can verify design with different levels of abstraction. This is mainly because BDD/SAT/AIG based approaches can only handle bit-level problems. SMT solvers, on the other hand, have no advantages to solve problems at bit-level. Besides, SMT solvers formulate every problem over rings instead of finite fields. Take Eqnations 7.12 for example, $C_0 = A_0 \cdot B_1$ represents a 2-bit finite field multiplication. In SMT, $C_0 = A_0 \cdot B_1$ represents a 2-bit integer multiplication. As we know, the multiplication over rings and over finite fields differs significantly.

Fortunately, due to the fact that both bits and words information can be formulated as polynomials, this verification problem is algebraic in nature and therefore can be easily formulated as a system of polynomials and solved by ideal membership testing, which is described in Algorithm 5.

Example 7.2 *Our high-level verification problem is illustrated in Table 7.1. Let F denote all the polynomials representing implementation, specification and vanishing polynomials. Let F_0 denote the vanishing polynomials for primary inputs. After all the polynomials in $\{F\}$ are available, we just need to check whether $S + Z$ is a member of the ideal $\langle F, F_0 \rangle$.*

7.3 Experimental Results

With the approach presented above, we have conducted experiments to hierarchically verify Mastrovito multiplier implementations M against the specification $S = A \cdot B \pmod{P(x)}$. Our verification setup is shown of Table 7.1. The implementation is given as a circuit over $\mathbb{F}_{(2^m)^n}$. With the given hierarchy information, we construct the polynomials representing high level designs M_H over $\mathbb{F}_{(2^m)^n}$ and low level designs M_L over $\mathbb{F}_{2^m}$ separately.

For high level designs M_H, the specification polynomials $S = A \cdot B \pmod{P(x)}$ is used. In contrast, for low level designs M_L over $\mathbb{F}_{2^m}$, the specification polynomials $S_L = A_m \cdot B_m \pmod{Q(x)}$ is used, of which, A_m, B_m represents the m-bit inputs for

Table 7.1. Verification Setup over $\mathbb{F}_{(2^2)^2}$

implementation	*specification*	*vanishing polynomials*
$a_{00} + a_0 + a_3$	$A + a_0 + a_1 \cdot \alpha + a_2 \cdot \alpha^2 + a_3 \cdot \alpha^3$	$a_0^2 - a_0$
$a_{01} + a_2 + a_3$	$B + b_0 + b_1 \cdot \alpha + b_2 \cdot \alpha^2 + b_3 \cdot \alpha^3$	$a_1^2 - a_1$
$a_{10} + a_1 + a_3$	$S + A \times B$	$a_2^2 - a_2$
$a_{11} + a_2$		$a_3^2 - a_3$
$A_0 + a_{00} + a_{01} \cdot x^5$		$b_0^2 - b_0$
$A_1 + a_{10} + a_{11} \cdot x^5$		$b_1^2 - b_1$
$b_{00} + b_0 + b_3$		$b_2^2 - b_2$
$b_{01} + b_2 + b_3$		$b_3^2 - b_3$
$b_{10} + b_1 + b_3$		
$b_{11} + b_2$		
$B_0 + b_{00} + b_{01} \cdot x^5$		
$B_1 + b_{10} + b_{11} \cdot x^5$		
$C_0 + A_0 \cdot B_0$		
$C_1 + A_1 \cdot B_0$		
$s_2 + A_1 \cdot B_1$		
$C_3 + A_1 \cdot B_1$		
$C_4 + C_0 + C_1$		
$C_5 + C_3 \cdot \alpha^5$		
$C_6 + C_3 \cdot \alpha^5$		
$Z_0 + C_4 + C_5$		
$Z_1 + C_2 + C_6$		
$Z + Z_0 + Z_1 \cdot \alpha$		
Property: $\mathbf{Z + S}$		

low level building-block circuits; $Q(x)$ is the primitive polynomial of $\mathbb{F}_{2^m}$. Then vanishing polynomials $a_0^2 - a_0, \ldots, a_{k-1}^2 - a_{k-1}, b_0^2 - b_0, b_{k-1}^2 - b_{k-1}$ are then appended to M_H and M_L at different levels of design. We use SINGULAR [68] to conduct polynomial reduction. When the circuits are correctly designed, we do observe that reduction result is 0, proving the equivalence.

Our experiments are conducted on a desktop with 2.40GHz CPU and 8GB memory running 64-bit Linux. The time-out limit is set as 24 hours.

The verification of low level circuits is the same as the one shown in Table 5.3. The number of low level design units is shown in Table 7.3. Note that this number is determined by n, which means $\mathbb{F}_{(2^{m_1})^n}$ and $\mathbb{F}_{(2^{m_2})^n}$ have the same number of low level

design units, even if $m_1 \neq m_2$.

Since high level verification cannot be solved by any other technique, we only show the results of our approach. Table 7.2 shows the runtime of high level designs verification over $\mathbb{F}_{(2^m)^n}$ for varying word-size $k = m \cdot n$. As shown in Table 7.2, with our approach, we are able to prove the correctness of finite field circuits for up to 1024-bit with decomposition $\mathbb{F}_{(2^{32})^{32}}$.

7.4 Conclusions

This chapter has targeted the implementation verification of hierarchically designed composite finite field circuits. Decomposing the finite field $\mathbb{F}_{2^k}$ as $\mathbb{F}_{(2^m)^n}$ introduces a hierarchical abstraction. Our approach requires that this hierarchy information be made available. Then, we formulate the verification problem using the polynomial reduction as a ideal membership testing at different levels of abstraction. First we verify low-level adders and multipliers at $\mathbb{F}_{2^m}$, and then verify the high-level interconnections between these blocks at $\mathbb{F}_{(2^m)^n}$. Using our approach, we can verify the correctness of up to 1024-bit multipliers where other contemporary techniques are not capable of verifying such circuits. This work was presented in [17].

Table 7.2. Verification of Mastrovito multiplier over $\mathbb{F}_{(2^m)^n}$ Using Proposed Approach. All times are given in seconds.

32			64			128			256			512			1024		
m	n	time	m	n	time	m	n	time	m	n	time	m	n	time	m	n	time
2	16	7.55	2	32	879.83	2	64	$*$	2	128	$*$	2	256	$*$	2	512	$*$
4	8	0.12	4	16	10.81	4	32	1619.51	4	64	$*$	4	128	$*$	4	256	$*$
8	4	0.01	8	8	0.46	8	16	35.04	8	32	2664.56	8	64	$*$	8	128	$*$
16	2	0.01	16	4	0.15	16	8	3.25	16	16	147.84	16	32	11510	16	64	$*$
-	-	-	32	2	0.11	32	4	2.14	32	8	37.71	32	16	1166.10	32	32	75336

Table 7.3. Statistics of Designs over $\mathbb{F}_{2^m}$

n	2	4	8	16	32
#Multipliers	6	36	168	720	2976
#Adders	3	27	147	675	2883

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This dissertation presents approaches to performing equivalence checking for arithmetic circuits over finite fields $\mathbb{F}_{2^k}$. In particular, we target two specific problems: i) verifying the correctness of a custom-designed arithmetic circuit implementation against a given word-level polynomial specification over $\mathbb{F}_{2^k}$; and ii) gate-level equivalence checking of two structurally dissimilar arithmetic circuits. We propose polynomial abstractions over finite fields to model and represent the circuit constraints. Subsequently, decision procedures based on modern computer algebra techniques – notably Gröbner bases related theory and technology – are engineered to solve the verification problem efficiently.

8.1 Computer Algebra Based Approaches for Equivalence Checking of Arithmetic Circuit over $\mathbb{F}_{2^k}$

The arithmetic circuit is modeled as a polynomial system in the ring $\mathbb{F}_{2^k}[x_1, x_2, \cdots, x_d]$, and computer-algebra and algebraic-geometry based results (Hilbert's Nullstellensatz) over finite fields are exploited for verification. Two formulations are presented to address the implementation verification and the equivalence checking problems.

Using the results of Strong Nullstellensatz over finite fields, the first verification problem is formulated as an ideal membership testing. For this ideal membership test, it is required to compute a Gröbner basis. The Gröbner basis computation is known to have double-exponential worst-case complexity in the input data, which makes this approach impractical. Therefore, straight-forward use of Gröbner basis engines for verification is infeasible for large circuits. To overcome this complexity, we analyze the given circuit

topology to get more theoretical insights into the polynomial ideals corresponding to the circuit constraints. Based on this circuit information, we derive efficient term orderings to represent the polynomials. Subsequently, using the theory of Gröbner bases over finite fields, we prove that our term orderings render the set of polynomials itself a Gröbner basis – thus obviating the need for Buchberger's algorithm. To fulfill our verification purpose, we simply conduct a polynomial reduction to test whether the equality property is a member of the ideal representing the circuit constraints.

The equivalence checking for two structurally dissimilar arithmetic circuits is still a challenge for contemporary techniques. By utilizing computer algebra theory, we formulate this problem as a weak Nullstellensatz proof using Gröbner bases computation. Once again, this would require the computation of a reduced Gröbner basis, which is expensive for large circuits. To overcome this complexity, we want to exploit our circuit-based term ordering for polynomial representation. Unfortunately, unlike in the previous case, the set of polynomials corresponding to this verification instance does not constitute a Gröbner basis. Instead of computing a Gröbner basis for the the whole circuit, we identify a minimal number of S-polynomial computations that are sufficient to prove equivalence or to detect bugs for the whole circuit.

The verification of composite field circuits is a successful application of our computer algebra based approaches. To construct a composite field circuit over $\mathbb{F}_{(2^m)^n}$, the finite field $\mathbb{F}_{2^k}$ is decomposed as $\mathbb{F}_{(2^m)^n}$, for a $k = m \cdot n$, and the arithmetic operations are then performed over $\mathbb{F}_{(2^m)^n}$. The decomposition introduces a hierarchy (modularity) in the design by lifting the ground field from $\mathbb{F}_2$ (bits) to $\mathbb{F}_{2^m}$ (words). We formulate the verification problem as an (radical) ideal membership test at different abstraction levels. By combining the circuit hierarchy information, we first verify the correctness of lower-level building-blocks (adders and multipliers) over the ground field $\mathbb{F}_{2^m}$; then verify the overall arithmetic at the higher-level over the extension field $\mathbb{F}_{(2^m)^n}$.

8.2 Future Work

The approaches and theories presented in this dissertation can be further extended to enhance the efficiency of equivalence checking of arithmetic circuits. Some future research directions are proposed here.

8.2.1 Speeding up Verification using a Graphics Processing Unit

As shown in Figure 6.2, the equivalence of "CIRCUIT1" and "CIRCUIT2" is formulated as a single miter at word-level. However, since the circuits have multiple outputs (k), we can create k miters for each output bit. In such cases, we will have to compute $Spoly(f_m, f_o) \xrightarrow{F,F_0}_+ r$ for each of the k outputs, and check if $r = 1$ in each case. These are going to be n independent computations. In that regard, they will immensely benefit from parallelization.

It is desirable to implement this technique on a hardware accelerator - particularly on a NVIDIA Graphics Processing Unit (GPU). In the Electronic Design Automation (EDA) community, there has been a lot of interest in exploiting GPU computing to improve synthesis and verification algorithms. Significant speed-ups have been observed in GPU implementation of circuit simulation algorithms (see for example [86]). It is needed to further study how to efficiently implement our circuit verification problem using independent S-polynomial reductions on a general purpose GPU.

8.2.2 Extraction of Circuit Abstraction

Suppose that we are given a circuit that implements a polynomial function over $\mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$, but we do not know what function does it implement. Can we identify a polynomial representation of this function: $f(X, Y)$ where X represents the input bit-vector and Y the output? This problem is one of hierarchy abstraction and is used in component matching and resource allocation in high-level synthesis.

To explain this idea, let us re-visit the example of Figure 5.2, a 2-bit multiplier. It implements a polynomial function $Z = A * B;\ Z, A, B \in \mathbb{F}_{2^2}$. Here $A = a_0 + a_1\alpha, B =$

$b_0 + b_1\alpha, Z = z_0 + z_1\alpha$. Let us represent a polynomial for each gate in the circuit. We will impose the following term order: **lex term order** with "circuit Variables" $>$ "Inputs, A, B" $>$ "Output Z". That is, we use lex term order with $c_0 > c_1 > c_2 > c_3 > r_0 > a_0 > a_1 > b_0 > b_1 > z_0 > z_1 > A > B > Z$. If we use this order to compute a Gröbner basis of the circuit polynomials, then we obtain the following polynomials:

$$f_1 : z_0 + z_1\alpha + Z$$

$$f_2 : b_0 + b_1\alpha + B$$

$$f_3 : a_0 + a_1\alpha + A$$

$$f_4 : c_3 + r_0 + z_1$$

$$f_5 : c_1 + c_2 + r_0$$

$$f_6 : c_0 + c_3 + z_0$$

$$f_7 : A \cdot B + Z$$

$$f_8 : a_1 \cdot b_1 + a_1 \cdot B + b_1 \cdot A + z_1$$

$$f_9 : r_0 + a_1 \cdot b_1 + z_1$$

$$f_{10} : c_2 + a_1 \cdot b_0$$

Notice that the polynomial $f_7 : A * B + Z$ is indeed the polynomial representation of the function implemented by the circuit. And we were able to "extract" the polynomial representation using Gröbner basis.

Polynomial interpolation techniques for this problem were studied in [87] [88]. Further research should be conducted to investigate if we can use Gröbner basis techniques to efficiently interpolate a polynomial representation from a circuit.

8.2.3 Simulation Based Verification of Circuits

In our group's previous work [89] [90], we show that given two polynomial functions f, g over $\mathbb{Z}_{2^k}$, exhaustive simulation is not always necessary to prove their equivalence. We identified an integer λ such that functions (polynomials) f, g need to be evaluated only for λ inputs vectors: $\{V_1, \ldots, V_\lambda\}$. If $f = g$ for these λ vectors, then $f = g$ over the

entire design space. If $f \neq g$, then we guarantee to catch the bug within these λ vectors. In practice, $\lambda << 2^k$.

Unfortunately, this result did not find much practical application as it required that f, g be polynomial functions. Not every function (circuit) $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$ is a polynomial function. Instead of modeling a k-input/output circuit as a function from $f : \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^k}$, We conjecture the model can be viewed as a polynomial function over finite fields $f : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$. Though this way, we can then prove equivalence of two polyfunctions $f, g : \mathbb{F}_{2^k} \rightarrow \mathbb{F}_{2^k}$ without resorting to exhaustive simulation. It is promising to solve the same problem as in [89] [90], but now over a different domain: $\mathbb{F}_{2^k}$.

REFERENCES

[1] E. Biham, Y. Carmeli, and A. Shamir, "Bug Attacks", *in Proceedings on Advances in Cryptology*, pp. 221–240, 2008.

[2] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, First edition, 1991.

[3] E. Clarke, O. Grumberg, and D. Peled, *The Temporal Logic of Reactive and Concurrent Systems*, The MIT Press, 1999.

[4] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vencentelli, F. Somenzi, A. Aziz, S-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, G. Shiple, S. Swamy, and T. Villa, "VIS: A System for Verification and Synthesis", *in Computer Aided Verification*, 1996.

[5] K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[6] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley Professional, First edition, September 2003.

[7] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams", *in Proceedings of Design Automation Conference*, pp. 535–541, 1995.

[8] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", *in Computer Aided Verification*, vol. 6174, pp. 24–40. Springer, 2010.

[9] F. Lu, L. Wang, K. Cheng, and R. Huang, "A Circuit SAT Solver With Signal Correlation Guided Learning", *in IEEE Design, Automation and Test in Europe*, pp. 892–897, 2003.

[10] A. Gupta, "Formal Hardware Verification Methods: A Survey", *Formal Methods in System Design*, vol. 1, pp. 151–238, 1992.

[11] D. Stoffel and W. Kunz, "Verification of Integer Multipliers On the Arithmetic Bit Level", *in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 183–189, Piscataway, NJ, USA, 2001. IEEE Press.

[12] M. Clegg, J. Edmonds, and R. Impagliazzo, "Using the Gröbner Basis Algorithm to Find Proofs of Unsatisfiability", *in ACM Symposium on Theory of Computing*, pp. 174–183, 1996.

[13] G. Avrunin, "Symbolic Model Checking using Algebraic Geometry", *in Computer Aided Verification Conference*, pp. 26–37, 1996.

[14] C. Condrat and P. Kalla, "A Gröbner Basis Approach to CNF formulae Preprocessing", *in International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 618–631, 2007.

[15] Y. Watanabe and *et al*, "Application of Symbolic Computer Algebra to Arithmetic Circuit Verification", *in IEEE International Conference on Computer Design*, pp. 25–32, October 2007.

[16] O. Wienand, M. Wedler, D. Stoffel, W. Kunz, and G. Gruel, "An Algebraic Approach to Proving Data Correctness in Arithmetic Datapaths", *in Computer Aided Verification Conference*, pp. 473–486, 2008.

[17] J. Lv, P. Kalla, and F. Enescu, "Verification of Composite Galois Field Multipliers over GF($(2^m)^n$) using Computer Algebra Techniques", *in IEEE High-Level Design Validation and Test Workshop*, pp. 136–143, 2011.

[18] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, A. Dreyer, F. Seelisch, and G.-M. Greuel, "STABLE: A New QBF-BV SMT Solver for Hard Verification Problems Combining Boolean Reasoning with Computer Algebra", *in IEEE Design, Automation and Test in Europe Conference*, pp. 155–160, 2011.

[19] J. Lv, P. Kalla, and F. Enescu, "Formal Verification of Galois Field Multipliers using Computer Algebra", *in 25th IEEE International Conference on VLSI Design*, 2012.

[20] J. Lv, P. Kalla, and F. Enescu, "Efficient Groebner Basis Reductions for Formal Verification of Galois Field Multipliers", *in IEEE Design, Automation and Test in Europe*, 2012.

[21] B. Buchberger, *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, PhD thesis, University of Innsbruck, 1965.

[22] W. W. Adams and P. Loustaunau, *An Introduction to Gröbner Bases*, American Mathematical Society, 1994.

[23] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, 2007.

[24] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, August 1986.

[25] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving", *in Communications of the ACM*, vol. 5, pp. 394–397, 1962.

[26] E. A. Emerson, "Temporal and Modal Logic", *in Formal Models and Semantics*, vol. B of *Handbook of Theoretical Computer Science*, pp. 996–1072. Elsevier Science, 1990.

[27] A. Narayan, J. Jain, M. Fujita, and A. Sangiovanni-Vincentelli, "Partitioned ROBDDs: A Compact Canonical and Efficient Representation for Boolean Functions", *in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 547–554, 1996.

[28] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications", *in Proceed-*

ings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 188–191, Nov. 93.

[29] S. Horeth and Drechsler, "Formal Verification of Word-Level Specifications", *in IEEE Design, Automation and Test in Europe*, pp. 52–58, 1999.

[30] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid Decision Diagrams - Overcoming the Limitation of MTBDDs and BMDs", *in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 159–163, 1995.

[31] R. Dreschler, B. Becker, and S. Ruppertz, "The K*BMD: A Verification Data Structure", *IEEE Design & Test of Computers*, vol. 14, pp. 51–59, 1997.

[32] M. Ciesielski, P. Kalla, Z. Zheng, and B. Rouzyere, "Taylor Expansion Diagrams: A Compact Canonical Representation with Applications to Symbolic Verification", *in IEEE Design, Automation and Test in Europe*, pp. 285–289, 2002.

[33] M. Ciesielski, P. Kalla, Z. Zheng, and B. Rouzyere, "Taylor Expansion Diagrams: A New Representation For RTL Verification", *in IEEE International High Level Design Validation and Test Workshop*, pp. 70–75, Nov. 2001.

[34] P. Kalla, M. Ciesielski, and E. Boutillon, "High-Level Design Verification using Taylor Expansion Diagrams: First Results", *in IEEE International High Level Design Validation and Test Workshop*, pp. 13–17, 2002.

[35] Priyank Kalla, *An Infrastructure for RTL Validation and Verification*, PhD thesis, University of Massachusetts Amherst, 2002.

[36] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory", *Journal of the ACM*, vol. 7, pp. 201–215, 1960.

[37] J. Silva and K. Sakallah, "GRASP: A New Search Algorithm for Satisfiability", *in Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 220–227. IEEE Computer Society, 1996.

[38] B. Dutertre and L. Moura, "The Yices SMT Solver", Technical report, 2006.

[39] R. Bruttomesso, A. Cimatti, A. Franzen, A. Griggio, and R. Sebastiani, "The MathSAT 4 SMT Solver", *in Computer Aided Verification Conference*, vol. 5123. Springer, 2008.

[40] R. Brummayer and A. Biere, "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays", *in TACAS 09, Volume 5505 of LNCS*. Springer, 2009.

[41] S. Jha, R. Limaye, and S. Seshia, "Beaver: Engineering An Efficient SMT Solver for Bit-Vector Arithmetic", *in Computer Aided Verification Conference*, pp. 668–674, 2009.

[42] F. Lu, L. Wang, K. Cheng, J. Moondanos, and Z. Hanna, "A Signal Correlation Guided ATPG Solver And Its Applications For Solving Difficult Industrial Cases", *in Design Automation Conference*, pp. 436–441, 2003.

[43] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 1377–1394, Nov. 2006.

[44] N. Shekhar, P. Kalla, and F Enescu, "Equivalence Verification of Polynomial Datapaths using Ideal Membership Testing", *IEEE Transactions on CAD*, pp. 1320–1330, July 2007.

[45] S. Morioka, Y. Katayama, and T. Yamane, "Towards Efficient Verification of Arithmetic Algorithms Over Galois Fields $GF(2^m)$", *Computer Aided Verification Conference*, vol. 2102, pp. 465–477, 2001.

[46] D. Mukhopadhyaya, G. Sengar, and D. Chowdhury, "Hierarchical Verification of Galois Field Circuits", *IEEE Transactions on CAD*, 2007.

[47] T. L. Rajaprabhu, A. K. Singh, A. M. Jabir, and D. K. Pradhan, "MODD for CF: A Compact Representation for Multiple Output Function", *in IEEE International High Level Design Validation and Test Workshop*, 2004.

[48] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Ordered Kronecker Functional Decision Diagrams", *in Design Automation Conference*, pp. 415–419, 1994.

[49] A. Jabir and Pradhan D., "MODD: A New Decision Diagram and Representation for Multiple Output Binary Functions", *in IEEE Design, Automation and Test in Europe*, 2004.

[50] Robert J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

[51] S. Roman, *Field Theory*, Springer, 2006.

[52] R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge University Press, 1997.

[53] E. Mastrovito, "VLSI Designs for Multiplication Over Finite Fields GF(2^m)", *Lecture Notes in Computer Science*, vol. 357, pp. 297–309, 1989.

[54] P. Montgomery, "Modular Multiplication Without Trial Division", *Mathematics of Computation*, vol. 44, pp. 519–521, Apr. 1985.

[55] C. Koc and T. Acar, "Montgomery Multiplication in GF(2^k)", *Designs, Codes and Cryptography*, vol. 14, pp. 57–69, Apr. 1998.

[56] H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields", *IEEE Transactions On Computers*, vol. 51, May 2002.

[57] M. Knežević, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF(2^n) Without Pre-Computational Phase", *in Proceedings of the International Workshop on Arithmetic of Finite Fields*, pp. 77–87, 2008.

[58] ST Microelectronics, *ST23YLxx series Microcontroller for Smart Cards*.

[59] K. Kobayashi, *Studies on Hardware Assisted Implementation of Arithmetic Operations in Galois Field*, PhD thesis, Nagoya University, Japan, 2009.

[60] S. Morioka and Y. Katayama, "Design methodology for a one-shot reed-solomon encoder and decoder", *in IEEE International Conference on Computer Design*, pp. 60–67, 1999.

[61] Y. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID", *IEEE Transactions on Computers*, vol. 57, pp. 1514–1527, Nov. 2008.

[62] D. Hankerson, J. Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields", 2000.

[63] V. Miller, "Use of Elliptic Curves in Cryptography", *in Lecture Notes in Computer Sciences*, pp. 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[64] P. Barrett, "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor", *in Proceedings of Advances In Cryptology*, pp. 311–323, London, UK, UK, 1987. Springer-Verlag.

[65] J. López and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)", *in Proceedings of the Selected Areas in Cryptography*, pp. 201–212, London, UK, UK, 1999. Springer-Verlag.

[66] S. Gao, "Counting Zeros over Finite Fields with Gröbner Bases", Master's thesis, Carnegie Mellon University, 2009.

[67] B. Buchberger, "A criterion for detecting unnecessary reductions in the construction of a groebner bases", *in EUROSAM*, 1979.

[68] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-3 — A computer algebra system for polynomial computations", 2011, http://www.singular.uni-kl.de.

[69] J. Lv, "SAT and SMT benchmarks", `http://ece.utah.edu/~lv/uugb.html`.

[70] M. Soos, "Cryptominisat-a SAT Solver for Cryptographic Problems", `http://www.msoos.org/cryptominisat2/`, 2009.

[71] A. Biere, "SAT 2009 Competition", 2009.

[72] N. E*én* and Niklas Sörensson, "An Extensible SAT-solver", *Theory And Applications of Satisfiability Testing*, vol. 2919, pp. 333–336, 2004.

[73] A. Biere, "Picosat Essentials", *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, vol. 4, pp. 75–97, 2008.

[74] C. Barrett and C. Tinelli, "CVC3", *in Computer Aided Verification Conference*, pp. 298–302. Springer, July 2007.

[75] L. Moura and N. Bjrner, "Z3: An Efficient SMT Solver.", *in TInternational Conference on Tools and Algorithms for the Construction and Analysis of Systems*, vol. 4963. Springer, 2008.

[76] "Sonolar, SMT-COMP2010", `http://www.smtcomp.org/2010`.

[77] "SimplifyingSTP, SMT-COMP2010", `http://www.smtcomp.org/2010`.

[78] F. Somenzi, "CUDD: CU Decision Diagram Package Release", 1998.

[79] J. L*ó*pez, R. Dahab, and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)", pp. 201–212. Springer-Verlag, 1998.

[80] David Hilbert, "Über die Theorie der algebraischen Formen", *Math. Annalen*, vol. 36, pp. 473–534, 1890.

[81] J. C. $Faugère$, "A New Efficient Algorithm for Computing Gröbner Bases (F_4)", *Journal of Pure and Applied Algebra*, vol. 139, pp. 61–88, June 1999.

[82] J. Lv, P. Kalla, and F. Enescu, "Scalable Equivalence Checking of Finite Field Arithmetic Circuits using Gröner Bases and F-4 Style Reduction", *in IEEE Design Automation and Test in Europe*, 2013.

[83] C. Paar, *Efficient VLSI Architecture for Bit-Parallel Computation in Galois Fields*, PhD thesis, University of Essen, Germany, 1994.

[84] C. Paar, "A New Architecture for A Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields", *IEEE Transactions on Computers*, vol. 45, pp. 856–861, July 1996.

[85] B. Sunar, E. Savas, and C. Ko, "Constructing Composite Field Representations for Efficient Conversion", *IEEE Transactions on Computers*, vol. 52, pp. 1391–1398, November 2003.

[86] Z. Feng, Z. Zeng, and P. Li, "Parallel On-Chip Power Distribution Network Analysis on Multicore GPU Platforms", *IEEE Transactions VLSI*, 2011.

[87] J. Smith and G. DeMicheli, "Polynomial methods for component matching and verification", *in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1998.

[88] J. Smith and G. DeMicheli, "Polynomial Methods for Allocating Complex Components", *in IEEE Design, Automation and Test in Europe*, 1999.

[89] N. Shekhar, P. Kalla, M. B. Meredith, and F. Enescu, "Simulation Bounds for Equivalence Verification of Polynomial Datapaths using Finite Ring Algebra", *IEEE TransactionsVLSI*, vol. 16, pp. 376–387, 2008.

[90] N. Shekhar, P. Kalla, M. B. Meredith, and F. Enescu, "Simulation Bounds for Equivalence Verification of Arithmetic Datapaths with Finite Word-Length Operands", *in Formal Methods in Computer Aided Design*, pp. 179–186, November 2006.