Elevating Boolean Matching to the Word Level

Jiun-Hao Chen[1], Hsin-Ying Tsai[2], Kuo-Wei Ho[2] and Jie-Hong Roland Jiang[1,2,3]
[1]Graduate School of Advanced Technology, National Taiwan University
[2]Graduate Institute of Electronics Engineering, National Taiwan University
[3]Department of Electrical Engineering, National Taiwan University
{r12k41016, r12943092, r11943109, jhjiang}@ntu.edu.tw

ABSTRACT

Boolean matching, a widely employed technique in industrial applications, has conventionally centered on the bit domain. This research addresses a critical gap in the existing literature by extending Boolean matching to accommodate the complexities introduced by modern digital Integrated Circuit (IC) designs featuring datapaths and buses with many input/output ports. Our approach abstracts a gate-level netlist to a word-level expression by reverse engineering techniques and transforms the matching procedure into a Satisfiability Modulo Theories (SMT) problem. Our method may powerfully discover functional correspondence between two circuits under comparison beyond the standard negation and permutation equivalences of Boolean matching. Experimental results validate the efficacy of our proposed methodology in solving arithmetic benchmarks with extended Boolean matching constraints beyond the capability of standard (bit-level) Boolean matching algorithms. This paper contributes to advancing Boolean matching techniques, providing a valuable framework for handling intricate design elements in contemporary ICs.

1 INTRODUCTION

Boolean matching is the problem of determining whether two (possibly multi-output) Boolean functions are functionally equivalent up to some input and output correspondence constraints. There are various types of constraints, such as NPNP-equivalence (Negation and Permutation of input and Negation and Permutation of output), PP-equivalence (Permutation of input and Permutation of output), NP3-equivalent (Non-exact Projective NPNP) [20], etc. In all variant problems, NP3-equivalence is the most general problem formulation. Boolean matching finds many applications in logic synthesis and verification. For instance, NPNP-equivalence between two circuits can achieve better quality in synthesis [21] and FPGA technology mapping [8]. Large-scale Boolean matching has been used in engineering change order [10, 12], equivalence checking [15], etc.

The importance of Boolean matching triggers the challenge posed by the 2023 ICCAD CAD Contest, Problem A: Multi-bit Large-scale Boolean Matching [6]. The task is to check the functional equivalence between two gate-level netlists under NP3-equivalence constraints with additional bus information. The bus information in the challenge motivates us to elevate the traditional (bit-level) Boolean matching to the word level. We refer to the extended word-level Boolean matching problem as *elevated Boolean matching*. The formal definition of this problem will be given in Section 3. This work aims to tackle this extended problem.

There have been extensive efforts in traditional Boolean matching, which can be classified into three categories: signature-based, canonical-form-based, and SAT-based methods. Signature-based methods [1, 5] detect and prune impossible matching solutions by some signatures. Canonical-form-based methods [2, 3] compare the canonical form of two netlists to find a feasible solution. Although the signature-based and canonical-based methods are effective, they are either incomplete or non-scalable. In contrast, recent satisfiability (SAT) based methods [11, 13, 14, 17] achieve high efficiency and scalability compared to the methods in the other two categories.

When solving Boolean matching for arithmetic circuits, traditional methods involve bit-blasting the entire arithmetic formula into a Conjunctive Normal Form (CNF) formula for SAT solving, which often results in the loss of word-level structure information and, thus, computation inefficiency. In contrast, using the bit-vector theory in Satisfiability Modulo Theories (SMT) solving may leverage the characteristics of the theory to achieve greater efficiency. Moreover, previous methods only focus on bit-level manipulation. We additionally consider the integer word (bit-vector) negation in 2's complement, which means flipping all bits and adding a constant 1. This requires additional logic such as an adder or an incrementer, which is beyond the capability of traditional Boolean matching. From a practical standpoint, this additional aspect extends the application of traditional Boolean matching in logic synthesis and verification. For instance, it facilitates macro matching [18] to identify embedded arithmetic macros, assists in handling pin order faults, missing pins, or negations in engineering change orders for word-level datapath patching, provides a debugger for designers to verify the design after optimization, and more, to arithmetic circuits.

To overcome the limitation of prior works, our work extracts the word-level function from the gate-level netlist and then performs the word-level matching. This not only prevents computational inefficiency but also shifts the focus from a bit-level to a word-level perspective. The primary results of this work include the following:

- This is the first work that performs Boolean matching to identify word-level functional correspondence between two netlists.
- Besides the bit-level correspondence constraints, such as bit-flip, constant insertion, bit duplication, the elevated Boolean matching further allows word-level negation, which cannot be solved by any previous

methods. We show that if an elevated Boolean matching solution exists, the word-level function of two netlists is the same up to input word permutation and coefficient sign change subject to possible input bit-flip, constant insertion, and bit duplication.
- We formulate the procedure of extracting word-level function from a pseudo-Boolean polynomial into an SMT problem, detailed in Section 4.2.
- An elevated Boolean matching flow is proposed, integrating the above techniques to solve the task.
- The experimental results of benchmark circuits demonstrate the effectiveness of our approach and suggest computation bottlenecks for future improvement.

In this work, the crucial step is the polynomial rewriting introduced in Section 2.2. Polynomial rewriting has proven successful in verifying large arithmetic circuits [4] by deriving a bit-level representation independent of the circuit structure. We can extract the word-level functions along with bitwise manipulation by analyzing the resulting pseudo-Boolean polynomial. This approach provides us an opportunity to perform Boolean matching at the bit-level and further consider word-level matching.

The remainder of the paper is organized as follows. Section 2 gives the preliminaries, and the problem statement is formalized in Section 3. In Section 4, the proposed algorithms are given. Finally, Section 6 shows the experimental results, and Section 7 concludes the paper.

2 PRELIMINARIES

2.1 Boolean Matching

Let each signal in a netlist be associated with a Boolean variable $x \in \mathbb{B} = \{0, 1\}$. A *literal* is a variable x or the negation $\neg x$ of a variable.

There are several Boolean matching types, whose detailed definitions can be found in [13, 14, 20]. In this work, we focus on the general *non-exact NP equivalence*, defined as follows. Boolean matching under *non-exact NP equivalence* is to determine whether two Boolean functions are functionally equivalent under input bit permutation, negation, sharing, and constant insertion. Formally, given two multi-output Boolean functions $F : \mathbb{B}^n \to \mathbb{B}^p$ and $G : \mathbb{B}^m \to \mathbb{B}^p$ over variables $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$, respectively, assuming $n \leq m$. Let $\xi : (X \to L_X)^n$, where L_X is the set of literals of variables X, called the *negation function*, consist of $\xi_i : \{x_i\} \to \{x_i, \neg x_i\}$ for $i = 1, \ldots, n$. Let $\pi : (L_X \cup \{\perp\} \to L_X \cup \{0, 1\})^m$, called the *"loose permutation" function*, consist of $\pi_i : L_X \cup \{\perp\} \to L_X \cup \{0, 1\}$ for $i = 1, \ldots, m$, where $\perp$ denotes null (to make the composition $\pi \circ \xi$ consistent in the dimension). The objective of Boolean matching is to find a matching solution $\pi \circ \xi$ such that $F(X) = G(\pi \circ \xi(X))$.

2.2 Polynomial Rewriting

Let a bit-vector $W = (w_{\ell-1}, \ldots, w_1, w_0)$ with ℓ variables be associated with some arithmetic data, called *word*, where w_i denotes the i^{th} bit of W with w_0 being the least significant bit. Let $\mathcal{W} = (W_{n-1}, \ldots, W_1, W_0)$ denotes a vector of n words. The number of words in $\mathcal{W}$ is denoted as $|\mathcal{W}|$. The number of bits in a word W is denoted as $|W|$. In the sequel, the capital letter W denotes a word, and the capital letters X or Y denote a word at the primary input of a circuit. The calligraphic font letter $\mathcal{X} = (X_{n-1}, \ldots, X_1, X_0)$ or $\mathcal{Y} = (Y_{n-1}, \ldots, Y_1, Y_0)$ denote a vector of primary input words of a netlist.

A *pseudo-Boolean polynomial*, i.e., a polynomial of Boolean variables, can be written as

$$c_0 M_0 + c_1 M_1 + c_2 M_2 + \cdots + c_k M_k \,,$$

for every coefficient $c_i \in \mathbb{Z}$ and monomial M_i being a product of Boolean variables. With each signal of a netlist being associated with a Boolean variable, the unsigned integer value of a bit-vector $W = (w_{\ell-1}, \ldots, 0)$ can be represented by a pseudo-Boolean polynomial as

$$\sum_{i=0}^{\ell-1} 2^i w_i \,, \tag{1}$$

and the signed integer in 2's complement as

$$-2^{\ell-1} w_{\ell-1} + \sum_{i=0}^{\ell-2} 2^i w_i \,. \tag{2}$$

Given a circuit, from the pseudo-Boolean polynomial of some primary output (PO) bit-vector in the form of Eq. (1), the process of *polynomial rewriting* iteratively replaces each variable in the polynomial with its input variables using the following algebraic models of Boolean connectives:

$$\begin{aligned} \neg w_0 &= 1 - w_0 \\ w_0 \wedge w_1 &= w_0 w_1 \\ w_0 \vee w_1 &= w_0 + w_1 - w_0 w_1 \\ w_0 \oplus w_1 &= w_0 + w_1 - 2 w_0 w_1 \end{aligned} \tag{3}$$

The resultant pseudo-Boolean polynomial describes the relationship between the primary output bit-vector and its input signals. Word-level functions can be extracted by analyzing such pseudo-Boolean polynomials, see, e.g., [9].

3 PROBLEM FORMULATION

We define the elevated Boolean matching problem as follows.

Problem 1 (Elevated Boolean matching). *Let H be a set of arithmetic operations. Given two gate-level netlists N_1 and N_2 whose functions are $F(\mathcal{X})$ and $G(\mathcal{Y})$, respectively, let G and F involve arithmetic operations in H among other Boolean connectives. The elevated Boolean matching problem asks to find a matching solution $\Psi \circ \Delta \circ \Omega$ such that*

$$F(\mathcal{X}) = G(\Psi \circ \Delta \circ \Omega(\mathcal{X})) \,, \tag{4}$$

*where functions Ω, Δ, and Ψ are defined as follows. The Ω-*transform *is the function $\Omega : \mathcal{X} \to \mathcal{X}'$, which consists of $\Omega_i(X_i) = \Pi_i \circ \Xi_i(X_i)$, for $i = 1, \ldots, |\mathcal{X}|$, where each $\Pi_i : (L_{X_i} \cup \{\perp\} \to L_{X_i} \cup \{0, 1\})^{|X_i|}$ and $\Xi_i : (X_i \to L_{X_i})^{|X_i|}$, similar to the π and ξ functions defined in Section 2.1. Letting $-\mathcal{X}'$ denote the set $\{-X' \mid X' \in \mathcal{X}'\}$, where $-X'$ is the integer word negation of X', the Δ-*transform *is a* word negation function $\Delta : \mathcal{X}' \to \mathcal{X}' \cup -\mathcal{X}'$, *which consists of $\Delta_i(X'_i) = X'_i$ or $-X'_i$, for $i = 1, \ldots, |\mathcal{X}|$. The Ψ-*transform *is a* word permutation function $\Psi : \mathcal{X}' \to \mathcal{X}'$, *which forms a bijection and induces a permutation over $\mathcal{X}'$.*

Algorithm 1 Elevated Boolean Matching (basic)

Require: two netlists N_1, N_2, the word information $\mathcal{X}$, $\mathcal{Y}$
1: $B \leftarrow$ Polynomial_Rewriting(N_1)
2: $G(\mathcal{Y}) \leftarrow$ WolFEx(N_2)
3: $T \leftarrow \emptyset$
4: **do**
5: $G'(\Omega(\mathcal{X})) \leftarrow$ Function_Extraction($B, G(\mathcal{Y}), T$)
6: **if** Ω is $\emptyset$ **then**
7: **return** $\emptyset$
8: **end if**
9: $T \leftarrow T \cup G'(\Omega(\mathcal{X}))$
10: $(\Delta, \Psi) \leftarrow$ Word_Matching($G(\mathcal{Y}), G'(\Omega(\mathcal{X}))$)
11: **while** (Δ, Ψ) is $\emptyset$
12: **return** (Δ, Ψ, Ω)

In this work, we assume $H = \{+, -, \times\}$. We note that functions F and G may have control inputs, which are considered single-bit words.

4 METHODS

Algorithm 1 shows the proposed elevated Boolean matching algorithm. N_1, N_2 are two gate-level netlists and their corresponding word information are $\mathcal{X}$ and $\mathcal{Y}$. In the following, we call N_1 the *target netlist* and N_2 the *base netlist*. *WolFEx* is a process that can get the function of an arithmetic netlist, and the details of the process are introduced in [9]. *Function extraction* is a proposed process that can extract the word-level function from a pseudo-Boolean polynomial under the Ω-transform defined in Section 3, and the details will be presented in Section 4.2. *Word matching* is another proposed process that can match the word-level function; the details are explained in Section 4.3.

First, polynomial rewriting is applied to the target netlist N_1 to get its pseudo-Boolean polynomial B. Then, the WolFEx is performed on N_2 to get the word-level function $G(\mathcal{Y})$ of the base netlist N_2. Next, polynomial rewriting is applied to the target netlist N_1 to get its pseudo-Boolean polynomial B. Next, perform function extraction on B to get the word-level function $G'(\Omega(X))$ of the target netlist N_1. Note that we will prevent finding any duplicate $G'(\Omega(X))$ from the previous solutions T. Here, G' and G exhibits functional correspondence. The detail will be presented in Section 4.1. At the end, the word matching is performed between $G(\mathcal{Y})$ and $G'(\Omega(\mathcal{X}))$. If a valid word matching exists, a solution is found. However, since $G'(\Omega(\mathcal{X}))$ is not unique, the procedure needs to repeat function extraction and word matching until a solution is found or function extraction fails, at which point the procedure terminates.

Before going into detail, we illustrate the proposed flow with a simple case.

Example 4.1. Consider matching the target netlist N_1 and the base netlist N_2. Both netlists contain three primary input (PI) words and one primary output (PO) word. In netlist N_1, the PI words X and Z are 2-bit words, while Y is a 3-bit word. In netlist N_2, the PI words P and R are 2-bit integers, and Q is a 3-bit integer.

First, polynomial rewriting was applied to the target netlist N_1. The pseudo-Boolean polynomial B of the target netlist N_1 is given by

$$\begin{aligned} B &= \text{Polynomial\_Rewriting}(N_1) \\ &= -2x_1y_2 + x_1 - 2y_2 + z_0 + 2z_1 - 1, \end{aligned}$$

where x_i, y_i, and z_i are the i^{th} bit of words X, Y, and Z, respectively. Next, perform WolFEx on the base netlist N_2 to get its word-level function $G(P, Q, R)$.

$$\begin{aligned} G(P, Q, R) &= \text{WolFEx}(N_2) \\ &= PQ + R \end{aligned}$$

The function $G(P, Q, R) = PQ + R$ with the word $P = (-2p_1 + p_0)$, the word $Q = (-4q_2 + 2q_1 + q_0)$, and the word $R = (-2r_1 + r_0)$.

The same technique (WolFEx) cannot be applied to the target netlist N_1 since the input bits of target netlist N_1 have been flipped, inserted the constant, and duplicated. Therefore, the proposed process function extraction comes into play. According to the pseudo-Boolean polynomial B of the target netlist N_1. The word-level function of N_1 can be speculated as

$$G'_U = u_0X'Y' + u_1X' + u_2Y' + u_3Z' + u_4, \tag{5}$$

where $u_0, u_1, \ldots, u_4$ are some coefficients, and X', Y', Z' are three Ω-transformed words ($(X', Y', Z') = \Omega(X, Y, Z)$). We call G'_U a speculated parametrized word-level function. When the coefficient U is decided, G'_U would reduce to G' (without undecided coefficients). In Section 4.1, we show that G' and G exhibits functional correspondence (G' and G is the same up to input word permutation and coefficient sign change), i.e., two third of u_1, u_2, u_3 is 0, and u_4 is 0. After the function extraction in Section 4.2, the $G'(X', Y', Z')$ can be obtained

$$\begin{aligned} B &= -2x_1y_2 + x_1 - 2y_2 + z_0 + 2z_1 - 1 \\ &= -(-2 + (1 - x_1))(-4y_2 + 2y_2 + 1) + (-2(1 - z_1) + z_0) \\ &= -X'Y' + Z' \\ &= G'(X', Y', Z') \end{aligned},$$

and $u_0 = -1, u_1 = 0, u_2 = 0, u_3 = 1, u_4 = 0$ in G'_U are also be decided. The function $G'(X', Y', Z') = -X'Y' + Z'$ with $X' = (-2+(1-x_1))$, $Y' = (-4y_2+2y_2+1)$, and $Z' = (-2(1-z_1)+z_0)$ is obtained.

Then, word matching is performed on $G(P, Q, R) = PQ + R$ and $G'(X', Y', Z') = -X'Y' + Z'$ to get the word matching. The word matching solution is given: P is matched with X', Q is matched with Y', Z is matched with Z', and either P or Q is negated.

Finally, the matching solution can be obtained.

$$\begin{aligned} P = (-2p_1 + p_0) \quad &\leftrightarrow \quad X' = (-2 + (1 - x_1)) \\ Q = (-4q_2 + 2q_1 + q_0) \quad &\leftrightarrow \quad Y' = (-4y_2 + 2y_2 + 1) \\ R = (-2r_1 + r_0) \quad &\leftrightarrow \quad Z' = (-2(1 - z_1) + z_0) \end{aligned}$$

That is $p_1 = 1$, $p_0 = \neg x_1$, $q_2 = y_2$, $q_1 = y_2$, $q_0 = 1$, $r_1 = \neg z_1$, $r_0 = z_0$, and an addition word negation is applied on the word P.

4.1 Functional Correspondence between Target and Base Netlists

Given two functions $F(\mathcal{X})$ and $G(\mathcal{Y})$, recall that a matching solution $\Psi \circ \Delta \circ \Omega$ must satisfies

$$F(\mathcal{X}) = G(\Psi \circ \Delta \circ \Omega(\mathcal{X})) = G \circ \Psi \circ \Delta \circ \Omega(\mathcal{X}) \quad (6)$$

Let $\mathcal{X}' = \Omega(\mathcal{X})$, and $G' = G \circ \Psi \circ \Delta$. Then, Eq. (6) becomes

$$F(\mathcal{X}) = (G \circ \Psi \circ \Delta)(\Omega(\mathcal{X})) = G'(\mathcal{X}') \quad (7)$$

Then, if there exists a function G' with the Ω-transformed words $\mathcal{X}'$ as its inputs such that Eq. (7) holds, a matching solution $\Psi \circ \Delta \circ \Omega$ is found. The Eq. (7) shows the connection between F and G with the intermediate function G'. The following shows the relation between G' and G and some of their properties.

A *word-level polynomial* (polynomial for short) with output bit-vector of size m is defined as follows. Let $\mathcal{X} = (X_0, X_1, \ldots, X_{n-1})$ be a vector of n input words. A monomial

$$\mathcal{X}^{\alpha} = X_0^{\alpha_0} \ldots X_{n-1}^{\alpha_{n-1}} \quad (8)$$

is the power product of these n input words $(X_0, X_1, \ldots, X_{n-1})$, where α_i is the exponent of X_i, and $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ is the vector of exponents. An polynomial P of input words $\mathcal{X}$, with the coefficient in $\mathbb{Z}_{2^m}$, is a finite linear combination of monomials

$$P = \sum_i c_i \mathcal{X}^{\alpha_i}, \quad (9)$$

where $c_i \in \mathbb{Z}_{2^m}$ is coefficient.

In Section 3, G is supposed to be some arithmetic function composed of the operation in $H = \{+, -, \times\}$; that is, G is a polynomial.

Claim 1. *$G' = G \circ \Delta \circ \Psi$ is a polynomial, and G' is the same as G up to word permutation and coefficient sign change.*

Proof. First, consider the Ψ-transform. Let $P_\Psi = P \circ \Psi$, and Ψ is a permutation over $\mathcal{X}$, i.e., it permutes the words in every monomial (Eq. (8)). That is P_Ψ is the same as P except for the words permutation in every monomial, and P_Ψ is thus a polynomial. Second, consider the Δ-transform. Let $P_\Delta = P \circ \Delta$. It is easy to find that P_Δ is also a polynomial since there are only coefficient sign changes, i.e., P_Δ and P only differ in some sign of c_i in Eq. (9), and P_Δ is thus also a polynomial. Combining the Ψ-transform and Δ-transform to form $G' = G \circ \Delta \circ \Psi$ results in a polynomial that is the same as G up to word permutation and coefficient sign change. □

Claim 1 is denoted by $G \approx G'$ for simplicity. We say that G and G' exhibit functional correspondence if they satisfy Claim 1.

In conclusion, if there exists a matching solution between $F(\mathcal{X})$ and $G(\mathcal{Y})$, $F(\mathcal{X})$ can be represented by some polynomial G' with the Ω-transformed words $\mathcal{X}'$ as inputs, i.e., $F(\mathcal{X}) = G'(\mathcal{X}')$, and G and G' exhibits functional correspondence ($G \approx G'$).

4.2 SMT-based Word Level Function Extraction

Let N be a gate-level netlist, and B is its pseudo-Boolean polynomial whose function is $F(\mathcal{X})$, where $\mathcal{X}$ is the primary input words without the specified bit-order, and the size of output word of N is m. Then, given N, the objective is to extract the word-level function G' with the Ω-transformed words $\mathcal{X}' = \Omega(\mathcal{X})$ as its input words that is functionally equivalent to N, i.e. $F(\mathcal{X}) = G'(\mathcal{X}')$. The following four steps describe the function extraction flow.

(1) Speculating a word-level function $G'(\mathcal{X}')$ with parametrized coefficient ($U = \{u_0, \ldots\}$) for its monomials, denoted as $G'_U(\mathcal{X}')$. The function $G'_U(\mathcal{X}')$ is called a speculated parametrized word-level function, and it can reduce to G' with the given coefficient U.
(2) Encoding the Ω-transformed words $\mathcal{X}' = \Omega(\mathcal{X})$ with SMT constraints, and expand the speculated parametrized word-level function G'_U into the pseudo-Boolean polynomial B'.
(3) Asserting that the pseudo-Boolean polynomial B' generated by G'_U is functionally equivalent to the pseudo-Boolean polynomial B of the netlist N using SMT constraints.
(4) Solving the encoded SMT constraints to obtain the coefficient U of G'_U and Ω-transform. With the coefficient U, G' can be decided.

First, given a pseudo-Boolean polynomial B, which represents the value of the output word in terms of input bits.

$$B = \sum_i d_i M_i,$$

where $d_i \in \mathbb{Z}_{2^m}$ is the coefficient and M_i is the product of a subset of input bits $\{x_{0,0}, \ldots, x_{|\mathcal{X}|-1,\ell_{|\mathcal{X}|-1}-1}\}$, and $x_{i,j}$ denotes the j^{th}-bit of the word X_i.

Let $N(M_i, k)$ be the number of bits in word X'_k within M_i. For each $d_i M_i$, all monomial divisor $\mathcal{X}'^{\alpha}$ of $\prod_{k=0}^{|\mathcal{X}'|-1} X_k'^{N(M_i,k)}$ are required in G'_U. Then, given a pseudo-Boolean polynomial B, the speculated parametrized word-level function G'_U can be speculated as a polynomial

$$G'_U = \sum_j u_j \mathcal{X}'^{\alpha_j}, \quad (10)$$

where $u_j \in \mathbb{Z}_{2^m}$ is the coefficient and $\mathcal{X}'^{\alpha_j}$ is a monomial.

Second, let X' be a Ω-transformed word with ℓ bits from $X = (x_0, x_1, \ldots, x_{\ell-1})$.

$$X' = \hat{c} + c_0 x_0 + c_1 x_1 + \cdots + c_{\ell-1} x_{\ell-1} \quad (11)$$

where $c_i \in \mathbb{Z}_{2^m}$ is the coefficient of x_i and $\hat{c} \in \mathbb{Z}_{2^m}$ is a constant coefficient.

The Ω-transform performs bit-level transformation $\Pi \circ \Xi$ within each word. The bit-level transformation $\Pi \circ \Xi$ includes bit permutation, negation, sharing, and constant insertion. Moreover, the sign of each word needs to be considered. To encode the Ω-transform, four types of variables are declared to represent bit permutation, negation, sharing, constant insertion, and the sign of word X'.

(1) For each variable x_i, the variable $\beta_i \in \{0,1\}$ represents whether x_i is negated.
(2) For each variable x_i, declare ℓ variables $\gamma_{i,1}, \ldots, \gamma_{i,\ell-1}$, where $\gamma_{i,j} = 1$ represents that x_i is the j^{th} bit in X'.
(3) Declare ℓ variables $\hat{\gamma}_0, \hat{\gamma}_1, \ldots, \hat{\gamma}_{\ell-1}$ where $\hat{\gamma}_i \in \{0,1\}$ represents whether the i^{th} bit of the X' is constant 1.
(4) Declare a variable $\sigma \in \{0,1\}$ that represents whether the transformed word X' is signed or unsigned.

A valid transformed word X' should meet the following SMT constraints. Let p_i be an intermediate variable of x_i.

$$p_i = \sum_{j=0}^{\ell-2} 2^j \gamma_{i,j} + (-1)^{\sigma} 2^{\ell-1} \gamma_{i,l-1}, \text{ where } 0 \le i < \ell \tag{12}$$

$$c_i = (1-\beta_i) \cdot p_i - \beta_i \cdot p_i, \text{ where } 0 \le i < \ell \tag{13}$$

$$\hat{c} = (\sum_{i=1}^{\ell-1} (\beta_i \cdot p_i)) + (\sum_{j=0}^{\ell-2} 2^j \hat{\gamma}_j + (-1)^{\sigma} 2^{\ell-1} \hat{\gamma}_{\ell-1}) \tag{14}$$

$$\textsc{At-Most-One}(\{\hat{\gamma}_j, \gamma_{0,j}, \gamma_{1,j}, \ldots, \gamma_{\ell-1,j}\}), \text{ where } 0 \le j < \ell \tag{15}$$

Eq. (12) encodes the bit permutation, sharing, and sign of word in the 2's complement encoding system. Eq. (13) encodes the bit negation. Eq. (14) encodes the constant insertion. Eq. (15) ensures each bit can only be selected once to make X' a valid integer word. Now, consider multi-word $\mathcal{X}' = \{X'_0, X'_1, \ldots\}$, let $x_{k,i}, c_{k,i}, \hat{c}_k, \beta_{k,i}$ and ℓ_k denotes the $x_i, c_i, \hat{c}, \beta_i$ and ℓ of the word X'_k respectively.

With the structural information from the given netlist N, the additional constraints can be added to prune the solution space of Ω-transform using the support information of each input bit. Assuming that the transitive output signals of x_i contain the transitive output signals of x_j, then x_i must be used in a lower bit than x_j. The constraint can be written as

$$\bigwedge_{u=0}^{m-2} \bigwedge_{v=u}^{m-1} ((\beta_{j,u}) \rightarrow (\neg \beta_{i,v})) \tag{16}$$

Moreover, we can analyze the containment relationship of each bit in a word to get the range of each bit it belongs to. Therefore, the solution space of Ω-transform can be pruned further.

Example 4.2. Consider a 4-bit word $W = \{w_3, w_2, w_1, w_0\}$. w_0 contains w_1, w_2, w_3. w_1 contains w_2, w_3. w_2 and w_3 does not contain any signal. Then, the relation between w_0 to w_3 should be $\{w_0\} < \{w_1\} < \{w_2, w_3\}$. The range of each bit can be obtained with this relation analysis. There are three sets of signals, $\{w_0\}$, $\{w_1\}$, and $\{w_2, w_3\}$. Each set occupied at least 1 bit and the size of set bits at most. In this case, w_0 is 0^{th} or 1^{st} bit. w_1 is 1^{st} or 2^{nd} bit. w_2, w_3 are 2^{nd} or 3^{rd} bit.

Plugging the Ω-transformed word X' (Eq. (11)) into the speculated parametrized word-level function G'_U (Eq. (10)), the pseudo-Boolean polynomial of G'_U can be computed and denote as B'

$$\begin{aligned} B' &= \sum_i u_i \mathcal{X}'^{\alpha_i} \\ &= \sum_j d'_j M'_j, \end{aligned} \tag{17}$$

where d'_j should be expressed in terms of u_i, $c_{j,k}$, and $\hat{c}_l$, and M_j is the product of a subset of input bits $\{x_{0,0}, \ldots, x_{|\mathcal{X}|-1,\ell_{|\mathcal{X}|-1}-1}\}$, and $x_{i,j}$ denotes the j^{th}-bit of the word X_i.

Third, the following SMT constraints are added to make B functionally equivalent to B':

$$\begin{cases} d_j \equiv d'_i \pmod{2^m}, \text{ if } M'_i = M_j \\ 0 \equiv d'_i \pmod{2^m}, \text{ if } M'_i \text{does not occur in } B. \end{cases} \tag{18}$$

Finally, by solving the above SMT constraints with SMT solver, the objective G' and Ω can be obtained.

Example 4.3. Recall the illustrative Example 4.1, X', Y', Z' can be represented in Eq. (11)

$$\begin{cases} X' = \hat{c}_{X'} + c_{X',0} x_0 + c_{X',1} x_1 \\ Y' = \hat{c}_{Y'} + c_{Y',0} y_0 + c_{Y',1} y_1 + c_{Y',2} y_2 \\ Z' = \hat{c}_{Z'} + c_{Z',0} x_0 + c_{Z',1} x_1 \end{cases} \tag{19}$$

Plugging Eq. (19) into the speculated function $G'_U = u_0 X'Y' + u_1 X' + u_2 Y' + u_3 Z' + u_4$ gives the pseudo-Boolean polynomial B' in Eq. (17). Comparing its coefficients to that of the pseudo-Boolean polynomial $B_1 = -2x_1y_2 + x_1 - 2y_2 + z_0 + 2z_1 - 1$ of the target netlist N_1, we can add corresponding constraints to assert that B' and B are functionally equivalent. For instance, comparing the coefficient of the monomial $x_1 y_2$ gives the constraint

$$-2 \equiv u_0 c_{X',1} c_{Y',2} \pmod{2^4}.$$

The function can be extracted by solving the above constraints.

$$\begin{cases} u_0 = -1, u_1 = 0, u_2 = 0, u_3 = 1, u_4 = 0 \\ \hat{c}_{X'} = -1, c_{X',0} = 0, c_{X',1} = -1 \\ \hat{c}_{Y'} = 1, c_{Y',0} = 0, c_{Y',1} = 0, c_{Y',2} = -2 \\ \hat{c}_{Z'} = -2, c_{Z',0} = 1, c_{Z',1} = 2 \end{cases}$$

That is, $G'(X', Y', Z') = -X'Y' + Z'$, $X' = (-2 + (1 - x_1))$, $Y' = (-4y_2 + 2y_2 + 1)$, and $Z' = (-2(1 - z_1) + z_0)$.

4.3 SAT-based Word Level Matching

Given two word-level functions $G'(\mathcal{X}')$ and $G(\mathcal{Y})$, and their output bit-vector size is m, the objective is to determine Δ-transform and Ψ-transform such that

$$G'(\mathcal{X}') = G(\Psi \circ \Delta(\mathcal{X}')) = G(\mathcal{Y}) \tag{20}$$

In Section 4.1, the derivation is given that if Eq. (20) holds for some Δ-transform and Ψ-transform, then $G \approx G'$. Assuming G and G' have s terms, and the number of input words is n.

$$G' = \sum_{i=0}^{s-1} u_i \mathcal{X}'^{\mu_i} \tag{21}$$

$$G = \sum_{i=0}^{s-1} v_i \mathcal{Y}^{\kappa_i} \tag{22}$$

where $u_i, v_i \in \mathbb{Z}_{2^m}$ are coefficients, μ_i, κ_i are exponent vector of monomial, and $\mathcal{X}'^{\mu_i}, \mathcal{Y}^{\kappa_i}$ are monomials.

To encode Δ-transform and Ψ-transform, three variables are declared to represent word negation and permutation.

(1) For each X'_i, declare variable $\rho_i \in \{0,1\}$ represents either X'_i is negated.
(2) For each X'_i, declare n variables $\tau_{i,0}, \tau_{i,1}, \ldots, \tau_{i,n-1}$ where $\tau_{i,j} = 1$ represents either X'_i is matched with Y_j.
(3) For each monomial $u_i \mathcal{X}'^{\mu_i}$, declare s variables $\varepsilon_{i,0}, \varepsilon_{i,1}, \ldots, \varepsilon_{i,s-1}$ where $\varepsilon_{i,j} = 1$ represents either $u_i \mathcal{X}'^{\mu_i}$ is matched with $v_j \mathcal{Y}^{\kappa_j}$.

For convenience, $\{X' : X' \in \mathcal{X}'^{\mu_i}\}$ and $\{Y : Y \in \mathcal{Y}^{\kappa_j}\}$ denotes a set of words with non-zero exponent in monomial $\mathcal{X}'^{\mu_i}$ and $\mathcal{Y}^{\kappa_j}$, respectively. The variables $\mu_{i,j}$ and $\kappa_{i,j}$ denotes the exponent of X'_j in monomial $\mathcal{X}'^{\mu_i}$ and $\mathcal{Y}^{\kappa_j}$, respectively. The exponent vector $\hat{\mu_i}$ and $\hat{\kappa_j}$ be the sorted version of μ_i and κ_j. A feasible word matching should meet the following SAT constraints.

$$\textsc{At-Least-One}(\{\varepsilon_{0,i}, \varepsilon_{1,i}, \ldots, \varepsilon_{s-1,i}\}) \text{ ,where } 0 \le i < s \quad (23)$$

$$\textsc{At-Most-One}(\{\varepsilon_{0,i}, \varepsilon_{1,i}, \ldots, \varepsilon_{s-1,i}\}) \text{ ,where } 0 \le i < s \quad (24)$$

$$\textsc{At-Most-One}(\{\tau_{0,i}, \tau_{1,i}, \ldots, \tau_{n-1,i}\}) \text{ ,where } 0 \le i < n \quad (25)$$

$$\varepsilon_{i,j} \rightarrow \bigwedge_{X'_u \in \mathcal{X}'^{\mu_i}} \left(\bigvee_{Y_v \in \mathcal{Y}^{\kappa_j}, \mu_{i,u} = \kappa_{j,v}} \tau_{u,v} \right) \quad (26)$$

$$\neg \tau_{i,j}, \text{ if } |X'_i| \neq |Y_j|, 0 \le i, j < n \quad (27)$$

$$\varepsilon_{i,j} \rightarrow \left(u_i = \text{ITE}\left(\bigoplus_{X'_k \in \mathcal{X}'^{\mu_i}} \left(\bigoplus_{\ell=0}^{\mu_{i,k}} \rho_k \right), -v_j, v_j \right) \right) \quad (28)$$

$$\neg \varepsilon_{i,j}, \text{ if } \hat{\mu_i} \neq \hat{\kappa_j}, 0 \le i, j < s \quad (29)$$

Eq. (23) and Eq. (24) ensure monomials in G' and G has one-to-one and onto mapping. Eq. (25) and Eq. (26) ensure words in $\mathcal{X}'$ and $\mathcal{Y}$ has one-to-one and onto mapping. Eq. (27) ensures only the words with the same bit-vector size can be mapped. Eq. (28) ensures the consistency of coefficients. Eq. (29) ensures the consistency of exponents.

Solving the above SAT constraint with SAT solver; the objective Δ-transform and Ψ-transform can be obtained.

5 IMPLEMENTATION DISCUSSION

Recall the overall elevated Boolean matching flow in Algorithm 1. Given two netlists N_1 and N_2 whose function is $F(\mathcal{X})$ and $G(\mathcal{Y})$, and the size of the output word is m. The objective is to obtain a matching solution $\Psi \circ \Delta \circ \Omega$ such that $F(\mathcal{X}) = G(\Psi \circ \Delta \circ \Omega(\mathcal{X}))$.

The pseudo-Boolean polynomial B of N_2 is derived first. Next, the base function $G(\mathcal{Y})$ of N_2 is obtained. In function extraction, an speculated function $G'(\mathcal{X}')$ is extracted such that $F(\mathcal{X}) = G'(\mathcal{X}')$. Finally, performs word matching between G and G' to find a matching solution. However, G' and $\mathcal{X}'$ is not unique. Therefore, the Algorithm 1 needs to repeat function extraction and word matching until a matching solution is found.

The issue in Algorithm 1 is that the solution space of function extraction is too large. In addition, repeating function extraction and word matching until a matching solution is found takes time. Therefore, Algorithm 1 is not expected to be efficient and thus not implemented.

A preprocessing technique is introduced to prune the solution space of word matching and function extraction by constraining the speculated parametrized word-level function G'_U; the details are elaborated in Section 5.1. A refined implementation, presented in Algorithm 2, is proposed to address the issue outlined in Algorithm 1 by swapping the word matching and function extraction steps; further details will be provided in Section 5.2.

5.1 Speculated Function Preprocessing

Section 4.1 guarantees that $G \approx G'$. Therefore, the speculated parametrized word-level function G'_U can be constrained by analyzing the possible matching of monomials between G and G'. $G(\mathcal{Y})$ and $G'_U(\mathcal{X}')$ can be expressed as

$$G = \sum_{i=0}^{s} v_i \mathcal{Y}^{\kappa_i} \quad (30)$$

$$G'_U = \sum_{i=0}^{t} u_i \mathcal{X}'^{\mu_i}, \quad (31)$$

where $v_i, u_i \in \mathbb{Z}_{2^m}$ are coefficients, κ_i, μ_i are exponent vector of monomial, and $\mathcal{Y}^{\kappa_i}, \mathcal{X}'^{\kappa_i}$ are monomials. Note that t is greater than or equal to s ($t \ge s$). Since $G \approx G'$, the mapping between each monomial $u_i \mathcal{X}'^{\mu_i}$ in $G'_U(\mathcal{X}')$ and $v_j \mathcal{Y}^{\kappa_j}$ in $G(\mathcal{Y})$ can be constrained. A monomial mapping is valid only when the bit-width and the exponent of words can be mapped one by one.

Let monomial $\hat{\mathcal{X}}'^{\hat{\mu}_i}$ and $\hat{\mathcal{Y}}^{\hat{\kappa}_i}$ be the sorted version of $\mathcal{X}'^{\mu_i}$ and $\mathcal{Y}^{\kappa_i}$, where $\hat{\mu}_i$ and $\hat{\kappa}_i$ is the sorted exponent vector. The words in each $\hat{\mathcal{X}}'^{\hat{\mu}_i}$ or $\hat{\mathcal{Y}}^{\hat{\kappa}_i}$ are sorted according to their exponents. If two words have the same exponent, they are sorted according to their bit width. Let $\lambda_{\hat{\mathcal{X}}'^{\hat{\mu}_i}}$, $\lambda_{\hat{\mathcal{Y}}^{\hat{\kappa}_i}}$ be a vector of bit-width of $\hat{\mathcal{X}}'^{\hat{\mu}_i}$, $\hat{\mathcal{Y}}^{\hat{\kappa}_i}$. The pair $(\lambda_{\hat{\mathcal{X}}'^{\hat{\mu}_i}}, \hat{\mu}_i)$ and $(\lambda_{\hat{\mathcal{Y}}^{\hat{\kappa}_i}}, \hat{\kappa}_i)$ is the signature of the monomial $\hat{\mathcal{X}}'^{\hat{\mu}_i}$ in G'_U and those of the monomial $\hat{\mathcal{Y}}^{\hat{\kappa}_i}$ in G. Partitioning the monomials in G'_U and those in G into some sets $\{S'_0, S'_1, \ldots\}$ and $\{S_0, S_1, \ldots\}$ with respect to the signature. S'_i is said to matched with S_j if their signature is the same. The properties of valid monomial mappings are listed below.

i) If S'_i is matched with S_j, every monomial in S_j should be matched with exactly one monomial in S'_i.
ii) If S'_i cannot matched with any S_j, every monomial's coefficient in S'_i should be 0.

A valid monomial mapping ensures $G \approx G'$ if the above properties are met. With this preprocessing technique, the solution space of word matching is pruned, as impossible matching solutions are prohibited. Additionally, the solution space for function extraction is pruned because we can directly constrain the coefficient U in G'_U, rather than leaving it to be decided by the function extraction process.

Example 5.1. Recall the illustrative Example 4.1, The speculated parametrized word-level function $G'_U(X', Y', Z') = u_0 X'Y' + u_1 X' + u_2 Y' + u_3 Z' + u_4$ and $G(P, Q, R) = PQ + R$. The bit-width $|X'| = |Z'| = 2$, $|Y'| = 3$, $|P| = |R| = 2$, and $|Q| = 3$. Then, the signature of every monomial in G'_U and G

Algorithm 2 Elevated Boolean Matching (refined)

Require: two netlists N_1, N_2, the word information $\mathcal{X}$, $\mathcal{Y}$

$B \leftarrow$ Polynomial_Rewriting(N_1)
$G(\mathcal{Y}) \leftarrow$ WolFEx(N_2)
$G'_U(\Omega(\mathcal{X})) \leftarrow$ Function_Speculation(B_1)
$T \leftarrow \emptyset$
do
  $(\Psi, \Delta) \leftarrow$ Word_Matching($G'_U(\Omega(\mathcal{X})), G(\mathcal{Y}), T$)
  if (Ψ, Δ) is $\emptyset$ **then**
    return $\emptyset$
  end if
  $G' = G \circ \Psi \circ \Delta$
  $\Omega \leftarrow$ Function_Extraction($G'(\Omega(\mathcal{X})), B_1$)
  $T \leftarrow T \cup (\Psi, \Delta)$
while Ω is $\emptyset$
return (Δ, Ψ, Ω)

are

$$u_0X'Y' : ((3,2,2),(1,1,0)) \quad PQ : ((3,2,2),(1,1,0))$$
$$u_1X' : ((2,3,2),(1,0,0)) \quad R : ((2,3,2),(1,0,0))$$
$$u_2Y' : ((3,2,2),(1,0,0))$$
$$u_3Z' : ((2,3,2),(1,0,0))$$
$$u_4 : ((3,2,2),(0,0,0))$$

The partitioned sets are

$$((3,2,2),(1,1,0)) \rightarrow S'_0 : \{u_0X'Y'\} \quad S_0 : \{PQ\}$$
$$((2,3,2),(1,0,0)) \rightarrow S'_1 : \{u_1X', u_3Z'\} \quad S_1 : \{R\}$$
$$((3,2,2),(1,0,0)) \rightarrow S'_2 : \{u_2Y'\}$$
$$((3,2,2),(0,0,0)) \rightarrow S'_3 : \{u_4\}$$

Then, $u_0X'Y'$ is matched with PQ, either u_1X' or u_3Z' is matched with R (either u_1 or u_3 is 0), and u_2Y', u_4 does not matched with any monomial in G ($u_2 = 0, u_4 = 0$). In other words, the parameterized speculated function $G'_U(\mathcal{X}')$ is either $u_0X'Y' + u_1X'$ ($u_0 = \pm 1, u_1 = \pm 1$) or $u_0X'Y' + u_3Z'$ ($u_0 = \pm 1, u_3 = \pm 1$).

5.2 Refined Implementation

The refined implementation performs word matching before function extraction to obtain a candidate coefficient U for G'_U. This allows us to guide the function extraction towards a specific function, instead of conducting an exhaustive search for a feasible function. The Algorithm 2 is the pseudo-code of the refined implementation.

First, the WolFEx is performed on N_2 to get the word-level function $G(\mathcal{Y})$ of the base netlist N_2. Next, polynomial rewriting is applied to the target netlist N_1 to get its pseudo-Boolean polynomial B. With the pseudo-Boolean polynomial B, we can speculate a parametrized word-level function $G'_U(\Omega(\mathcal{X}))$ under the Ω-transform using the same method in Section 4.2 (Eq. (10)).

In Section 5.1, the preprocessing technique for matching G' and G'_U is introduced, and the constraint Eq. (23) in Section 4.3 can be modified. According to the signature of the monomial, the monomials in G'_U and those in G can be partitioned into some sets $\{S'_0, S'_1, \dots\}$ and $\{S_0, S_1, \dots\}$, respectively. The constraint Eq. (23) can be modified with the valid monomial mapping properties mentioned in Section 5.1 as follows:

$$\text{At-Least-One}(\varepsilon_{i_0,j}, \varepsilon_{i_1,j}, \dots, \varepsilon_{i_k,j}) \tag{32}$$

where $i_0, \dots, i_k \in S'_m, j \in S_n$ and the signature of S'_m and those of S_n is the same. Then, perform the word matching, and all SAT constraints in Section 4.3 are used and the modified constraint Eq. (32) is added to get the candidate word matching $\Psi \circ \Delta$. The candidate word-level function G' of the target netlist N_1 can be obtained with the candidate Ψ-transform and Δ-transform. Last, the function extraction is performed on the pseudo-Boolean polynomial B of the target netlist N_1 to extract the candidate target function G'. If a valid extraction exists, a solution is found. Otherwise, record the current word matching Ψ, Δ with the argument T to prevent the next iteration from giving the duplicate assignment. Repeat word matching and function extraction until the matching solution is found or the new word matching doesn't exist; then, the procedure terminates.

Note that the traditional Boolean matching algorithm enumerates all permutations and negations on the input bits, resulting in a time complexity of $O(N!2^N)$, where N is the number of input bits. However, using word-level information, the proposed Algorithm 2 only needs to consider permutations and negations of input words. This significantly reduces N from the number of input bits to the number of input words, resulting in a substantial reduction in complexity. Note that the complexity here refers to the number of SMT or SAT solver calls.

6 EXPERIMENTAL RESULTS

The proposed methods were implemented using C++ and Python. We adopted `Boolector` [16] as the underlying SMT solver. Our preliminary experiments were conducted on a Linux machine with an Intel Core i9-12900 CPU and 128 GB RAM.

The benchmarks for evaluation were taken (as the base netlist N_2) and modified (as the target netlist N_1) from the word-level function extraction circuits of the 2022 ICCAD CAD Contest [7] by the Cadence Design Systems, Inc. The word-level functions involve only $\{+, -, \times\}$ operations. The base netlist N_2 of each case implements a circuit involving integer-valued polynomial functions and some control logic. The target netlist N_1 of each case was generated from the base netlist N_2 with two steps: 1) random bit negation, permutation, sharing, and constant insertion, and 2) random word negation and permutation. The netlists N_1 and N_2 were synthesized into the gate level by `Yosys` [19]. A time-out limit of 3600 seconds was imposed for solving each case.

Table 1 shows the experimental results. Cases test04, test10, and test 11 involve multiple output words, which are separated in different cases in the table. Also, the cases marked by the "*" sign were bitwidth-reduced versions, and so are their corresponding bit widths. The detail of each case is shown in Table 1, where columns "#gate," "#word," "#bit," "#operation," "#change@wl," and "#change@bl" report the

Table 1: Results of elevated Boolean matching on the word-level function extraction benchmarks.

Case	#gate	#word	#bit		#operation		#change@wl	#change@bl			CPU time (s)		
			PI	PO	ADD	MUL	neg	inv	share	const	P+W	SMT	total
test01	56	4	12	4	3	0	0	4	1	4	1.527	0.033	1.56
test02	73	4	12	4	1	1	0	8	2	3	1.85	0.081	1.931
test03	430	4	57	20	3	0	1	25	7	18	3.28	0.192	3.472
test04_1	123	3	32	17	1	0	1	16	6	7	14.00	0.232	14.232
test04_2	249	4	48	17	2	0	1	16	9	7	15.00	0.140	15.140
*test04_3	3175	6	48	*16	4	3	1	26	25	13	296.53	39.664	336.194
*test05	290	5	64	*8	3	3	2	30	14	15	45.52	10.37	55.89
test12	14231	25	264	26	11	11	4	97	44	72	388.257	210.09	598.347
test04_3	6228	6	48	33	4	3	1	26	25	13	-	47.189	-
test05	1714	5	64	34	3	3	2	30	14	15	-	23.192	-
test06	5756	4	96	65	3	2	1	40	18	23	-	28.697	-
test10_1	2334	4	48	32	2	1	1	13	10	11	-	2.397	-
test10_2	2327	4	48	32	2	1	1	18	8	12	-	7.43	-
test11_1	1360	2	8	24	0	2	1	3	2	0	-	2.397	-
test11_2	573	3	16	16	1	1	1	1	5	3	-	0.562	-
test20	4985	26	121	17	13	18	4	54	27	19	-	288.12	-

gate count of N_2, the total number of words, the total number of PI/PO signals, the operations that N_2 used, the number of word-level negation in N_1, and the bit-level negation, sharing, constant insertion in N_1, respectively. The execution time is shown in the last three columns "P+W," "SMT," and "total," which denote the execution time of polynomial rewriting and WolFEx (corresponding to computation of lines 1-3 in Algorithm 1), the SMT solving time (corresponding to lines 4-15 in Algorithm 1), and the total runtime, respectively.

The cases in Table 1 are divided into two parts, separated by a line. The cases in the upper part were solved completely, whereas those in the lower part reached the time-out limit when performing polynomial rewriting on N_1. Because the functionality of N_2 is known, we can still validate the efficiency and correctness of our SMT encoding for the cases in the lower part.

The correctness of all results was validated. The mapping solutions of all cases were written back to the pseudo-Boolean polynomial. The pseudo-Boolean polynomials after matching in all cases coincide with their original pseudo-Boolean polynomials.

In Table 1, we can find that without output bit truncation, the procedure solves 6 out of the 14 circuits. All of the unsolved cases were timed out in the P+W computation. On the other hand, the SMT computation exhibited a relatively efficient solving performance compared to the P+W part.

For the cases where the P+W part timed out, it has been observed that WolFEx [9] can solve them in the context of word-level function extraction but not Boolean matching. It is because, in function extraction, these cases do not require a complete polynomial rewriting but only for the lower few bits. In contrast, in the Boolean matching setting, because every bit may undergo some possible change, a complete polynomial rewriting (for all bits) is necessary in order to obtain a matching solution. Consequently, it resulted in the time-outs.

The performance of polynomial rewriting appeared to be influenced more significantly by the maximum bitwidth of words than the number of input bits or of gates. For instance, test05, despite being much smaller than test12, still timed out during the P+W part. On the other hand, in the SMT solving part, we found that the maximum bitwidth of the input words has a greater impact on the performance than the number of gates. For instance, test12 has more gates than test20, but the runtime of test12 is shorter than test20. This result is due to the larger maximum input-word bitwidth of test20 compared to test12.

Based on the observation mentioned above, it can be concluded that the complete polynomial rewriting process is the most time-consuming and limiting factor. To overcome the bottleneck, one approach could be to perform abstraction by setting some primary inputs and outputs to constants before the polynomial rewriting process. We leave the abstraction and refinement strategy for future exploration.

7 CONCLUSIONS AND FUTURE WORK

We have lifted Boolean matching to the word level and developed a solution engine based on WolFEx [9] and SMT-solving techniques. Experiments on the word-level function extraction benchmarks demonstrated the effectiveness and feasibility of our approach. Our results provide a first step toward functional correspondence checking at the word level for potential synthesis and verification applications.

For future work, we plan to seek ways to circumvent the current bottleneck of polynomial rewriting to handle more complex designs. We also plan to consider non-polynomial arithmetic circuits involving other operation types besides $\{+, -, \times\}$ and beyond integer arithmetic.

REFERENCES

[1] Afshin Abdollahi. 2008. Signature Based Boolean Matching in the Presence of Don't Cares. In *Proceedings of the Design Automation Conference*. 642–647.

[2] Afshin Abdollahi and Massoud Pedram. 2005. A New Canonical Form for Fast Boolean Matching in Logic Synthesis and Verification. In *Proceedings of the Design Automation Conference*. 379–384.

[3] Giovanni Agosta, Francesco Bruschi, Gerardo Pelosi, and Donatella Sciuto. 2007. A Unified Approach to Canonical Form-Based Boolean Matching. In *Proceedings of the Design Automation Conference*. 841–846.

[4] Mohammed Barhoush, Alireza Mahzoon, and Rolf Drechsler. 2021. Polynomial word-level verification of arithmetic circuits. In *In Proceedings of the International Conference on Formal Methods and Models for System Design* (Virtual Event, China) *(MEMOCODE '21)*. 1–9.

[5] Luca Benini and Giovanni De Micheli. 1997. A Survey of Boolean Matching Techniques for Library Binding. *ACM Transactions on Design Automation of Electronic Systems* 2, 3 (1997), 193–226.

[6] Chung-Han Chou, Chih-Jen Hsu, Chi-An Wu, Kuan-Hua Tu, and Kei-Yong Khoo. 2023. Invited Paper: 2023 ICCAD CAD Contest Problem A: Multi-Bit Large-Scale Boolean Matching. In *Proceedings of the International Conference on Computer-Aided Design*. 1–4.

[7] Chung-Han Chou, Chih-Jen Jacky Hsu, Chi-An Rocky Wu, and Kuan-Hua Tu. 2022. 2022 CAD Contest Problem A: Learning Arithmetic Operations from Gate-Level Circuit. In *Proceedings of the International Conference on Computer-Aided Design*. 1–4.

[8] J. Cong and Yean-Yow Hwang. 2006. Boolean Matching for LUT-Based Logic Blocks With Applications to Architecture Evaluation and Technology Mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20, 9 (2006), 1077–1090.

[9] Kuo-Wei Ho, Shao-Ting Chung, Tian-Fu Chen, Yu-Wei Fan, Che Cheng, Cheng-Han Liu, and Jie-Hong R. Jiang. 2023. WolFEx: Word-Level Function Extraction and Simplification from Gate-Level Arithmetic Circuits. In *Proceedings of the International Conference on Computer-Aided Design*. 1–9.

[10] Shao-Lun Huang, Wei-Hsun Lin, and Chung-Yang (Ric) Huang. 2011. Match and Replace: A Functional ECO Engine for Multi-Error Circuit Rectification. In *Proceedings of the International Conference on Computer-Aided Design*. 383–388.

[11] Hadi Katebi and Igor Markov. 2010. Large-scale Boolean matching. In *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 227–247.

[12] Smita Krishnaswamy, Haoxing Ren, Nilesh Modi, and Ruchir Puri. 2009. DeltaSyn: An Efficient Logic Difference Optimizer for ECO Synthesis. In *Proceedings of the International Conference on Computer-Aided Design*. 789–796.

[13] Chih-Fan Lai, Jie-Hong R. Jiang, and Kuo-Hua Wang. 2010. Boolean Matching of Function Vectors with Strengthened Learning. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD '10)*. 596–601.

[14] Chih-Fan Lai, Jie-Hong R. Jiang, and Kuo-Hua Wang. 2010. BooM: A Decision Procedure for Boolean Matching with Abstraction and Dynamic Learning. In *Proceedings of the Design Automation Conference*. 499–504.

[15] Janett Mohnke, Paul Molitor, and Sharad Malik. 2001. Application of BDDs in Boolean Matching Techniques for Formal Logic Combinational Verification. *International Journal on Software Tools for Technology Transfer* 3 (2001), 207–216.

[16] Aina Niemetz, Mathias Preiner, and Armin Biere. 2014. Boolector 2.0. *J. Satisf. Boolean Model. Comput.* 9, 1 (2014), 53–58. https://doi.org/10.3233/sat190101

[17] Chak-Wa Pui, Peishan Tu, Haocheng Li, Gengjie Chen, and Evangeline F. Y. Young. 2018. A Two-Step Search Engine for Large Scale Boolean Matching under NP3 Equivalence. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 592–598.

[18] Xing Wei, Yi Diao, Tak-Kei Lam, and Yu-Liang Wu. 2015. A universal macro block mapping scheme for arithmetic circuits. In *Design, Automation & Test in Europe Conference & Exhibition*. 1629–1634.

[19] Claire Wolf. [n. d.]. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys/.

[20] Chi-An Wu, Chih-Jen Hsu, and Inc Cadence Taiwan. 2016. NP3: Non-exact Projective NPNP Boolean Matching. https://www.iccad-contest.org/2016/Problem_B/default.html

[21] Chaofan Yu, Lingli Wang, Chun Zhang, Yu Hu, and Lei He. 2013. Fast Filter-Based Boolean Matchers. *IEEE Embedded Systems Letters* 5, 4 (2013), 65–68.