Master's Thesis

Advance Methods for Formal Divider Verification Based on Symbolic Computer Algebra

Akshay Tukaram Rao

Examiners: Prof. Dr. Christoph Scholl
Prof. Dr. Armin Biere

Advisers: Alexander Konrad, M.Sc

University of Freiburg
Faculty of Engineering
Department of Computer Science
Operating Systems Chair

September 23th, 2024

Writing Period
21. 03. 2024 – 23. 09. 2024

Examiner
Prof. Dr. Christoph Scholl

Second Examiner
Prof. Dr. Armin Biere

Advisers
Alexander Konrad, M.Sc

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore,
I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg , 23 September 2024

Place, Date

Signature

Abstract

Recent methods based on Symbolic Computer Algebra (SCA) have made good progress in verifying divider circuits. The current thesis focuses on analyzing the verification of an optimized reduced restoring divider using the SCA-based method. This work illustrates the challenges of verifying the reduced restoring divider and extends the existing optimization technique-**Don't Care Optimization** to **Delayed Backward Rewriting**. Although this optimization technique aims to verify reduced restoring dividers efficiently to some extent, it does not fully address the scalability issues for larger divider circuits. The thesis proposes an overview of a potential solution that could verify larger divider circuits without causing any memory explosion.

Contents

List of Figures

List of Tables

1 Introduction

Integrated circuits that perform arithmetic operations have become integral to computationally heavy applications such as image processing, signal processing and cryptography. Formal verification of these circuits is critical in detecting or preventing design errors that appear when translating a specification into the final integrated circuit. To meet the surge in requirements for high computational speed, low area, and low power, designers have introduced various architectures and optimizations in the design and implementation, which pose challenges to formal verification.

The multiplier and divider circuits are the most difficult to verify among the arithmetic circuits. Verification methods like binary decision diagrams (BDDs) and binary moment diagrams (BMDs) fail to model large divider and multiplier circuits as they grow exponentially with the circuit's size [1]. Similarly, SAT-based methods do not scale well because they require bit-blasting to model equivalence checking [1]. Other methods, such as Theorem Provers, use mathematical reasoning, rewriting rules, and complex formulas to verify arithmetic circuits. This method is often inconclusive because the choice and order of the application of the rules determine its effectiveness [2].

Recently, the introduction of Symbolic Computer Algebra (SCA) in formal verification has given rise to efficient methods for verifying the correctness of the arithmetic circuits. SCA-based methods use polynomials to represent the circuit specification(F) and hardware implementation (G). Such SCA-based methods translate the verification problem into proving the obligation that hardware implementation (G) satisfies circuit specification(F). SCA-based methods also extend to verify large multiplier and divider circuits.

The underlying principle of a symbolic computer algebra-based method to verify a divider circuit involves replacing the output variables in the specification polynomial($Fspec$): $Q \cdot D + R - R^{(0)}$ with their corresponding gate polynomials such that the specification polynomial reduces to zero. Additionally, it is crucial to verify the condition $0 \leq R < D$ to confirm that the implementation satisfies the specification of a divider. Here, the output variables Q and R are quotient and remainder, respectively, and the input variables D, $R^{(0)}$ are divisor and dividend, respectively.

There have been several algorithms, such as restoring division, non-restoring division, division by constant, and SRT division, that realize division operations in practical applications. The present work focuses on verifying the unsigned integer reduced restoring divider using the symbolic computer algebra-based method. The major setback of this method is the size of the intermediate polynomials, which grow exponentially during the replacement of the output variables Q and R in terms of the input variables D and $R^{(0)}$.

The presented thesis focuses on analysing the challenges encountered while verifying the reduced restoring divider circuit, mainly due to its design. The current thesis is structured as follows: Section 2 discusses previous work and recent developments involved in verifying divider circuits. Section 3 presents prerequisites and fundamental principles related to the current work, which include a few important concepts such as backward rewriting, restoring division algorithm and the circuit structure of an optimized reduced restoring divider. Section 4 demonstrates the preliminary analysis of a carry-ripple adder and multiplexer that are a part of the restoring divider circuit discussed in the current work. Section 5 forms the core of this thesis, which provides the complete analysis of verifying the restoring divider circuit. It illustrates the shortcomings of the optimization technique presented in [3] that strives to verify the divider circuits efficiently. Section 5 also presents a new optimization technique: *Delayed Backward Rewriting*, which up to some extent, addresses the challenges present in the verification of the reduced restoring divider circuit. Section 6 illustrates the principle involved in verifying the condition $0 \leq R < D$. Section 8 presents the implementation of two optimization methods involved in verifying the reduced restoring divider. Section 9 presents the experimental evaluation of the optimization techniques applied during the verification of the reduced restoring divider. Section 10 summarizes the analysis of the reduced restoring divider and gives an outlook on open questions that provide opportunities for further work.

2 Previous Work

Although formal verification has made substantial progress in verifying large and complex multiplier circuits, it still faces significant challenges when applied to divider circuits. According to [4], verification of SRT dividers use Decision Diagrams and word-level models; however, they either restrict themselves to proving the correctness of a single divider stage or consider an abstract and clean sequential divider without gate-level optimizations and yet fail to deduce a fully automated verification method conducive for divider circuits.

Hamaguchi et al. [5] conceived the initial idea of a fully automated method to verify divider circuits using *BMD to represent $Q \cdot D + R$ and employ backward construction to replace the bits of Q and R incrementally with *BMDs representing the gates of the divider. The goal is to obtain a *BMD representation for the dividend $R^{(0)}$, proving the divider circuit's correctness. Despite being a promising method, this approach has not been successful in practice, as experimental results have shown exponential blow-ups of *BMDs during backward construction [4].

The approach presented in [6] uses SCA-based methods to automate the verification of divider circuits. However, it is confined to division by constant and fails to address general dividers without being hampered by memory explosion. The work in [2] also employs the SCA-based method with *layered* rewriting to prove the correctness of general divider circuits. The *layered* technique rewrites each row or layer of the divider, which represents the subtraction of the shifted dividend with the partial remainder to obtain the quotient bit. This method leverages the fact that synthesis tools do not optimize the logic between two adjacent rows, thereby preserving the partial remainder signals during synthesis [7].

Additionally, layered rewriting also offers the advantage of identifying and locating errors in the circuit by inspecting the local signature obtained after rewriting a specific layer. Errors or bugs exist in the particular layer if the local signature is not equivalent to the polynomial representing the partial remainder [2]. Nonetheless, this technique presumes that hierarchy information is available for a divider under verification and uses this information to validate the correctness of the divider circuit. Moreover, layered rewriting does not scale well with larger divider circuits and limits itself to 21-bit dividers.

Recently, an approach has been mentioned in [1] that extends the algebraic rewriting technique to give rise to a new technique called hardware rewriting. Similar to layered rewriting, hardware rewriting also verifies the circuit layer-by-layer, where each layer computes the partial remainder R_j and quotient bit q_j from the divisor D and the previous remainder R_{j-1}. This technique also introduces an inverse circuit in each layer whose output Z_i is equivalent to $D * q_j + R_j = R_{j-1}$, and the goal remains to prove that every bit of Z_i matches R_{j-1}. This approach employs hardware synthesis tools to synthesise the circuit and examines the equivalence between Z_j and R_{j-1}. If hardware synthesis does not simplify the circuit to a redundant state, which is the case, especially for larger divider circuits, hardware rewriting verifies the circuit by employing bit-by-bit SAT or XOR comparison. Although this technique verifies the correctness of large divider circuits without memory explosion, it requires knowing the boundaries between the layers.

The success of the methods mentioned above depends mainly on hierarchical information. In most cases, logical optimizations tend to remove the hierarchical information, which could hamper the success of previously discussed methods. However, the approaches [3], [8] successfully verify large divider circuits at the gate level without utilising hierarchical information. These methods use optimization techniques such as *SAT-based information forwarding* (SBIF) and *Don't Care optimization* (DCOP) to circumvent the memory explosion problem that occurs when the backward rewriting replaces the quotient Q and the remainder R starting with the specification polynomial $Q \cdot D + R - R^{(0)}$ to verify the divider circuit.

As will be explained in the later sections, the divider must work within the range ($0 \leq R^{(0)} < D \cdot 2^{n-1}$, also known as the input constraint) of the allowed inputs. These methods extract the information from the input constraint and propagate the information during backward rewriting to avoid the exponential size of the intermediate polynomial. [8] is an SAT-based information forwarding that takes advantage of input constraints to obtain or identify equivalent and antivalent signals in the circuit. [3] employs Binary Decision Diagrams (BDDs) to compute satisfiability don't cares that arise from the divider circuit's structure and the input constraints.

The experimental results in [3] emphasise the drawbacks of other existing methods, such as SAT-based equivalence checking and AIG-based combinational equivalence checking, while confirming that DCOP works well even with large divider circuits. The experiment involves validating the equivalence of the divider circuit with a "golden specification circuit" by realising a miter circuit between the divider circuit and its golden specification circuit along with a circuit representing the input constraint $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The SAT solver fails to prove the equivalence for divider circuits with a size greater than 8 bits. Though the AIG-based rewriting approach via structural hashing, simulation and SAT solving identifies internal equivalent nodes to reduce the complexity of equivalence checking, it similarly fails as SAT-solving to verify divider circuits beyond 8 bits. Contrary to the results obtained by deploying SAT-solving and AIG-based rewriting using ABC, the backward rewriting with don't care optimization verifies divider circuits up to 512 bits in 162 CPU minutes without generating a memory explosion.

3 Background

The current section discusses preliminaries such as monomials and polynomials, symbolic computer algebra, algebraic rewriting, the restoring division algorithm, and the reduced restoring divider circuit structure that are required to understand the subsequent sections.

3.1 Monomials and Polynomials

Polynomials are nothing but algebraic expressions consisting of variables and coefficients. The formal definition of a monomial m in $x_1 \dots x_n$ is a product of form:

$$m^a = x_1^{a_1} \cdot x_2^{a_2} \cdots x_n^{a_n} \tag{1}$$

as defined in [9], where $a = (a_1, \dots, a_n)$ is a tuple that represents non-negative integer exponents and the total degree of the monomial is the sum of all exponents denoted by $|a| = a_1 + a_2 + \cdots + a_n$.

A polynomial is a linear combination of monomials with coefficients in the field K. The formal definition of a polynomial p in $x_1 \dots x_n$ with coefficients in field K is given as

$$p = \sum_{i=1}^{n} k_i m^{(i)}, \quad k_i \in K \tag{2}$$

Where k_i is the coefficient of the monomial $m^{(i)}$ [9]. The total degree of a polynomial is the maximum $|a|$ with a non-zero coefficient.

3.2 Symbolic Computer Algebra

As already introduced, Symbolic Computer Algebra-Based methods use polynomial rings in the algebraic domain to model the circuit specification (F) and implementation (G). The proof obligation that the circuit implementation (G) satisfies the specification (F) to verify the correctness of the circuit reduces to the *ideal membership test*.

The ideal membership test basically examines whether F lies in J; in other words, if $F \in J$. Here, $J =< f_1, \dots f_s >$ is an ideal generated by set of polynomials: $G = \{f_1, \dots f_s\}$ with $f_i \in \mathbb{Z}[X]$. Polynomials $f_i \dots f_s$ serve as the polynomial model of circuit implementation and are called generators or the base of ideal J. The examination of if $F \in J$ proceeds by evaluating the result of F modulo G denoted as $\mathrm{F} \xrightarrow{G}_{+} \mathrm{r}$, where r is the remainder. F modulo G is mathematically computed by first establishing the term ordering " $>$ " on monomials of F and G. As a result, each polynomial has a defined leading term. Then, the polynomial F is iteratively divided by the polynomials $\{f_1 \dots f_s\}$ in G to obtain the remainder r finally. This successive division aims to cancel the leading term of F by one of the leading terms of G. If this division yields the reminder $\mathrm{r} = 0$, then F vanishes on $V(J)$, thereby confirming that implementation satisfies the specification. Variety $V(J)$ is a set of all simultaneous solutions $f_1(x_1 \dots x_n) = 0, \dots f_s(x_1 \dots x_n) = 0$ for a given set of polynomials $\{f_1 \dots f_s\}$. Variety $V(J)$ can also serve as all signal values of a circuit produced from all possible input combinations. However, the limitation of this approach is its inconclusiveness in stating that an implementation G satisfies the specification F when G is insufficient to reduce F to 0, that is, when the remainder $\mathrm{r} \neq 0$. [10]

The computation of the canonical set of generators $G_b = \{g_1, g_2, \ldots g_s\}$ known as Groebner basis is one of the possible solutions to determine whether F is reducible to zero for a given ideal J. Therefore, with the Groebner basis, one can firmly conclude whether $F \in J$ [10]. Though using the Groebner basis for ideal membership testing provides unequivocal results, the algorithms realized to compute the Groebner basis are expensive. According to [11], *Groebner basis polynomial reduction* technique avoids costly computation of the Groebner basis by ordering the terms of each polynomial in set $G = \{f_1, \ldots f_t\}$ representing the gate-level implementation of the circuit in such a way that the leading term is the gate output, making the set G a Groebner basis. However, this approach also requires the *field polynomials* $J_0 =< x^2 - x >$ to restrict the variables to the Boolean domain, thus increasing the Groebner basis's size, resulting in larger search space. Authors of [10] propose a more efficient approach that implements ideal membership testing without using expensive polynomial division.

3.3 Backward Rewriting

The backward rewriting derives a unique bit-level polynomial function that encodes the gate-level implementation of an arithmetic circuit. Backward rewriting achieves this by translating the polynomial that represents the encoding of the primary outputs into a polynomial defined in terms of the primary inputs. The polynomial defined in terms of the primary input signals expresses the arithmetic function of the circuit and is known as the *input signature* (Sig_{in}) [10]. Similarly, the polynomial expressed in terms of the primary output signals represents the result computed by the circuit and is known as the *output signature* (Sig_{out}) [10]. Polynomial Sig_{out} is an n-bit encoding of the output. For example, the output signature of a one-bit adder is $2C + S$, where C and S are the carry-out and sum of a full adder, respectively.

The goal of backward rewriting is to transform the output signature (Sig_{out}) into the input signature (Sig_{in}) by considering the circuit's internal elements(logic gates) as algebraic models. The algebraic models represent logic gates by boolean expressions and signals by boolean variables. A network of logical elements of arbitrary complexity constitutes a circuit, where *pseudo-Boolean polynomial* $f_i[X]$ expresses each logical element with variables from $\mathbb{Z}_2$ (Binary) and coefficients from $\mathbb{Z}_{2^n}$.[10]

The following table provides algebraic equations for basic logic gates. Backward rewriting also extends to the verification of arithmetic circuits by comparing the Sig_{in} with the *Specification* (F_{spec}) polynomial.

Operation	Boolean model	Algebraic model
$INV(a)$	$\neg a$	$1 - a$
$AND(a, b)$	$a \wedge b$	ab
$OR(a, b)$	$a \vee a$	$a + b - ab$
$XOR(a, b)$	$a \oplus b$	$a + b - 2ab$
$XOR(a, b, c)$	$a \oplus b \oplus c$	$a + b + c - 2ab - 2bc - 2ac + 4abc$

Table 1: Algebraic models for basic logic functions

The specification polynomial is usually expressed in terms of primary inputs to characterise the integer function of the circuit. If the implementation satisfies the specification, that is, if the input signature is equivalent to the specification polynomial, the circuit's implementation is correct. Otherwise, the implementation is faulty. Backward rewriting serves as a means of verification only if the specification polynomial (F_{spec}) is known or provided by the designer; otherwise, Sig_{in} obtained from backward rewriting is considered as the circuit's function. With gate-level netlist and output signature Sig_{out} as inputs, the first step of the backward rewriting algorithm [10, alg. 1] is to extract algebraic equations from the gate-level implementation using Table 1. The next step involves the deployment of algorithms to derive the ordering of the algebraic equations according to the circuit's structure and topology. The ordering of the algebraic equation is crucial in maintaining the small size of intermediate polynomials. [10] discusses a few algorithms, such as levelization and dependency, that compute the substitution order for the algebraic equations.

Once ordering is complete, the rewriting process uses the predetermined order to transform one polynomial expression into another, starting with the output signature Sig_{out} at the primary outputs. Rewriting is an iterative process in which, in each iteration, the variables in the current equation are replaced by their corresponding gate polynomials, giving rise to intermediate polynomials (f_i). The backward rewriting algorithm terminates

when it expresses all the variables in terms of primary inputs or utilizes all the algebraic equations available.

A successful backward rewriting produces the input signature Sig_{in} that a formal verification algorithm uses to compare with the specification polynomial F_{spec} to determine the correctness of the circuit's implementation. The following example illustrates the extension of the backward rewriting process to verify the implementation of a one-bit full adder circuit. Figure 1 shows the gate-level implementation of a one-bit full adder. The specification of a one-bit adder is $F_{spec} = a + b + c_{in}$, and the output signature is equal to $Sig_{out} = 2c + s$. In Figure 1, (a, b, c_{in}) are primary inputs and (c, s) are primary outputs of the adder.

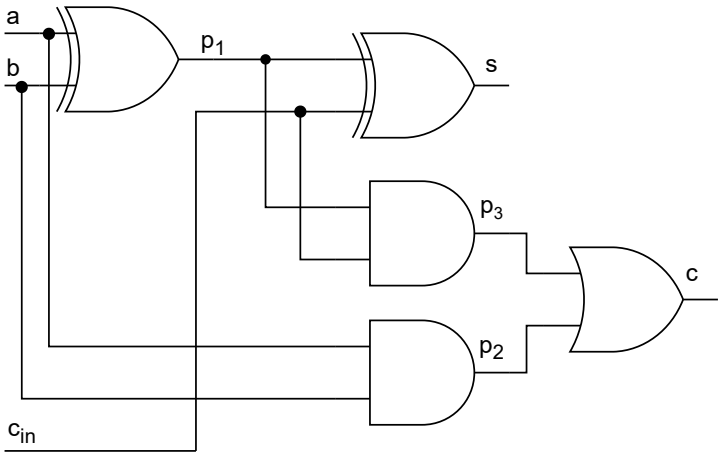

Figure 1: Gate-level implementation of one-bit full adder

The following illustration shows the iterative backward rewriting process of replacing every variable in the current expression with its corresponding gate polynomial, where $\xrightarrow{x}$ indicates that the variable 'x' is being replaced by its gate polynomial. The process begins at the primary outputs, with the polynomial f_0 representing Sig_{out}. It terminates with the polynomial f_5 defined in terms of the primary inputs after successively replacing all the variables in the algebraic expressions with their corresponding gate polynomial.

$$
\begin{aligned}
f_0 &= 2c + s \\
f_1 &\xrightarrow{c} 2p_3 + 2p_2 - 2p_2p_3 + s \\
f_2 &\xrightarrow{p_3} 2c_{in}p_1 + 2p_2 - 2c_{in}p_1p_2 + s \\
f_3 &\xrightarrow{s} 2p_2 - 2c_{in}p_1p_2 + p_1 + c_{in} \\
f_4 &\xrightarrow{p_2} 2ab - 2p_1abc_{in} + p_1 + c_{in} \\
f_5 &\xrightarrow{p_1} a + b + c_{in}
\end{aligned}
$$

Backward rewriting extends to verifying the presented implementation of the full adder by comparing the polynomial f_5 (Sig_{in}) with the corresponding specification polynomial F_{spec}. In the presented example, Sig_{in} is identical to F_{spec}, thereby concluding that the implementation is correct. Another way to formulate the verification problem is to check if the backward rewriting process reduces the polynomial $F = Sig_{out} - F_{spec}$ to zero. One of the critical advantages of the algebraic rewriting process is that it avoids the inclusion of field polynomials $< x^2 - x >$ because any variable x^k can be reduced to x as the variables in the pseudo-Boolean polynomial are in the binary domain. The work in [11] corroborates the conclusion that algebraic rewriting is an efficient approach for ideal membership testing and is a sound and complete method.

3.4 Two's Complement Representation of Binary Numbers

There are several ways to represent a signed binary number; the most efficient and widely used scheme is the two's complement representation. This representation resembles unsigned binary numbers except that the weight of the most significant bit is -2^{n-1} [12, Sec. 1.4.6]. The most significant bit is either zero when the binary number is a positive integer or one when it is a negative integer; thus, the most significant bit serves as the sign bit in this type of representation. For a given bit vector $(a_n, a_{n-1} \dots a_0) \in \{0,1\}^{n+1}$, the 2's complement representation is denoted as $[a_n, a_{n-1} \dots a_0]_2$ and interpreted as

$$[a_n, a_{n-1} \dots a_0]_2 = \sum_{i=0}^{n-1} a_i 2^{(i)} - a_n 2^n \tag{3}$$

Representing a negative binary number in 2's complement involves inverting all the bits of the binary number and then adding 1 to its least significant bit. This representation has two main advantages: 1) the addition works correctly for positive and negative n-bit numbers. 2) the subtraction of two binary numbers can be transformed into adding the first number with the negative version of the second number. Mathematically, $[a]_2 - [b]_2 = [a]_2 + ([-b]_2)$, where $[-b]_2 = [\bar{b}]_2 + 1$. Since this representation can represent both positive and negative numbers, the range of n-bit 2's complement number spans between $[-2^{n-1}, 2^{n-1} - 1]$.

3.5 Atomic Blocks

An atomic block is a basic building block of a digital circuit that, depending on the logic function, processes multiple binary inputs to produce single or multiple binary outputs. Atomic blocks are usually part of larger digital circuits, such as dividers and multipliers. The current work defines two specific atomic blocks, *Full Adder* (FA) and *Multiplexer* (MUX), which are part of the reduced restoring divider.

3.5.1 Full Adder

A full adder (Figure 1) is a combinational circuit that sums two one-bit binary numbers, a and b, with a one-bit carry-in c_{in} as the third input to produce a one-bit sum (s) and a carry-out (c_{out}) of one bit. The sum is determined as $s = a \oplus b \oplus c_{\text{in}}$, while the carry-out is determined by the expression $c_{\text{out}} = ab + ac_{\text{in}} + bc_{\text{in}}$. The following equation expresses the word-level relation between the inputs and the outputs of a full adder.

$$2c_{out} + s = a + b + c_{in} \tag{4}$$

3.5.2 One-Bit Multiplexer

Multiplexer or *mux* is a combinational circuit that chooses an output from multiple possible inputs based on the value of the select line. Figure 2 shows a 2:1 multiplexer with one-bit line s, and two data inputs a, b, and one-bit output y. The output y is equal to a when the select line $s = 1$ or equal to b when the select line $s = 0$. Since the select input of a 2:1 multiplexer is one-bit, a 2:1 multiplexer can also be called a one-bit multiplexer.

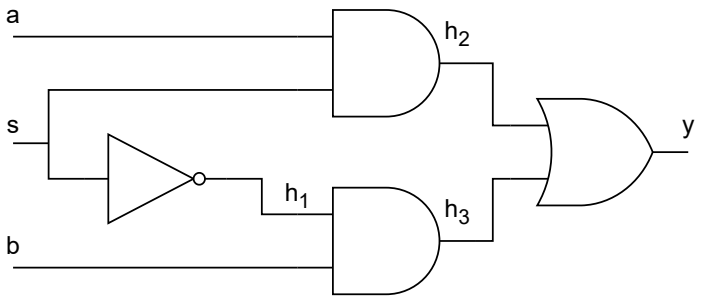

Figure 2: 2:1 multiplexer

The following equation expresses the word-level relation between the inputs and the outputs of a full adder.

$$y = s \cdot a + (1 - s)b \tag{5}$$

Equations (4) and (5) show that an atomic block has a compact word-level relation between its inputs and outputs. This unique property helps limit the size of the polynomial to a certain extent during backward rewriting.

3.6 Carry Ripple Adder

A carry-ripple adder is one of the most simple configurations of adders to add two n-bit binary numbers. Figure 3 illustrates an n-bit carry ripple adder that adds two n-bit inputs, $< a_{n-1}, a_{n-2}, \ldots, a_0 >$ and $< b_{n-1}, b_{n-2}, \ldots, b_0 >$, with one-bit carry-in (c_{in}) as the third input to output n-bit sum (S) and one-bit carry-out(c_{n-1}).

The full adders are basic building blocks of a carry-ripple adder and are connected as a chain to propagate the carry from one bit to the next; that is, the c_{out} of a previous full adder becomes c_{in} of the neighbouring full adder [12, Sec. 5.2.1]. The carry-ripple adder is a part of other larger circuits like restoring and non-restoring dividers because they provide modularity and regularity.

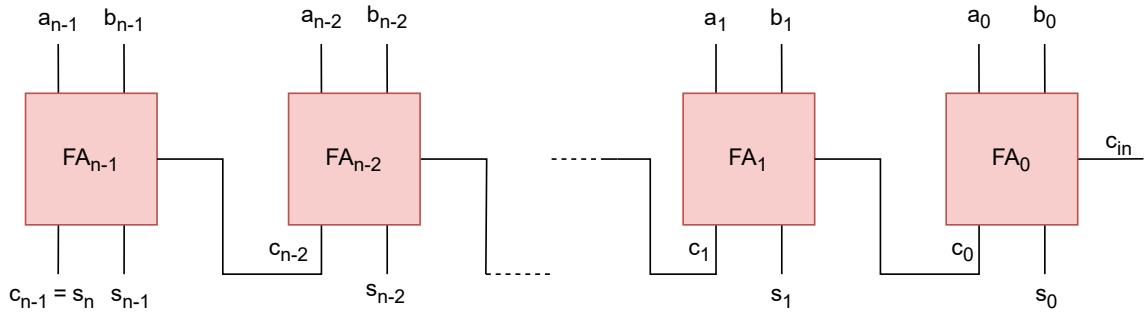

Figure 3: n-bit carry ripple adder

3.7 Multiplexers

Figure 4 illustrates an n-bit multiplexer consisting of n one-bit multiplexers with a single select line shared among all multiplexers [13]. It has two n-bit data inputs $a =< a_{n-1}, a_{n-2}, \ldots, a_0 >$, $b =< b_{n-1}, b_{n-2}, \ldots, b_0 >$, and n-bit output $y =< y_{n-1}, y_{n-2}, \ldots, y_0 >$. Similar to a one-bit multiplexer, if the select line is equal to one, then n-bit output y is equal to n-bit input a; else, the output y is equal to n-bit input b. This configuration of an n-bit multiplexer possesses the advantage of outputting an n-bit-wide input depending on the value of a single select line.

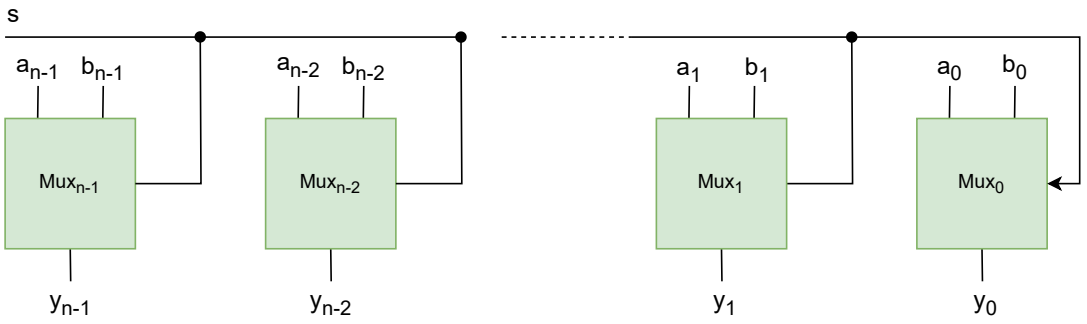

Figure 4: n-bit multiplexer

3.8 Binary Decision Diagrams

Binary decision diagrams (BDDs) efficiently manipulate Boolean functions and represent Boolean functions as directed acyclic graphs. A BDD is a directed acyclic graph $G = (V, E)$ with one root node $v \in V$ and is defined over a set of variables $X_n = \{x_1, x_2, \ldots, x_n\}$ and a value set $B := \{0, 1\}$. A BDD comprises terminal nodes and non-terminal nodes, where the label for each terminal node is a value $b \in B$, and the label of a non-terminal node is a variable $x_i \in X_n$. Each non-terminal node v has exactly two outgoing edges, which leads to nodes denoted as low(v) and high(v) ($low(v), high(v) \in V$). [14]

A BDD G represents a Boolean function $f : B^n \longrightarrow B$, which is defined inductively as following [14]; If f is constant function 0 (1), G represents this function with a single terminal node labeled with 0 (1). BDD G with root node v labeled as x_i represents the function f as

$$f = (\neg x_i \wedge f_{low(v)}) \vee (x_i \wedge f_{high(v)})$$

where $f_{low(v)}$ and $f_{high(v)}$ represent the functions represented by BDD rooted in $low(v)$ and $high(v)$, respectively. Shannon's expansion theorem [14] for Boolean functions forms the basis of the semantics used to represent this function. The advent of *Reduced Ordered Binary Decision Diagram* (ROBDDs) vanquishes the limitation of generic BDDs that fail to provide a canonical representation of a boolean function; that is, every boolean function has a unique representation.

A BDD G qualifies as a Reduced Ordered Binary Decision Diagram if it is both ordered and reduced [14]. Ordered and reduced are the two key aspects that have to be determined to classify a BDD as ROBDD or not. For a BDD to be considered an ordered BDD, it first has to be free; that is, each variable along every path from the root to a terminal node occurs at most once. A free BDD is an ordered BDD if the variables on every path from the root to a terminal node occur in the same order. A BDD free of redundancy is a reduced BDD. Isomorphism reduction and Shannon reduction are two primary reduction rules to eliminate redundancy in a BDD, thereby transforming an ordered BDD into a reduced ordered BDD [14]. The sequence in which variables occur on every path from the root to a terminal node is known as the variable order. The number of non-terminal nodes determines the size of a given BDD.[14]

If-Then-Else (ITE) is a widely used operator that possesses the advantage of representing any binary operation. The ternary ITE operator over three Boolean functions $F, G, H : B^n \to B$ is defined as follows [14] :

$$\text{ITE}(F, G, H) = (F \wedge G) \vee (\neg F \wedge H)$$

For example, the ITE operator realizes the AND operation between F and G as $\text{ITE}(F, G, 0)$, which translates as $(F \wedge G) \vee (\neg F \wedge 0) = (F \wedge G)$. Similarly, the ITE operator realizes other Boolean operations such as OR, NOT, NAND, and NOR [14]. The following theorem [14] forms the basis for constructing a BDD using the ITE operator.

Let $F, G, H : B^n \to B$ be three Boolean functions over the set of variables $X_n = \{x_1, \ldots, x_n\}$. For each $i \in \{1, \ldots, n\}$, the following holds:

$$\text{ITE}(F, G, H) = (\neg x_i \wedge \text{ITE}(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i})) \vee (x_i \wedge \text{ITE}(F_{x_i}, G_{x_i}, H_{x_i}))$$

The algorithm (improved ITE algorithm) in [14] deduced from the above theorem recursively constructs ROBDDs of $\text{ITE}(F_{\neg x_i}, G_{\neg x_i}, H_{\neg x_i})$ and $\text{ITE}(F_{x_i}, G_{x_i}, H_{x_i})$ for a top variable $x_i \in X$. The algorithm deploys Shannon reduction(line 9) and Isomorphism reduction(line 10) whenever applicable to construct a non-redundant BDD.

The procedure continues until the algorithm encounters a terminal case [14]. The algorithm maintains two hash tables, INSERT_COMPUTED_TABLE((F,G,H),r) and CHECK_COMPUTED_TABLE(F,G,H) to construct ROBDD efficiently.

The INSERT_COMPUTED_TABLE records the result r (a pointer to type robdd) for a given argument combination (F, G, H). The CHECK_COMPUTED_TABLE determines whether an argument combination has been previously processed and returns the corresponding result, thereby avoiding unnecessary recomputation. These hash tables avoid the exponential run-time of the algorithm and, in the best case, the run-time lies in $O(|F| \cdot |G| \cdot |H|)$, where $|\cdot|$ indicates the size of a ROBDD.

3.9 Restoring Division Algorithm

The restoring division algorithm is a simple algorithm that delineates the procedure to obtain the quotient Q and the remainder R when the dividend $R^{(0)}$ is divided by the divisor D. In short, the algorithm transforms the dividend into the remainder and obtains the quotient by successively subtracting the shifted version of the divisor from the dividend. The algorithm begins with the divided $R^{(0)} =< r_{2n-2}^{(0)} \dots r_0^{(0)} >$ and divisor $D =< d_{n-1} \dots d_0 >$. In each iteration j spanning from 1 to n, the divisor shifts left by $n - j$ positions, and the algorithm computes the partial remainder $R^{(j)}$ by subtracting the shifted version of the divisor from the previous partial remainder $R^{(j-1)}$. If the partial remainder $R^{(j)}$ is negative, the algorithm sets the quotient bit q_j to 0 and restores the partial remainder by adding the shifted version of the divisor back to the partial remainder, thereby ensuring the partial remainder remains non-negative; else, the algorithm sets the quotient bit q_j to 1, and the iteration continues with next shifted version of D. Algorithm 1 summarizes the restoring division [4].

At the end of the iteration, that is, when j = n, a complete quotient Q and a final reminder $R/R^{(n)}$ are obtained. The algorithm performs a complete and correct division to obtain unique values for R and Q if it obeys the *Quotient-Remainder theorem* [15, Sec. 4.1.3]. The theorem states: given any integers p and d, where $d > 0$, there exist integers q and r such that $p = dq + r$, where $0 \leq r < d$ and q,r are uniquely determined.

Algorithm 1 Restoring division

1: **for** $j = 1$ to n **do**
2:
3: $R^{(j)} := R^{(j-1)} - D \cdot 2^{n-j}$;
4:
5: **if** $R^{(j)} < 0$ **then**
6: $q_{n-j} := 0$;
7:
8: $R^{(j)} := R^{(j)} + D \cdot 2^{n-j}$;
9: **else**
10: $q_{n-j} := 1$;
11:
12: **end if**
13:
14: **end for**
15:
16: $R := R^{(n)}$;

According to *Lemma 3* in [16, Section 2.1] restoring division algorithm computes the quotient Q and remainder R with $R^{(0)} = Q \cdot D + R$ and $0 \leq R < D$, which directly obeys the Quotient-Remainder theorem.

Here *Condition 1 (VC1)*: $R^{(0)} = Q \cdot D + R$ ensures that the division is correct and complete, whereas *Condition 2 (VC2)*: $0 \leq R < D$ restricts the range of the remainder to guarantee a unique quotient and remainder, thus providing unambiguous results.

Therefore, it is essential to verify conditions *(VC1)* and *(VC2)* to confirm the correctness of a restoring divider. Additionally, the constraint: $0 \leq R^{(0)} < D \cdot 2^{n-1}$ is established on the inputs of the divider circuit to satisfy condition 2. *Lemma 6* in [16, Section 5.1](rewritten as *Lemma 1* in Section 3.9) supports this statement by extending the input constraint *IC* to derive the size of all partial remainders. As a result of this, the condition $0 \leq R < D$ holds. Since the constraint applies to the inputs of the divider, it is called the *Input Constraint (IC)*.

3.10 Structure of Restoring Divider

Despite various design principles for implementing restoring division, the current section illustrates two structures: a non-optimized restoring divider and a reduced restoring divider (optimized). Both use the carry-ripple adder and multiplexers to perform the same functionality. This section discusses the implementation of the non-optimized

restoring divider first, followed by an explanation of how a non-optimized restoring divider transforms into a reduced restoring divider.

The following steps summarize the restoring division algorithm: 1) Subtract the shifted version of the divisor D from the partial reminder $R^{(j)}$. 2) Set the quotient bit q_j to 1 if the partial remainder computed in Step 1 is positive; otherwise, set $q_j = 0$. 3) Restore the partial remainder $R^{(j)}$ if q_j is 0; else, continue to Step 4. 4) Repeat steps 1 to 3 with D shifted by one bit less in each iteration. These four steps are key to designing and implementing a correct restoring divider. A high-level circuit structure of a non-optimized restoring divider is shown in Figure 5. The circuit implements n iterations specified in the algorithm 1 by realising n stages. The design splits each stage $Stage_j$ into two sub-stages, subtractor stage denoted as $Stage_j^{SUB}$ and multiplexer stage denoted as $Stage_j^{MUX}$, where j spans from 1 to n. Figure 6 illustrates the implementation of an individual stage $Stage_j$, using the first stage as an example. The design principle and the circuit's structure remain consistent across all stages of a non-optimized restoring divider. One of the key differences between the first stage and the subsequent stages is that, in the latter, one of the inputs to $Stage_j^{SUB}$ is the partial remainder $R^{(j-1)}$ instead of the dividend $R^{(0)}$. The incoming partial remainder of any stage is the outgoing partial remainder from the previous stage. For any subtractor stage $Stage_j^{SUB}$, the inputs comprise of incoming partial remainder $R^{(j-1)}$ (dividend $R^{(0)}$ with bit width $2n-1$ for $Stage_1^{SUB}$) and divisor $D =< d_{n-1} \dots d_0 >$ shifted left by $n-j$ positions . The implementation realizes each subtractor stage $Stage_j^{SUB}$ using a carry-ripple adder (CRA) extended with an overflow correction circuit (OV) and inverters. This stage performs the subtraction by adding the negative version of the shifted divisor D to the incoming partial remainder $R^{(j-1)}$. In other words, $Stage_j^{SUB}$ translates $R^{(j-1)} - D \cdot 2^{n-1}$ to $R^{(j-1)} + (-D) \cdot 2^{n-1}$ using 2's complement representation. Thus, to realize $-D$ in 2's complement representation, the circuit uses inverters to flip every bit of D before using it as one of the inputs to the CRA. The implementation also sets the carry-in bit of the CRA to one, which adds one to the flipped version of D, thereby effectively realising $-D$ in 2's complement representation. The implementation uses the underlying principle of **Theorem 2** present in [16] to extend the carry-ripple adder with overflow correction. The output of $Stage_j^{SUB}$, denoted as intermediate result $T^{(j)}$ represents, $R^{(j-1)} - D \cdot 2^{n-j}$.

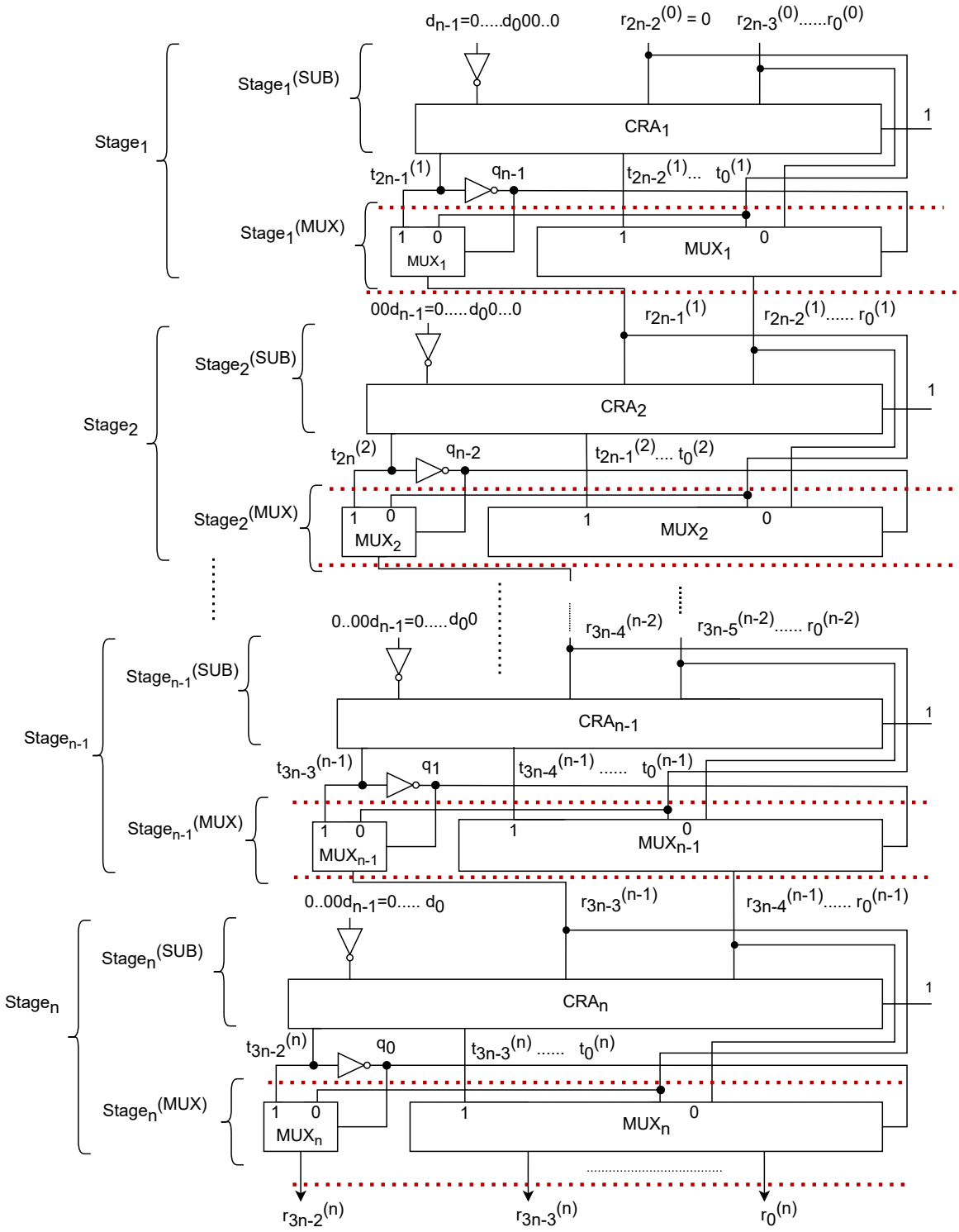

Figure 5: Non-optimized restoring divider

The most significant bit $t^{(j)}_{2n+j-2}$ of the CRA with overflow circuit belonging to $Stage^{SUB}_j$ is negated to determine the quotient bit q_{n-j}. This quotient bit q_{n-j} also serves as a select line to the multiplexer stage $Stage^{MUX}_j$. The output of the $Stage^{SUB}_j$, that is, $T^{(j)}$ and the incoming partial remainder $R^{(j-1)}$ are the inputs to the $Stage^{MUX}_j$, which determines the outgoing partial remainder $R^{(j)}$. The outgoing partial reminder $R^{(j)}$ is equivalent to $T^{(j)}$ when the select line q_{n-j} is equal to one, indicating the subtraction result was positive. The quotient bit q_{n-j} equal to zero indicates that the subtraction result was negative, therefore, the multiplexer chooses $R^{(j-1)}$ as the outgoing partial remainder $R^{(j)}$. This design principle followed in each multiplexer stage $Stage^{MUX}_j$ enables a simple way of restoring the partial remainder. Furthermore, the implementation of the multiplexer stage $Stage^{MUX}_j$ always ensures that the partial remainder remains positive. As mentioned earlier, the subsequent stage also follows the same design principle, but due to the presence of an overflow correction circuit, the bit width of the partial remainder $R^{(j)}$ is one bit more than the previous partial remainder $R^{(j-1)}$. The implementation pads the $n-j$ left shifted positions of D with zero and extends the sign bit of D to match the bit width of the incoming partial remainder $R^{(j-1)}$.

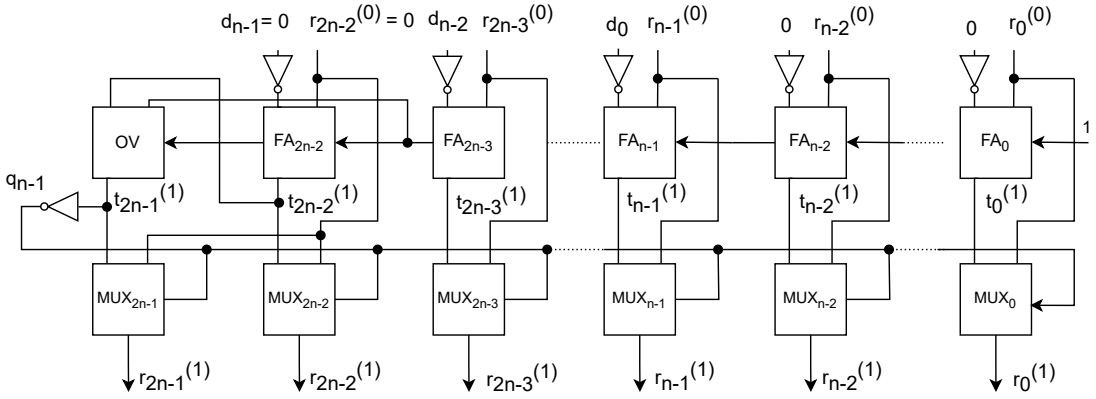

Figure 6: First stage of a non-optimized restoring divider

The following part of this section illustrates the transformation of a non-optimized restoring divider into a n bit width reduced restoring divider. According to [12, Sec. 1.4.6], adding a positive number to a negative number never causes an overflow. From the restoring division algorithm 1, it is clear that the partial remainder at any given stage is a positive integer. Therefore, the operation $R^{(j-1)} - D \cdot 2^{n-j}$ shall never cause

an overflow. Hence, in each stage $Stage_j$, the optimized design removes the overflow correction circuit and the corresponding multiplexer. Furthermore, in each iteration the algorithm left-shifts D by n-j positions, therefore eliminating the need for $n-j$ right-most full adders of the CRA belonging to $Stage_j^{SUB}$. Consequently, removing the $n-j$ right-most full adders makes the corresponding multiplexers of $Stage_j^{MUX}$ unnecessary because these multiplexers receive one of their inputs from the full adders. Hence, the design also removes the $n-j$ right-most multiplexers of $Stage_j^{MUX}$. The present section reintroduces *Lemma 6* from [16] to explain the pruning of the leftmost full adders and multiplexers of $Stage_j$. The following lemma holds for the size of the partial remainder [16].

Lemma 1: For all partial remainders in a restoring division with $IC: 0 \leq R^{(0)} < D \cdot 2^{n-1}$, $0 \leq R^{(n-j)} < D \cdot 2^{n-j}$ is true. [16] proves this lemma by using the method of induction. *Lemma 1* also extends to the size of intermediate result $T^{(j)}$: $-D \cdot 2^{n-j} \leq T^{(j)} < D \cdot 2^{n-j}$. These size constraints on all the partial reminders and intermediate results help to prove the following statement.

Corollary 1: For $1 \leq j \leq n$, $R^{(j)}$ and $T^{(j)}$ can be represented by a two's complement representation with $2n-j$ bits [16].

Proof : From lemma 1, $0 \leq R^{(n-j)} < D \cdot 2^{n-j}$ and $-D \cdot 2^{n-j} \leq T^{(j)} < D \cdot 2^{n-j}$ are true. Additionally, $1 \leq D < 2^{n-1}$ is true since an unsigned integer divisor divides the dividend. Therefore, $0 \leq R^{(n-j)} < 2^{2n-j-1}$ and $-2^{2n-j-1} \leq T^{(j)} < 2^{2n-j-1}$ is true.

Corollary 1 helps to establish that the restoring divider circuit does not have to compute any bits with indices higher than $2n-j-1$. As a result, the partial remainder $R^{(j)}$ of any $Stage_j$ can be represented by a bit sequence $< r_{2n-j-1}^{(j)} \dots r_0^{(j)} >$. The same argument holds for $T^{(j)}$. This representation of $R^{(j)}$ and $T^{(j)}$ enables the pruning of $2j-2$ leftmost full adders of each $Stage_j^{SUB}$ because to calculate $T^{(j+1)}$, all the bits in $R^{(j)}$ with indices exceeding $2n-j-2$ are not required. All the bits in $T^{(j+1)}$ with indices exceeding $2n-j-2$ of is just a sign extension of $< t_{2n-j-2}^{(j+1)} \dots t_0^{(j+1)} >$. Thus, the pruning of $2j-2$ left most full adders and corresponding multiplexers of every stage ($Stage_j$) transforms a clean non-optimized restoring divider to an n bit-width optimized reduced restoring divider. Figure 7 illustrates a 4-bit reduced restoring divider, which follows the design principle presented in the current section. The 4-bit reduced restoring divider has four stages in total, with each stage comprising a 4-bit carry ripple adder and a 4-bit multiplexer, thereby having a uniform width of 4 in all stages.

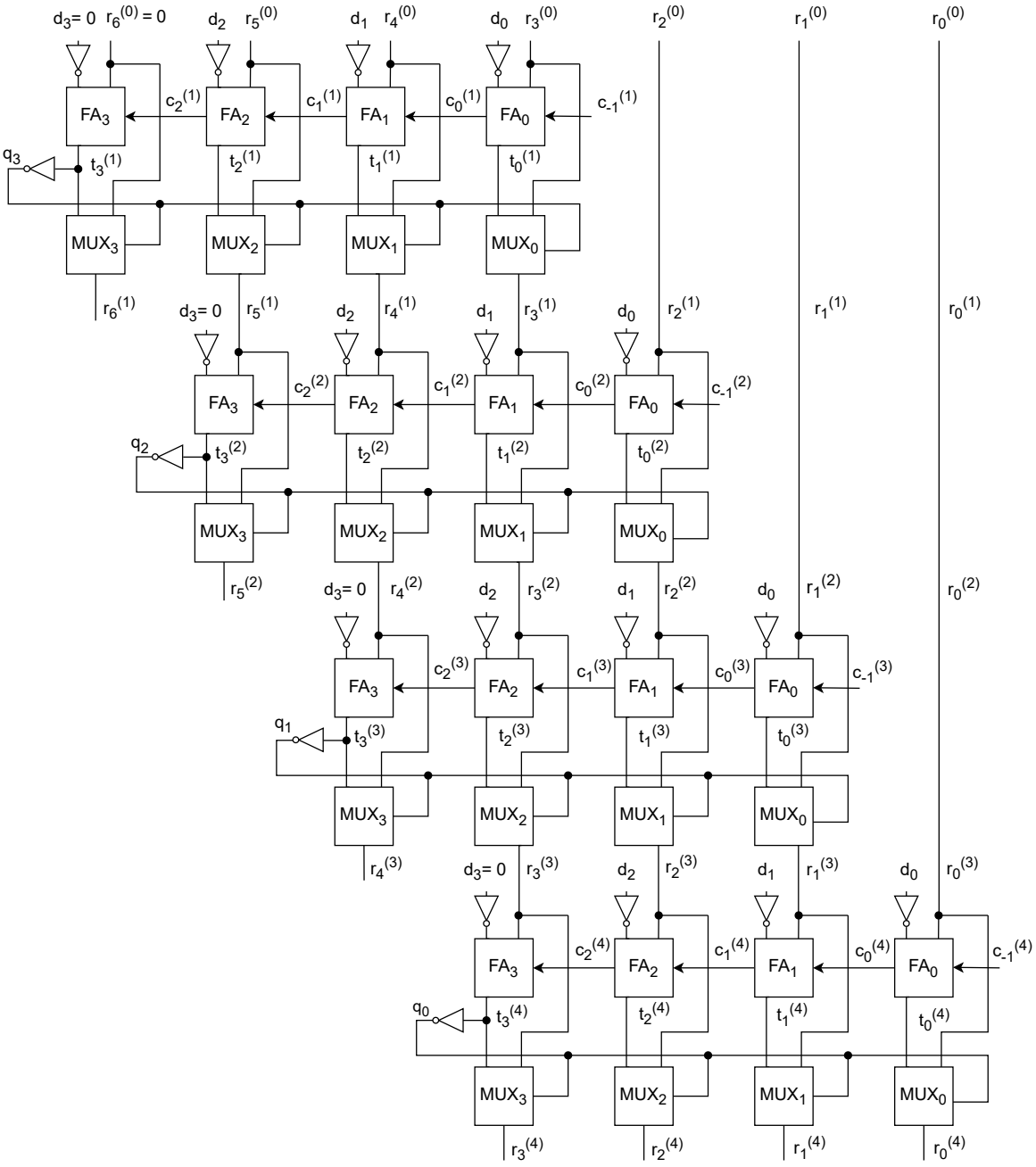

Figure 7: 4-bit reduced restoring divider

4 Preliminary Analysis

The current section presents two prerequisite analyses that act as building blocks for a more convenient analysis of the divider. It consists of two key sub-sections: Section 4.1 demonstrates the analysis of n-bit carry-ripple adder, and Section 4.2 illustrates the formal analysis of n-bit multiplexers. Section 3.6 briefly introduces carry-ripple adders, and Section 3.7 illustrates their role in a reduced restoring divider. The analysis provided in this section forms the foundation for verifying the reduced restoring divider.

4.1 Analysis of Carry Ripple Adder

The analysis of a carry-ripple adder begins with a full adder, which is the basic building block of a carry-ripple adder. As already mentioned in Section 3.5.1, a full adder atomic block establishes a compact word-level relationship between its inputs and outputs (see Equation 4): $2c_{out} + s = a + b + c_{in}$. This expression $2c_{out} + s = a + b + c_{in}$ plays a vital role in verifying a carry-ripple adder efficiently.

The output signature of an n-bit carry-ripple adder is equivalent to $\sum_{i=0}^{n} s_i 2^{(i)}$, where s_n is the carry-out and the remaining bits $< s_{n-1} \dots s_0 >$ correspond to the n-bit sum of the carry-ripple adder. Rewriting the output signature as

$$\sum_{i=0}^{n} s_i 2^{(i)} = \sum_{i=0}^{n-2} s_i 2^{(i)} + s_{n-1} 2^{n-1} + s_n 2^n \tag{6}$$

In the above polynomial, $s_{n-1}2^{n-1} + s_n 2^n$ is identical to Equation (4): $2c_{out} + s$. Therefore, the backward rewriting replaces $s_{n-1}2^{n-1} + s_n 2^n$ as $a_{n-1}2^{n-1} + b_{n-1}2^{n-1} + c_{n-2}2^{n-1}$. where $(a_{n-2}, b_{n-2}, c_{n-3})$ are the inputs to the most significant full adder FA_{n-1}. This substitution simplifies the above equation to the following expression.

$$\sum_{i=0}^{n} s_i 2^{(i)} = \sum_{i=0}^{n-2} s_i 2^{(i)} + a_{n-1} 2^{n-1} + b_{n-1} 2^{n-1} + c_{n-2} 2^{n-1} \tag{7}$$

Similarly, $s_{n-2}2^{n-2} + c_{n-2}2^{n-1}$ represent the outputs of the full adder FA_{n-2} and backward rewriting replaces this expression by $a_{n-2}2^{n-2} + b_{n-2}2^{n-2} + c_{n-3}2^{n-2}$, where

$(a_{n-2}, b_{n-2}, c_{n-3})$ are the inputs of the full adder FA_{n-2}. Consequently, giving rise to the following polynomial:

$$\sum_{i=0}^{n} s_i 2^{(i)} = \sum_{i=0}^{n-3} s_i 2^{(i)} + \sum_{i=n-2}^{n-1} a_i 2^i + \sum_{i=n-2}^{n-1} b_i 2^i + c_{n-3} 2^{n-3} \tag{8}$$

The backward rewriting replaces the remaining full adders by their corresponding equations (Equation (4) to transform the output signature to a polynomial defined in terms of the primary inputs, thats is, the input signature. The final expression (Sig_{in}) is given as:

$$\sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i + c_{in} \tag{9}$$

The specification polynomial of an n-bit carry-ripple adder: $F_{spec} = \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i + c_{in}$ is precisely equivalent to the input signature derived through backward-rewriting. Moreover, substituting the complete adder equation while rewriting avoids exponential blow-up of the intermediate polynomial size, as every full adder is replaced with a constant number of monomials (three monomials), keeping the overall size of the polynomial linear.

4.2 Analysis of Multiplexer

The analysis of an n-bit multiplexer begins with analysing a one-bit multiplexer, which is the basic building block of an n-bit multiplexer discussed in the current work. The current subsection considers the gate-level implementation shown in Figure 4 to verify a one-bit multiplexer. The fundamental backward rewriting process replaces the output y of the 2:1 multiplexer with its corresponding gate polynomial (OR gate), resulting in the intermediate polynomial $h_3 + h_2 - h_3 h_2$. In the next iteration, backward rewriting replaces intermediate signals $h3$ and $h2$ with the polynomial representation for the two-input AND gate. As a result, this gives rise to the following expression:

$$b.h1 + a.s - bh1sa. \tag{10}$$

Replacement of $h1$ as $1 - s$ ($\because$ NOT gate) results in the input signature $Sig_{in} = a.s + (1 - s)b$, which is identical to the specification polynomial of a one-bit multiplexer.

Thus, the presented implementation is equivalent to the specification of a one-bit multiplexer. This analysis of a one-bit multiplexer derives the following equation that relates the output of the multiplexer to its inputs, which is essential in verifying an n-bit multiplexer. The following equation is also identical to Equation (5) that establishes a word-level relation of a one-bit mux atomic block.

$$y = a.s + (1 - s)b \tag{11}$$

The following equation provides the specification polynomial of an n-bit multiplexer with two n-bit signed data inputs $[a_{n-1} \dots a_0]_2 = \sum_{i=0}^{n-1} a_i 2^{(i)} - a_{n-1}2^{n-1}$, $[b_{n-1} \dots b_0]_2 = \sum_{i=0}^{n-1} b_i 2^{(i)} - b_n 2^{n-1}$, and one bit unsigned select bit s.

$$F_{spec} = s(\sum_{i=0}^{n-1} a_i 2^{(i)} - a_{n-1}2^{n-1}) + (1 - s)\sum_{i=0}^{n-1} b_i 2^{(i)} - b_n 2^{n-1} \tag{12}$$

Similarly, the output signature that encodes the result of a signed n-bit multiplexer is equivalent to $\sum_{i=0}^{n-1} y_i 2^{(i)} - y_{n-1}2^{n-1}$. The backward rewriting substitutes every term in the output signature with its corresponding multiplexer equation (11). For example, substituting $y_{n-1} = a_{n-1}.s + (1 - s)b_{n-1}$ in the output signature results in,

$$\sum_{i=0}^{n-1} y_i 2^{(i)} - 2^{n-1}a_{n-1}.s - 2^{n-1}(1 - s)b_{n-1} \tag{13}$$

Where a_{n-1} and b_{n-1} are inputs to the multiplexer MUX_{n-1}. With similar substitutions for the remaining multiplexers in an n-bit multiplexer, the output signature transforms into the input signature, $Sig_{in} = s(\sum_{i=0}^{n-1} a_i 2^{(i)} - a_{n-1}2^{n-1}) + (1 - s)\sum_{i=0}^{n-1} b_i 2^{(i)} - b_n 2^{n-1}$. The input signature is equivalent to the specification polynomial (12), thus indicating the correct implementation of the n-bit multiplexer. The current analysis shows that the multiplexer equation (11) used during the backward rewriting of the n-bit multiplexer makes the verification simple and efficient without an exponential increase in the polynomial's size.

5 Analysis of Restoring Divider

After the brief introduction to restoring division and fundamental analysis of an n-bit carry-ripple adder and n-bit multiplexer, the current section provides insight into verifying the restoring dividers. This section begins by presenting an unusual specification polynomial (Section 5.1) for an integer divider, followed by a high-level analysis of the non-optimized restoring divider (Section 5.2). It continues by highlighting the shortcomings of basic backward rewriting without any optimization (Section 5.3). The remaining sections introduce novel optimization techniques to overcome the limitations of basic backward rewriting.

5.1 Specification Polynomial of Integer Dividers

Unlike other arithmetic circuits, divider outputs cannot be directly expressed in terms of its inputs because it lacks a closed-form formula to represent its outputs as functions of its inputs. Hence, the input and output signatures of a divider circuit are obscure. The following characteristic equation expresses the functionality of the divider

$$R^{(0)} = D \cdot Q + R, \quad \text{with} \quad R < D \tag{14}$$

Where Q (quotient), R (reminder) are the outputs, and $R^{(0)}$ (dividend) , D (divisor) are the inputs. The specification polynomial for any divider resembles Equation (14). Instead of the usual backward rewriting that begins with the polynomial representation of the output, backward rewriting of a divider starts with specification polynomial: $D \cdot Q + R$ and it replaces Q and R with their corresponding gate polynomials. Alternatively, the goal of verification is to determine whether backward rewriting reduces $D \cdot Q + R - R^{(0)}$ to zero. However, the verification of the divider is incomplete without the verification of the condition: $0 \leq R < D$ that governs the uniqueness of the result obtained from the division.

5.2 High-Level Analysis of Restoring Divider

Section 3.9 delineates the restoring division algorithm, and Section 3.10 illustrates the implementation that realizes the algorithm. This section presents a high-level analysis of the non-optimized restoring divider. Figure 5 shows a generalized view of the non-optimized restoring divider. The output of each subtractor stage $Stage_j^{SUB}$ fulfils the following equation.

$$T^{(j)} = R^{(j-1)} - D \cdot 2^{n-j} \tag{15}$$

and the output of each multiplexer $Stage_j^{MUX}$ fulfills the equation

$$R^{(j)} = q_{n-j}T^{(j)} + (1 - q_{n-j})R^{(j-1)} = q_{n-j}T^{(j)} - q_{n-j}R^{(j-1)} + R^{(j-1)} \tag{16}$$

The following part of the current section generalises the polynomials that must occur at the cuts (dashed red lines in Figure 5) of the non-optimized restoring divider circuit when the backward rewriting begins with the specification polynomial $D \cdot Q + R - R^{(0)}$ and replaces the gates with their corresponding polynomial one stage after the other.

The polynomial after $Stage_n^{MUX}$: $\sum_{i=0}^{n-1} q_i 2^i \cdot D + R^{(n)} - R^{(0)}$. Backward rewriting uses Equation (16) to replace $Stage_n^{MUX}$, which results in the expression : $\sum_{i=0}^{n-1} q_i 2^i.D + q_0T^{(n)} - q_0R^{(n-1)} + R^{(n-1)} - R^{(0)}$, and it further simplifies to $\sum_{i=1}^{n-1} q_i 2^i \cdot D + [q_0D + q_0T^{(n)} - q_0R^{(n-1)}] + R^{(n-1)} - R^{(0)}$. Similarly, the polynomial after replacing $Stage_n^{SUB}$ is $\sum_{i=1}^{n-1} q_i 2^i D + R^{(n-1)} - R^{(0)}$, since $[q_0D + q_0T^{(n)} - q_0R^{(n-1)}]$ reduces to zero when the backward rewriting employs Equation (15) to replace $T^{(n)}$ by $R^{(n-1)} - D$.

Generally, every $Stage_j^{MUX}$ imposes Equation (16) and the backward rewriting of any $Stage_j^{MUX}$ produces a polynomial.

$$P_1(j) = \sum_{i=n-j+1}^{n-1} q_i 2^i D + [q_{n-j} 2^{n-j} D + q_{n-j} T^{(j)} - q_{n-j} R^{(j-1)}] + R^{(j-1)} - R^{(0)} \tag{17}$$

A similar generalization also holds for the polynomial corresponding to $Stage_j^{SUB}$ when the backward rewriting imposes Equation (15) to reduce $[q_{n-j} 2^{n-j} D + q_{n-j} T^{(j)} - q_{n-j} R^{(j-1)}]$ to zero and give rise to the following expression between $Stage_{j-1}^{MUX}$ and $Stage_j^{SUB}$.

$$P_2(j) = \sum_{i=n-j+1}^{n-1} q_i 2^i \cdot D + R^{(j-1)} - R^{(0)} \tag{18}$$

Additionally, the following expression generalises the polynomial at the cut of every stage $Stage_j$, where $j \in \{1 \ldots n\}$.

$$\sum_{i=n-j}^{n-1} q_i 2^i \cdot D + R^{(j)} - R^{(0)} \tag{19}$$

The generalization made from equations (17) and (18) also extends to the first stage, where the backward rewriting of $Stage_1^{MUX}$ results in polynomial $[q_{n-1} 2^{n-1} D + q_{n-1} T^{(1)} - q_{n-1} R^{(0)}]$ between $Stage_1^{MUX}$ and $Stage_1^{SUB}$. Finally, the backward rewriting of $Stage_1^{SUB}$ using Equation (15) reduces the polynomial to 0. Consequently, this indicates that the implementation of the restoring divider aligns with the specification.

The analysis presented in the current section makes backward rewriting simple. However, the analysis presented in the current section only applies to a clean non-optimized restoring divider. In practice, a few obvious obstacles can make the backward rewriting of divider circuits very challenging: 1) the backward rewriting may not hit the expected cuts between stages, and 2) there may be a significant increase in the size of polynomials between cuts [4]. The preceding section highlights the challenges that usually occur in practice.

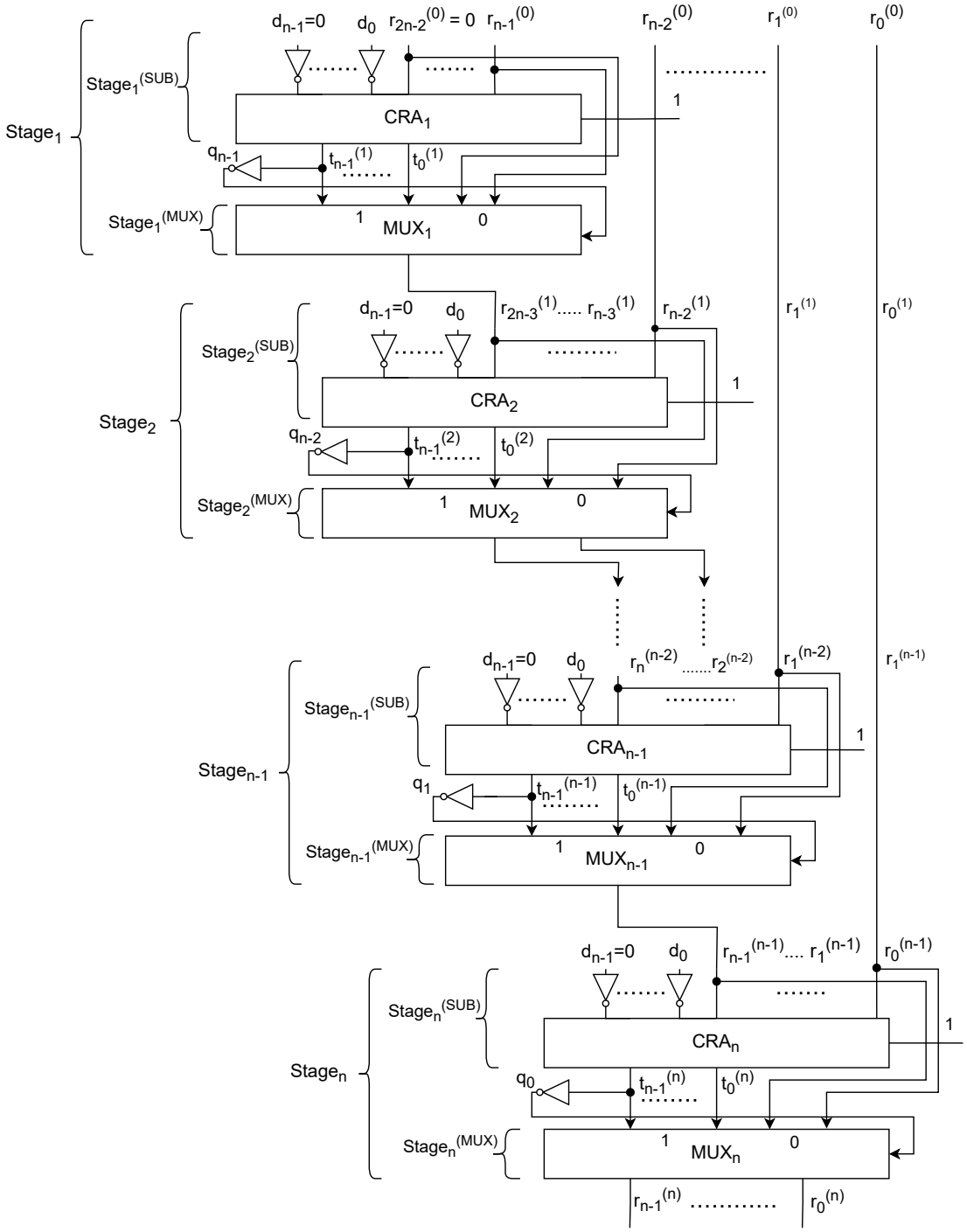

Figure 8: High-level reduced restoring divider

5.3 Naive Analysis of Reduced Restoring Divider

This section discusses the fundamental backward rewriting of the reduced restoring divider shown in Figure 8. This naive approach does not use any optimizations such as *Don't Care optimization* or *SAT-based information forwarding* but instead replaces the nontrivial atomic blocks [17] such as full adders, multiplexers, and trivial atomic blocks like original gates (AND, OR, XOR, NOT) with their corresponding polynomial equation.

Verifying a divider is not just restricted to reducing the specification polynomial F_{spec} to zero but also proving that the condition $0 \leq R < D$ always holds. However, the current section highlights the drawback of naive backward rewriting employed to reduce the specification polynomial to zero. The analysis begins with the last stage ($Stage_n$) of the divider, consisting of an n-bit CRA and an n-bit multiplexer, as shown in Figure 8. Additionally, to simplify the analysis, 2's complement representation expresses $R^{(n)}$ as $\sum_{i=0}^{n-2} r_i^{(n)} 2^i - 2^{n-1} r_{n-1}^{(n)}$. As a result of this, Equation (14) translates to:

$$F_{spec} = \sum_{i=0}^{n-1} q_i 2^i \sum_{i=0}^{n-2} d_i 2^i + \sum_{i=0}^{n-2} r_i^{(n)} 2^i - 2^{n-1} r_{n-1}^{(n)} - R^{(0)} \tag{20}$$

The Backward rewriting obtains the polynomial between $Stage_n^{MUX}$ and $Stage_n^{SUB}$ by replacing every bit in $R^{(n)}$ by the multiplexer equation (Equation (11)): $q_0 t_i^{(n)} + (1-q_0) r_i^{(n-1)}$, where $0 \leq i \leq n-1$. The polynomial simplifies further by rearranging the term associated with q_0 together, resulting in the following expression.

$$\begin{aligned} &\sum_{i=1}^{n-1} q_i 2^i \cdot D + q_0 (D + \sum_{i=0}^{n-2} t_i^{(n)} 2^i - 2^{n-1} t_{n-1}^{(n)} - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^{n-1} r_{n-1}^{(n-1)}) \\ &+ \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)} \end{aligned} \tag{21}$$

Where $T^{(n)} = \sum_{i=0}^{n-2} t_i^{(n)} 2^i - t_{n-1}^{(n)} 2^{n-1}$, $R^{(n-1)} = \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + r_{n-1}^{(n-1)} 2^{n-1}$ and $D = \sum_{i=0}^{n-2} d_i 2^i$. According to [8], the backward rewriting of the polynomial $\sum_{i=0}^{n-2} t_i^{(n)} 2^i - t_{n-1}^{(n)} 2^{n-1}$ representing the output of an n-bit carry-ripple adder will result in an exponential blow-up of the polynomial size. Although, according to the predetermined topological order, the

backward rewriting should replace q_0 before $T^{(n)}$, the current analysis replaces $T^{(n)}$ before q_0. $T^{(n)}$'s most significant bit $t_{n-1}^{(n)}$ is nothing but the sum bit of the most significant adder FA_{n-1} of $Stage_n^{SUB}$. The backward rewriting expresses $t_{n-1}^{(n)}$ as following.

$$\begin{aligned}\overline{d}_{n-1} \oplus r_{n-1}^{(n-1)} \oplus c_{n-2}^{(n)} = 1 - d_{n-1} - r_{n-1}^{(n-1)} - c_{n-2}^{(n)} + 2r_{n-1}^{(n-1)}c_{n-2}^{(n)} + 2d_{n-1}r_{n-1}^{(n-1)} \\ + 2d_{n-1}c_{n-2}^{(n)} - 4d_{n-1}r_{n-1}^{(n-1)}c_{n-2}^{(n)}\end{aligned} \tag{22}$$

As $d_{n-1} = 0$ for an unsigned integer restoring divider, the expression for $t_{n-1}^{(n)}$ simplifies to $1 - r_{n-1}^{(n-1)} - c_{n-2}^{(n)} + 2r_{n-1}^{(n-1)}c_{n-2}^{(n)}$. The backward rewriting substitutes $t_{n-1}^{(n)} = 1 - r_{n-1}^{(n-1)} - c_{n-2}^{(n)} + 2r_{n-1}^{(n-1)}c_{n-2}^{(n)}$ in Equation (21).

$$\begin{aligned}\sum_{i=1}^{n-1} q_i 2^i \cdot D + q_0(D + \sum_{i=0}^{n-2} t_i^{(n)} 2^i + 2^{n-1}c_{n-2}^{(n)} - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^n r_{n-1}^{(n-1)} \\ - 2^n r_{n-1}^{(n-1)} c_{n-2}^{(n)} - 2^{(n-1)}) + \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)}\end{aligned} \tag{23}$$

In Equation (23), $2^{n-2}t_{n-2}^{(n)} + 2^{n-1}c_{n-2}^{(n)}$ resembles a full adder equation (Equation (4)). Hence, backward rewriting replaces $2^{n-2}t_{n-2}^{(n)} + 2^{n-1}c_{n-2}^{(n)}$ with $2^{n-2}(1 - d_{n-2} + r_{n-2}^{(n-1)} + c_{n-3}^{(n)})$. Additionally, backward rewriting also replaces the term $c_{n-2}^{(n)}$ associated with the term $r_{n-1}^{(n-1)}$. The polynomial expression for the term $c_{n-2}^{(n)}$ is equivalent to $r_{n-2}^{(n-1)} - r_{n-2}^{(n-1)}d_{n-2} + [1 - r_{n-2}^{(n-1)} - d_{n-2} + 2r_{n-2}^{(n-1)}d_{n-2}]c_{n-3}^{(n)}$. For better readability, the term P_{n-1} denotes the polynomial for the carry bit $c_{n-2}^{(n)}$. Equation (23) transforms to the following:

$$\begin{aligned}\sum_{i=1}^{n-1} q_i 2^i \cdot D + q_0(D + \sum_{i=0}^{n-3} t_i^{(n)} 2^i - \sum_{i=0}^{n-3} r_i^{(n-1)} 2^i + 2^{n-2}c_{n-3}^{(n)} + 2^n r_{n-1}^{(n-1)} \\ - 2^{n-2}d_{n-2} - 2^n r_{n-1}^{(n-1)} P_{n-1} - 2^{n-2}) + \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)}\end{aligned} \tag{24}$$

The Backward rewriting replaces $2^{n-3}t_{n-3}^{(n)} + 2^{n-2}c_{n-3}^{(n)}$ with its full adder equation: $2^{n-3}(1 - d_{n-3} + r_{n-3}^{(n-1)} + c_{n-4}^{(n)})$ and $c_{n-3}^{(n)}$ present in P_{n-1} as $r_{n-3}^{(n-1)} - r_{n-3}^{(n-1)}d_{n-3} + [1 - r_{n-3}^{(n-1)} - d_{n-3} + 2r_{n-3}^{(n-1)}d_{n-3}]c_{n-4}^{(n)}$ to result in:

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + q_0(D + \sum_{i=0}^{n-4} t_i^{(n)} 2^i - \sum_{i=0}^{n-4} r_i^{(n-1)} 2^i + 2^{n-3} c_{n-4}^{(n)} + 2^n r_{n-1}^{(n-1)}$$
$$- \sum_{i=n-3}^{n-2} d_i 2^i - 2^n r_{n-1}^{(n-1)} P_{n-1} - 2^{n-3}) + \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)} \quad (25)$$

Similarly, backward rewriting replaces the remaining full adders with their respective full adder equation (Equation 4), and carry bit $c_i^{(n)}$, where $0 \leq i \leq n-4$, present in P_{n-1} as $r_i^{(n-1)} - r_i^{(n-1)} d_i + [1 - r_i^{(n-1)} - d_i + 2r_i^{(n-1)} d_i] c_{i-1}^{(n)}$ to result in following equation

$$\sum_{i=1}^{n-1} q_i 2^i D + q_0(D - \sum_{i=0}^{n-2} d_i 2^i + 2^n r_{n-1}^{(n-1)} - 2^n r_{n-1}^{(n-1)} P_{n-1})$$
$$+ \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)} \quad (26)$$

In the above equation, $D - \sum_{i=0}^{n-2} d_i 2^i$ simplifies to zero and yields $\sum_{i=1}^{n-1} q_i 2^i D + q_0(2^n r_{n-1}^{(n-1)} - 2^n r_{n-1}^{(n-1)} P_{n-1}) + \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)}$ as the final polynomial. From the insights of *Lemma 1*, *Lemma 2* in [16] and the above analysis, the polynomial for the carry bit $c_{n-2}^{(n)}$ can be constructed as follows: $P_1 = r_0^{(n-1)} - r_0^{(n-1)} d_0 + [1 - r_0^{(n-1)} - d_0 + 2r_0^{(n-1)} d_0] c_{-1}^{(n)}$ and $P_j = r_{j-1}^{(n-1)} - r_{j-1}^{(n-1)} d_{j-1} + [1 - r_{j-1}^{(n-1)} - d_{j-1} + 2r_{j-1}^{(n-1)} d_{j-1}] P_{j-1}$, for $j = 2, \ldots, n-1$. The size of the polynomial P_1 denoted as $|P_1|$ is equivalent to: $|P_1| = 2^1 + 2^2$. The size of the polynomial P_j denoted as $|P_j|$ is given as follows.

$$|P_j| = 2 + 2^2(\sum_{i=0}^{j-1} 2^{i+1}) = 8 \cdot 2^j - 6$$

Therefore, the size of the polynomial $|P_{n-1}|$ representing the carry bit $c_{n-2}^{(n)}$ is equal to $2^{n+2} - 6$. Thus, the final polynomial $\sum_{i=1}^{n-1} q_i 2^i \cdot D + q_0(2^n r_{n-1}^{(n-1)} - 2^n r_{n-1}^{(n-1)} P_{n-1}) + \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)} - R^{(0)}$ has an exponential size. The key observation to make here is that even before backward rewriting replaces q_0, the polynomial's size increases exponentially due to the presence of the term $r_{n-1}^{(n-1)} c_{n-2}^{(n)}$, as replacement of $c_{n-2}^{(n)}$ would give rise to an exponential number of terms. The backward rewriting does not hit the expected polynomial after

replacing $Stage_n^{SUB}$ mainly because the implementation of $Stage_n^{SUB}$ shown in Figure 8 is not truly a signed adder or subtractor stage. $Stage_j^{SUB}$ is a signed adder only if the implementation imposes the input constraint $0 \leq R^{(0)} < D \cdot 2^{n-1}$ on the inputs of the reduced restoring divider. This information is currently not used during backward rewriting, leading to an exponential blow-up of the intermediate polynomial. The analysis in the current section offers a clear picture of the limitations of the naive backward rewriting process that fails to use the information from the input constraint that could help in preventing the exponential blow-up of the intermediate polynomial. The optimized reduced-restoring divider in the current work requires optimized backward rewriting to facilitate computationally inexpensive verification of this divider. The subsequent sections analyze in detail the verification of the reduced restoring divider with optimized backward rewriting.

For a correctly implemented clean divider without any optimizations, the backward rewriting would often reduce the specification polynomial to zero. However, this is not always the case for an optimized divider. Hence, it is crucial to verify that the final polynomial evaluates to zero for all inputs in the allowed input range $0 < R^{(0)} < D \cdot 2^{n-1}$ [3].

5.4 Don't Care Optimization

The current section discusses one of the optimization techniques used to improve the backward rewriting of the reduced restoring divider. The fundamental idea is to optimize the polynomial P during backward rewriting, which involves computing a smaller polynomial $P^{'}$ from the the current polynomial P using satisfiability don't cares. Don't cares are those signal assignments amongst other assignments that cannot occur at the inputs of atomic blocks due to input constraint $0 \leq R^{(0)} < D \cdot 2^{n-1}$. Thus, using don't care assignments to derive the smaller polynomial $P^{'}$ would not produce an incorrect result because $P^{'}$ evaluates to the same values as P for all the input combinations that do not satisfy the don't care assignments [3]. For the same reason, don't care assignments can be realized as polynomials that can be multiplied with any integer variable v and added to the polynomial P without altering the functionality represented by the polynomial P. The aim of adding such don't care polynomials to the original polynomial is to assign values to the integer variables associated with the don't care polynomial such that it eliminates or reduces as many terms as possible in the existing polynomial, thereby decreasing its

overall size. For example, consider a polynomial $P(a, b, c) = a + 2b - 3c + 4ab - 5bc$ with $(a, b) = (1, 1)$ as a don't care cube at the input of some arbitrary atomic block. Since the monomial $m_1 = v_1 * a * b$ would reduce to zero for all care sets, the monomial m_1, where v_1 is an arbitrary integer variable, can be added to the polynomial $P(a, b, c)$. As a result of which, polynomial transforms to $P(a, b, c) = a + 2b - 3c + 4ab - 5bc + v_1ab$. By combining similar terms, the polynomial simplifies to $a + 2b - 3c + ab(v_1 + 4) - 5bc$. Assigning the value of the variable $v_1 = -4$ reduces the polynomial size while preserving its functionality. In the above example, the polynomial optimization seems quite simple; however, the intermediate polynomial can be relatively large with multiple don't care cubes for large divider circuits. Hence, it is challenging to find the correct solutions for the integer variables associated with the don't care cubes, which is vital for the optimsation of the polynomial. Nevertheless, progress in integer linear programming overcomes the challenge to a certain extent by reducing the optimization problem to an integer linear programming problem.

Extending the current optimization technique to include splitting of the don't care cube improves the polynomial's optimization. If any term x is associated with an existing don't care (*pure don't care*) polynomial, then the entire association is considered a don't care polynomial or split dont' care cubes. This phenomenon is called splitting of don't care. Consider a don't care cube $(a, b) = (1, 0)$ that translates to the polynomial $m = a - ab$. The term or signal x associated with m gives rise to a new polynomial m', which is also a don't care regardless of the assignment of the signal x; that is, it is invariant with respect to $x = 1$ or $x = 0$. The overall association is still a don't care because $a - ab$ is evaluated to zero for all the care sets, making any association with it also a don't care. To understand how don't care splitting aids the polynomial's optimization, consider a polynomial $K(a, b, c, d) = 3ad + 2b - 8cb + 5ac - 5abc + a - ab$ with a don't care cube $dc_1 = a - ab$. Since the term c is associated with dc_1 in the polynomial K, instead of just adding $v_1(a - ab)$, don't care splitting is employed to add $v_1(a - ab)c + v_2(a - ab)(1 - c)$ to the polynomial K. The addition of don't care cubes to the polynomial K results in $K(a, b, c, d) = 3ad + 2b - 8cb + ac(v_1 - v_2 + 5) - abc(v_1 - v_2 + 5) + a(v_2 + 1) - ab(v_2 + 1)$. The solution $v_1 = -6$ and $v_2 = -1$ reduces the polynomial $K(a, b, c, d) = 3ad + 2b - 8cb$. Overall, the polynomial reduces from six terms to three terms. The provided example is a best-case scenario of a polynomial optimization, which might not always be true.

5.4.1 Computation of Satisfiability Don't Care

The current subsection provides an overview of the computation of satisfiability don't cares. The initial step involves identifying non-trivial atomic blocks (Full-adders, Half-adders, Multiplexers) and trivial atomic blocks (gates, buffers not included in non-trivial atomic blocks), followed by establishing a topological order$\prec_{\text{top}}$ on these atomic blocks [3]. The next step involves procuring don't care candidates at the inputs of non-trivial atomic blocks using simulation vectors that obey the input constraint $0 \leq R^{(0)} < D \cdot 2^{n-1}$.

Since simulations are used to obtain the don't care candidates, there is a possibility that when the simulation vector is not big enough for a given size of the divider circuit, some of the don't care candidates are not true don't care cubes. According to [3], SAT fails to identify enough true don't care cubes for larger divider circuits, mainly due to lack of resources. However, BDD-based image computation efficiently identifies all the true don't care cubes up to a certain circuit size [3]. An atomic block is a combinational circuit that translates to a Boolean function $f : A \longrightarrow B$, where $A \in \{0,1\}^n$ is the domain and $B \in \{0,1\}^m$ is the codomain of the function representing the atomic block. For a subset $C \subseteq A$,the image of C under the function f is a subset $B^{'}$ of co-domain B, that is ,$B^{'} = \{f(x)|x \in c\}$ and $B' \subseteq B$.

Consider a half adder with inputs x,y and outputs f_1, f_2, where f_1 and f_2 represent the sum and carry-out respectively. The function $H_A : \{0,1\}^2 \longrightarrow \{0,1\}^2$ with $f_1 = x \oplus y$ and $f_2 = x \wedge y$ represents a half adder. Table 2 provides the truth table of the half-adder mentioned above.

x	y	$f_1 = x \oplus y$	$f_2 = x \wedge y$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 2: Truth Table for half Adder

Table 2 helps to deduce the input space $A = [(0,0),(0,1),(1,0),(1,1)]$ and the output space $B = [(0,0),(0,1),(1,0)]$ of the function H_A. If, due to some constraint, the

assignment $(x, y) \neq (0, 0)$, then the input space confines to the last three rows of Table 2. The last three rows of Table 2 now represent the subset $C = [(0, 1), (1, 0), (1, 1)]$ of the original input space A. The image of C under the function H_A, also denoted $img(H_A, C)$, is a subset $B^{'} = [(0, 1), (1, 0)]$ of the original output space B. The Boolean expression $(\overline{f_1} \wedge f_2) \vee (f_1 \wedge \overline{f_2})$ derived from the truth table (Table 2), describes $img(H_A, C)$.

The truth table-based image computation requires iterating through every possible input combination, which is inefficient for large divider circuits. The BDD-based image computation is more efficient than the truth table-based image computation. Given a BDD ($bdd_{(f)}(\vec{X}, \vec{Y})$) that represents a Boolean function f with input $\vec{X}$, output $\vec{Y}$, and a BDD ($bdd_{(C)}(\vec{X})$) that represents the set $C(\vec{X})$, the image of C under f ($img(f, C)$) can be obtained by computing the AND operation between $bdd_{(f)}(\vec{X}, \vec{Y})$ and $bdd_{(C)}(\vec{X})$ followed by existential quantification with respect to the input variables $\vec{X}$. In other words, $img(f, C) = \exists_{\vec{X}}[bdd_{(f)}(\vec{X}, \vec{Y}) \wedge bdd_{(C)}(\vec{X})]$. Existential quantification removes the input variable without changing the satisfiability of the function, resulting in a BDD expressed in terms of the output variables. [18]

The above discussion provides a generic overview of BDD-based image computation. However, other optimized BDD-based image computation methods, such as image computation with generalised co-factor and functional image computation [19], are more efficient than the generic method.

Now that the dc candidates at the inputs of the topologically ordered atomic blocks are available, the selection of true don't care cubes among the available candidates begins with constructing a BDD that represents the input constraint (IC) $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The BDD representing the input constraint is analogous to ($bdd_{(C)}(\vec{X})$) in the previous example. The first atomic block a_i with a non-empty set of don't care candidates is selected using the predetermined topological order. The slice sl_1 associated with the selected atomic block a_i is determined. The slice sl_1 constitutes of atomic blocks $a_1, a_2, \ldots a_{i-1}$ whose output signals connect the slice sl_1 with the atomic blocks $a_i, a_{i+1}, \ldots a_m$, that is, by design ,the output of sl_1 is the input to the atomic block a_i. The true don't care cubes present at the inputs of a_i can be determined by computing the image IMG_1 of IC under sl_1. IMG_1 represents the possible output space of the atomic block a_i , when subjected to the input constraint IC. To identify the true don't care cubes among the don't care (dc) candidates at the input of a_i, IMG_1 is co-factorized with respect to the dc candidates;

if the evaluation results in constant 0, the dc candidate is a true don't care cube. The procedure uses the topological queue to select the next atomic block a_j with a non-empty set of don't care candidates. Slice sl_2 for the a_j is determined, and image IMG_2 of IMG_1 under sl_2 is computed, followed by evaluation of IMG_2 with the don't care candidates at the inputs of a_j to confirm the true don't care cubes. The process evaluates every atomic block with don't care candidates and terminates once it has exhausted all the atomic blocks with a non-empty set of don't care candidates. In the end, image computation successfully and efficiently identifies all the true don't care cubes at inputs of the atomic blocks present in the divider circuit. The BDD-based image computation not only helps to identify the true don't care cubes at the inputs of the atomic blocks but also helps verify the condition $0 \leq R < D$, discussed in Section 6.

5.4.2 Optimization with Integer Linear Programming

The current work illustrates how *Integer Linear Programming* (ILP) optimizes the polynomial efficiently. The example (see Section 5.4) illustrating the splitting of don't care cubes to optimize the polynomial $K(a, b, c, d)$ gives rise to the system of linear equations $< v_1 - v_2 + 5 = 0 >$ and $< v_2 + 1 = 0 >$. Satisfying these equations by assigning $v_1 = -6$ and $v_2 = -1$ reduces the number of terms in the polynomial K, thus optimizing the polynomial. The problem of satisfying a maximum number of linear integer equations reduces to integer linear programming by replacing each equation $l_i(x_1, x_2 \ldots x_n)$ with the following equations [3].

$$l_i(x_1, x_2 \ldots x_n) \leq M \cdot d_i$$
$$l_i(x_1, x_2 \ldots x_n) \geq -M \cdot d_i$$

Here, M is a sufficiently large constant and d_i is a binary variable also known as the deactivation variable. $d_i = 1$ indicates the corresponding equation is active, whereas $d_i = 0$ indicates that the corresponding equation is inactive. The integer linear programming aims to minimize the number of deactivation variables assigned to one. Consequently, obtaining a polynomial with as few terms as possible. The current work uses the Gurobi ILP solver to solve the system of equations.

The following steps summarize the optimization of a polynomial $P(x_1, x_2...x_n)$ using don't care cubes.

1) For every don't cube dc_i $(1 \leq i \leq n)$, associate an integer variable v_i.

2)Add the transformed don't care cubes $v_i * dc_i$ to the polynomial P.

3) Simply the polynomial by combining similar monomials.

4)Use ILP to reduce the size of the polynomial P.

5.5 Proof of Don't Care Cubes in a Reduced Restoring Divider

The previous section briefly illustrates the principle of don't care optimization and its role in minimizing the size of the intermediate polynomial. Before proceeding to the actual analysis of the reduced restoring divider circuit using don't care optimization, the current section presents the proof for the existence of don't care cubes in a reduced restoring divider. Since the analysis of the reduced restoring divider presented in the current work focuses on $Stage_n$, the proofs illustrated in this section mainly justify the existence of the don't care cubes at the inputs of the atomic blocks present in $Stage_n$. Based on the simulations run on 4-bit, 8-bit,16-bit, up to 128-bit width reduced restoring divider, four don't care candidates are present at the inputs of four different atomic blocks in $Stage_n$. The same scenario exists in $Stage_2$ to $Stage_n$; therefore, similar proofs should hold for all stages of the reduced restoring divider.

Proof 1: To prove that $(t_{n-1}^{(n)}, r_{n-1}^{(n-1)}) = (1, 1)$ *is a don't care cube at the inputs of atomic block* MUX_{n-1} *present in* $Stage_n^{MUX}$

From Lemma 1 (Section 3.10), for all the partial remainders $R^{(j)}$ $(1 \leq j \leq n)$ of a reduced restoring divider with the input constraint $0 \leq R^{(0)} < D2^{n-1}$, the following holds:

$$0 \leq R^{(j)} < D \cdot 2^{n-j} \tag{27}$$

From Equation (27), the partial reminder at any stage must be a positive integer. $R^{(n)}$ is the final remainder , which is expressed as $\sum_{i=0}^{n-2} r_i^{(n)} 2^i - 2^{n-1} r_{n-1}^{(n)}$. The output of MUX_{n-1} is $r_{n-1}^{(n)}$, where $r_{n-1}^{(n)}$ is a signed bit in 2's complement representation. Therefore, it is sufficient to check if $R^{(n)} \geq 0$, as $r_{n-1}^{(n)} = 1$ would make $R^{(n)}$ negative, which then violates Equation (27). If $(t_{n-1}^{(n)}, r_{n-1}^{(n-1)}) = (1,1)$, then $r_{n-1}^{(n)} = 1$ (from construction of divider circuit) regardless of $q_0 = 0$ or 1, making $R^{(n)} < 0$. Hence, $(t_{n-1}^{(n)}, r_{n-1}^{(n-1)}) = (1,1)$ is true don't care assignment at the inputs of atomic block MUX_{n-1}.

Proof 2: To prove that $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}) = (1,1)$ *is don't care cube at the inputs of atomic block* FA_{n-1} *present in* $Stage_n^{SUB}$.

Since $\overline{d}_{n-1} = 1$ ($\because$ unsigned integer divider), proving $(t_{n-1}^{(n)}, c_{n-1}^{(n)}) \neq (1,1)$ implies $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}) = (1,1)$ is a true don't care. $(t_{n-1}^{(n)}, c_{n-1}^{(n)}) = (1,1)$ also implies $R^{(n-1)} \geq D + 2^{n-1}$. To prove this implication, consider the following equation derived from the carry ripple adder, which realizes the $Stage_n^{SUB}$.

$$2^n c_{n-1}^{(n)} + \sum_{i=0}^{n-1} t_i^{(n)} 2^i = \sum_{i=0}^{n-1} r_i^{(n-1)} 2^i + \sum_{i=0}^{n-1} \overline{d}_i 2^i + c_{-1} \tag{28}$$

Substituting $(t_{n-1}^{(n)}, c_{n-1}^{(n)}) = (1,1)$ and disregarding the values of the terms with index $i = 0 : n-2$ in $\sum_{i=0}^{n-1} t_i^{(n)} 2^i$, transforms the above equation to the following

$$2^n + 2^{n-1} \leq \sum_{i=0}^{n-1} r_i^{(n-1)} 2^i + \sum_{i=0}^{n-1} \overline{d}_i 2^i + c_{-1} \tag{29}$$

Substituting $c_{-1} = 1$, $R^{(n-1)} = \sum_{i=0}^{n-1} r_i^{(n-1)} 2^i$(since $r_n^{(n-1)} = 0$), $\overline{d}_i = 1 - d_i$ and $D = \sum_{i=0}^{n-1} d_i 2^i$ simplifies the above equation to the following.

$$2^n + 2^{n-1} \leq R^{(n-1)} - D + 2^n \tag{30}$$

Equation (30) illustrates $R^{(n-1)} \geq D + 2^{n-1}$ when $(t_{n-1}^{(n)}, c_{n-1}^{(n)})$ = (1,1). From Equation (27), the constraint $R^{(n-1)} < 2 \cdot D$ must hold. if $R^{(n-1)} \geq D + 2^{n-1}$, then $D + 2^{n-1} < 2 \cdot D$, which makes $D > 2^{n-1}$. This result contradicts the constraint: $1 \leq D \leq 2^{n-1} - 1$ for an unsigned integer divisor. Therefore, as long as $R^{(n-1)}$ obeys $0 \leq R^{n-1} < 2 \cdot D$, $(t_{n-1}^{(n)}, c_{n-1}^{(n)}) \neq (1,1)$. $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}) = (1,1)$ is the only combination that gives rise to $(t_{n-1}^{(n)}, c_{n-1}^{(n)}) = (1,1)$. Hence $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}) = (1,1)$ is a true don't care.

Proof 3: To prove $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,1)$ *is a don't care cube present at the inputs of atomic block* FA_{n-2} *present in* $Stage_n^{SUB}$.

Before proving $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,1)$ is a don't care cube, the current proof establishes the following prerequisite.

Lemma 2: n-bit width $Stage_n^{SUB}$ can be pruned to $n-1$ bit width $Stage_n^{sub}$ when $d_{n-2} = 0$.

The above lemma holds due to the implication $\overline{d}_{n-2} \implies \overline{r}_{n-1}^{(n-1)}$. Partial remainder $R^{(n-1)}$ is expressed as $\sum_{i=0}^{n-1} r_i^{(n-1)} 2^i - 2^n r_n^{(n-1)}$ and divisor D is expressed as $\sum_{i=0}^{n-1} d_i 2^i$. Now, assume that $r_{n-1}^{(n-1)} = 1$ with all other terms in $R^{(n-1)}$ equal to zero and similarly, all the terms: i = 0 to n-3 is equal to one in D. Since d_{n-1} and d_{n-2} are equal to zero, it is clear from the assumption that $R^{(n-1)} > 2D$ ($\because 2^{n-1} r_{n-1}^{(n-1)} > \sum_{i=0}^{n-3} d_i 2^i$). Hence, to satisfy the constraint $R^{(n-1)} < 2 \cdot D$ (from Equation (27)), bit $r_{n-1}^{(n-1)}$ should be equal to zero when $d_{n-2} = 0$. As $r_n^{(n-1)} = 0$ ($\because R^{(n-1)} \geq 0$) and $r_{n-1}^{(n-1)} = 0$ ($\because \overline{d}_{n-2} \implies \overline{r}_{n-1}^{(n-1)}$), the partial remainder $R^{(n-1)}$ can be expressed as $\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i$. Since $n-1$ bits are required to represent the partial remainder $R^{(n-1)}$, which is also one of the inputs to $Stage_n^{SUB}$, $Stage_n^{SUB}$ can also be reduced to the width of $n-1$.

Using Lemma 2, the underlying principle of Proof 2 can also be applied to prove $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,1)$ is a don't care cube present at the inputs of FA_{n-2}. For $\overline{d}_{n-2} = 1/d_{n-2} = 0$ to be true, the following condition must hold:

$$1 \leq D \leq 2^{n-2} - 1 \tag{31}$$

$(t_{n-2}^{(n)}, c_{n-2}^{(n)}) = (1,1)$ implies $R^{(n-1)} \geq D + 2^{n-2}$. To prove this implication, consider the following equation derived from the reduced carry ripple adder of width n-1.

$$2^{n-1}c_{n-2}^{(n)} + \sum_{i=0}^{n-2} t_i^{(n)} 2^i = \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + \sum_{i=0}^{n-2} \overline{d}_i 2^i + c_{-1} \tag{32}$$

Substituting $(t_{n-2}^{(n)}, c_{n-2}^{(n)})$ = (1,1), $c_{-1} = 1$, $R^{(n-1)} = \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i$ ($\because r_n^{(n-1)}, r_{n-1}^{(n-1)} = 0$), $\overline{d}_i = 1 - d_i$, $D = \sum_{i=0}^{n-2} d_i 2^i$ and disregarding the values of the terms i = 0: n-3 in $\sum_{i=0}^{n-2} t_i^{(n)} 2^i$ in the above equation, results in the following.

$$2^{n-1} + 2^{n-2} \leq R^{(n-1)} - D + 2^{n-1} \tag{33}$$

The above equation simplifies to $D + 2^{n-2} \leq R^{(n-1)}$. From Equation (27), it is clear that $R^{(n-1)} < 2 \cdot D$. if $R^{(n-1)} \geq D + 2^{n-2}$, then $D + 2^{n-2} < 2 \cdot D$ making $D > 2^{n-2}$. This result contradicts Equation (31). Therefore, as long as $R^{(n-1)}$ obeys $0 \leq R^{(n-1)} < 2 \cdot D$, $(t_{n-2}^{(n)}, c_{n-2}^{(n)}) \neq (1,1)$. $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,1)$ is the only combination that gives rise to $(t_{n-2}^{(n)}, c_{n-2}^{(n)}) = (1,1)$. Hence, it can be concluded that $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,1)$ is a true don't care.

Proof 4: To prove that $(t_{n-2}^{(n)}, q_0, r_{n-2}^{(n-1)})$*=(0,0,1) is a don't care cube at the inputs of atomic block* MUX_{n-2} *present in* $Stage_n^{MUX}$.

There are four possible combinations $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (0,0,0), (0,1,1), (1,0,1), (1,1,0)$ at the inputs of FA_{n-2} that would result in $t_{n-2}^{(n)} = 0$. Among these four combinations, only two combinations have $r_{n-2}^{(n-1)} = 1$, which is of interest in the current proof. Hence, the four combinations reduces to two cases:

Case 1 : $(\overline{d_{n-2}}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,0)$

Case 2 : $(\overline{d_{n-2}}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (0,1,1)$

Both cases will result in $c_{n-2}^{(n)} = 1$, which makes $r_{n-1}^{(n-1)} = 0$ (due to Proof 2) and this translates to $t_{n-1}^{(n)} = 0$ and $q_0 = 1$ ($\because$ circuit design). Hence $q_0 \neq 0$ when $t_{n-2}^{(n)} = 0$

and $r_{n-2}^{(n-1)} = 1$. Therefore $(t_{n-2}^{(n)}, q_0, r_{n-2}^{(n-1)})$=(0,0,1) is a don't care cube at the inputs of MUX_{n-2}.

Although the above proofs justify the presence of the mentioned don't care cubes at the inputs of the atomic blocks, it is also crucial to demonstrate these are the only don't care cubes available at the inputs of the atomic blocks present in $Stage_n$. The following proofs confirm this by illustrating the possible care sets at the inputs of the atomic block present in $Stage_n$. The current section introduces a concise version of the restoring divider algorithm, which simplifies the justification of the following proofs.

Algorithm 2 Restoring division version 2

1: **for** $j = 1$ **to** n **do**

$$q_{n-j} = \begin{cases} 1 & \text{if } R^{(j-1)} \geq D \cdot 2^{n-j}, \\ 0 & \text{if } R^{(j-1)} < D \cdot 2^{n-j}. \end{cases}$$

$$R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j};$$

2: **end for**
3: $R := R^{(n)}$;

Proof 5: To prove that at the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (0, 0, 0)$, and at inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0, 0, 0)$. The inputs of atomic block FA_{n-1} has the combination $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}, \overline{d}_{n-1}) = (0, 0, 1)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (0, 1, 0)$, MUX_{n-1} has the combination $(r_{n-1}^{(n-1)}, t_{n-1}^{(n)}, q_0) = (0, 1, 0)$, and MUX_0 has the care combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0, 1, 0)$.

The input assignment $R^{(0)} = 0$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 0$ holds for $1 \leq j \leq n$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 0) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2, it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 0$. Assume that the condition holds for $j-1$, that is , $R^{(j-1)} = 0$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 0$ and $(R^{(j-1)} = 0) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 0$. When $j = n-1$, $R^{(n-1)} = 0$. The

following equation describes the carry-ripple adder of $Stage_n^{SUB}$.

$$2^n c_{n-1}^{(n)} + \sum_{i=0}^{n-1} t_i^{(n)} 2^i = \sum_{i=0}^{n-1} r_i^{(n-1)} 2^i + \sum_{i=0}^{n-1} \overline{d}_i 2^i + (c_{-1} = 1) \tag{34}$$

Side Remark: In the current and the following proofs: $\overline{D} = \sum_{i=0}^{n-1} \overline{d}_i 2^i$ represents the negated version of D, $\overline{d}_{n-1} = 1$ and $q_0 = \overline{t}_{n-1}^{(n)}$ ($\because$ circuit design). For $D = 2^{n-1} - 1$ to be true, all bits with index 0 to $n-2$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$, which is equal to 1. As $R^{(n-1)} = 0$, all bits in $R^{(n-1)}$ are equal to zero. According to Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$, where $0 \leq i \leq n-2$, is equal to zero, thereby, $q_0 = 0$. Therefore , the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i)$ = (0,0,0), and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0, 0, 0)$. The inputs of atomic block FA_{n-1} has the combination $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}, \overline{d}_{n-1}) = (0, 0, 1)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (0, 1, 0)$, MUX_{n-1} has the combination $(r_{n-1}^{(n-1)}, t_{n-1}^{(n)}, q_0) = (0, 1, 0)$, and MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0, 1, 0)$.

Proof 6: To prove that the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i)$ = (0,0,1), and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the care combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0, 1, 0)$.

The input assignment $R^{(0)} = 0$ and $D = 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 0$ holds for $1 \leq j \leq n$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 0) < (D \cdot 2^{n-1} = 1 \cdot 2^{n-1})$, from the restoring division algorithm 2, it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 0$. Assume that the condition holds for $j-1$, that is, $R^{(j-1)} = 0$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 0$ and $(R^{(j-1)} = 0) < (D \cdot 2^{n-j} = 1 \cdot 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 0$. When $j = n-1$, $R^{(n-1)} = 0$.

For $D = 1$, all bits with index i $(1 \leq i \leq n-1)$ in $\overline{D}$ are equal to one, except for $\overline{d}_0$, which is equal to zero. As $R^{(n-1)} = 0$, all bits in $R^{(n-1)}$ are equal to zero. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$, where $0 \leq i \leq n-2$, is equal to zero,

thereby, $q_0 = 0$. Therefore, the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (0, 0, 1)$ and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0, 1, 0)$.

Proof 7: To prove that the inputs of atomic block FA_i has the combination $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (1, 0, 1)$, where $1 \leq i \leq n-2$, and the inputs of the atomic block FA_k has the combination $(r_k^{(n-1)}, c_{k-1}^{(n)}, \overline{d}_k) = (0, 1, 0)$, where $k = i + 1$ and $2 \leq k \leq n-2$.

The input assignment $R^{(0)} = 2^i$ and $D = 2^{n-1} - (2^i + 1)$ results in the combination $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (1, 0, 1)$ at the input of the atomic block FA_i, where $1 \leq i \leq n-2$, and the combination $(r_k^{(n-1)}, c_{k-1}^{(n)}, \overline{d}_k) = (0, 1, 0)$ at the inputs of the atomic block FA_k, where $k = i + 1$ and $2 \leq k \leq n-2$. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 2^i$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 2^i) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1+i} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^i$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^i$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^i$ and $(R^{(j-1)} = 2^i) < (D \cdot 2^{n-j} = 2^{2n-j-1} - 2^{n-j+i} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^i$. When $j = n-1$, $R^{(n-1)} = 2^i$.

As the bit width of $R^{(n)}$ is equivalent to D, that is, both $R^{(n)}$ and D have a bit width of n, the bit $r_i^{(n-1)}$ in $R^{(n-1)}$ and $\overline{d}_i$ in $\overline{D}$ are the inputs to the atomic block FA_i of $Stage_n^{SUB}$ (see Figure 5). For $R^{(n-1)} = 2^i$, every bit in $R^{(n-1)}$ is equal to zero, except for r_i^{n-1}, which is equal to 1. Similarly for $D = 2^{n-1} - (2^i + 1)$, every bit in $\overline{D}$ is equal to zero, except for $\overline{d}_i$ and $\overline{d}_{n-1}$, which are equal to one. From Equation (34) and circuit design, $c_{i-1}^{(n)}$ is equal to zero, thus $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (1, 0, 1)$ at the input of the atomic block FA_i. However, $c_i^{(n)}$ is equal to one, which is the carry-in of the atomic block FA_k, where $k = i + 1$. As a result of this, the combination $(r_k^{(n-1)}, c_{k-1}^{(n)}, \overline{d}_k) = (0, 1, 0)$ is possible at the inputs of the atomic block FA_k.

Proof 8: To prove that the inputs of all atomic blocks FA_i ,where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i)$ = (0,1,1), and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, has the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0,0,1)$. The inputs of the atomic block FA_{n-1} has the combination $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}, \overline{d}_{n-1}) = (0,1,1)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (1,1,0)$ and MUX_{n-1} has the combination $(r_{n-1}^{(n-1)}, t_{n-1}^{(n)}, q_0) = (0,0,1)$, MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1,0,1)$.

The input assignment $R^{(0)} = 1$ and $D = 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 1$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 1) < (D \cdot 2^{n-1} = 1 \cdot 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 1$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 1$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 1$ and $(R^{(j-1)} = 1) < (D \cdot 2^{n-j} = 1 \cdot 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 1$. When $j = n-1$, $R^{(n-1)} = 1$.

For $D = 1$, every bit with index i $(1 \leq i \leq n-1)$ in $\overline{D}$ is equal to one, except for $\overline{d}_0$, which is equal to zero. As $R^{(n-1)} = 1$, every bit with index i $(1 \leq i \leq n-1)$ in $R^{(n-1)}$ is equal to zero, except for $r_0^{(n-1)}$, which is equal to one. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$, where $0 \leq i \leq n-2$, is equal to one, thereby, $q_0 = 1$. Therefore, the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (0,1,1)$ and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0,0,1)$. The input of atomic block FA_{n-1} has the combination $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}, \overline{d}_{n-1}) = (0,1,1)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (1,1,0)$, MUX_{n-1} has the combination $(r_{n-1}^{(n-1)}, t_{n-1}^{(n)}, q_0) = (0,0,1)$, and MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1,0,1)$.

Proof 9: To prove that the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i)$ = (1,0,0), and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1,1,0)$.

The input assignment $R^{(0)} = 2^{n-1} - 2$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 2^{n-1} - 2$ holds for $1 \leq j \leq n-1$. The proof

uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1}D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-1} - 2) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-1} - 2$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-1} - 2$. $R^{(j)} = R^{(j-1)} - q_{n-j}D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-1} - 2$ and $(R^{(j-1)} = 2^{n-1} - 2) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-1} - 2$. When $j = n - 1$, $R^{(n-1)} = 2^{n-1} - 2$.

For $D = 2^{n-1} - 1$, every bit with index i $(0 \leq i \leq n-2)$ in $\overline{D}$ is equal to zero, except for $\overline{d}_{n-1}$, which is equal to one. As $R^{(n-1)} = 2^{n-1} - 2$, every bit with index i $(1 \leq i \leq n-2)$ in $R^{(n-1)}$ is equal to one, except for $r_{n-1}^{(n-1)}$ and $r_0^{(n-1)}$, which are equal to zero. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$, where $0 \leq i \leq n-2$, is equal to zero, thereby, $q_0 = 0$. Therefore, the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$ have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (1, 0, 0)$ and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$ have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1, 1, 0)$.

Proof 10: To prove that the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i)$ = (1,1,0), and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1, 0, 1)$.

The input assignment $R^{(0)} = 2^{n-1} - 1$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$.The condition: $R^{(j)} = 2^{n-1} - 1$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1}D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-1} - 1) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-1} - 1$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-1} - 1$. $R^{(j)} = R^{(j-1)} - q_{n-j}D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-1} - 1$ and $(R^{(j-1)} = 2^{n-1} - 1) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-1} - 1$. When $j = n - 1$, $R^{(n-1)} = 2^{n-1} - 1$.

For $D = 2^{n-1} - 1$, every bit with index i $(0 \leq i \leq n-2)$ in $\overline{D}$ is equal to zero, except for $\overline{d}_{n-1}$, which is equal to one. As $R^{(n-1)} = 2^{n-1} - 1$, all bits with index i $(0 \leq i \leq n-2)$ in $R^{(n-1)}$ are equal to one, except $r_{n-1}^{(n-1)}$, which is equal to zero. From Equation (34) and

circuit design (Figure 7), every carry bit $c_i^{(n)}$, where $0 \leq i \leq n-2$, is equal to one, thereby, $q_0 = 1$. Therefore, the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-2$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (1,1,0)$ and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-2$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1,0,1)$.

Proof 11: To prove that the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-3$, have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i)$ = (1,1,1), and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-3$ have the care combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1,1,1)$. The inputs of atomic block FA_{n-2} has the combination $(r_{n-2}^{(n-1)}, c_{n-3}^{(n)}, \overline{d}_{n-2}) = (1,1,0)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (1,1,1)$, MUX_{n-2} has the combination $(r_{n-2}^{(n-1)}, t_{n-2}^{(n)}, q_0) = (1,0,1)$, and MUX_0 has the care combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1,1,1)$.

The input assignment $R^{(0)} = 2^{n-1} - 1$ and $D = 2^{n-2}$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 2^{n-1} - 1$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-1} - 1) < (D \cdot 2^{n-1} = 2^{2n-3})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-1} - 1$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-1} - 1$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-1} - 1$ and $(R^{(j-1)} = 2^{n-1} - 1) < (D \cdot 2^{n-j} = 2^{2n-2-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-1} - 1$. When $j = n-1$, $R^{(n-1)} = 2^{n-1} - 1$.

For $D = 2^{n-2}$, every bit with index i $(0 \leq i \leq n-3)$ in $\overline{D}$ is equal to one, except for $\overline{d}_{n-2}$, which is equal to zero. As $R^{(n-1)} = 2^{n-1} - 1$, every bit with index i $(0 \leq i \leq n-2)$ in $R^{(n-1)}$ is equal to one, except $r_{n-1}^{(n-1)}$, which is equal to zero. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$, where $0 \leq i \leq n-2$, is equal to one, thereby, $q_0 = 1$. Therefore, the inputs of all atomic blocks FA_i, where $1 \leq i \leq n-3$ have the combination: $(r_i^{(n-1)}, c_{i-1}^{(n)}, \overline{d}_i) = (1,1,1)$, and the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-3$ have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1,1,1)$. The inputs of atomic block FA_{n-2} has the combination $(r_{n-2}^{(n-1)}, c_{n-3}^{(n)}, \overline{d}_{n-2}) = (1,1,0)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (1,1,1)$, MUX_{n-2} has the combination $(r_{n-2}^{(n-1)}, t_{n-2}^{(n)}, q_0) = (1,0,1)$, and MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1,1,1)$.

Proof 12: To prove that the inputs of atomic block FA_{n-1} has the combination $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}, \overline{d}_{n-1}) = (1,0,1)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (0,1,1)$ and MUX_{n-1} has the combination $(r_{n-1}^{(n-1)}, t_{n-1}^{(n)}, q_0) = (1,0,1)$, MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0,0,1)$.

The input assignment $R^{(0)} = 2^{n-1}$ and $D = 2^{n-1} - 2$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$.The condition: $R^{(j)} = 2^{n-1}$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-1}) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^n)$,from the restoring division algorithm 2 it is clear that$q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-1}$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-1}$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-1}$ and $(R^{(j-1)} = 2^{n-1}) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j+1})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-1}$. When $j = n-1$, $R^{(n-1)} = 2^{n-1}$.

For $D = 2^{n-1} - 2$ to be true, every bit with index i $(1 \leq i \leq n-2)$ in $\overline{D}$ is equal to zero, except for $\overline{d}_{n-1}$ and $\overline{d}_0$, which are equal to one. As $R^{(n-1)} = 2^{n-1}$, all bits with index i $(0 \leq i \leq n-2)$ in $R^{(n-1)}$ are equal to zero, except $r_{n-1}^{(n-1)}$, which is equal to one. From Equation (34) and circuit design (Figure 7), every carry-bit $c_i^{(n)}$, where $1 \leq i \leq n-2$, is equal to zero and q_0 is equal to one. Therefore, the inputs of the atomic block FA_{n-1} has the combination $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}, \overline{d}_{n-1}) = (1,0,1)$, FA_0 has the combination $(r_0^{(n-1)}, c_{-1}^{(n)}, \overline{d}_0) = (0,1,1)$ and MUX_{n-1} has the combination $(r_{n-1}^{(n-1)}, t_{n-1}^{(n)}, q_0) = (1,0,1)$, MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0,0,1)$.

Proof 13: To prove that the inputs of atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0,0,0)$ and inputs of atomic block FA_1 has the combination $(r_1^{(n-1)}, c_0^{(n)}$, $\overline{d}_1) = (0,1,0)$.

The input assignment $R^{(0)} = 0$ and $D = 2^{n-1} - 2$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 0$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$.As $(R^{(0)} = 0) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^n)$, from the

restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 0$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 0$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 0$ and $(R^{(j-1)} = 0) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^n)$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 0$. When $j = n-1$, $R^{(n-1)} = 0$.

For $D = 2^{n-1} - 2$ to be true, every bit with index i $(1 \leq i \leq n-2)$ in $\overline{D}$ is equal to zero, except for $\overline{d}_{n-1}$ and $\overline{d}_0$, which are equal to one. As $R^{(n-1)} = 0$, all bits with index i $(0 \leq i \leq n-1)$ in $R^{(n-1)}$ are equal to zero. From Equation (34) and circuit design (Figure 7), $c_0^{(n)}$ is equal to one and $q_0 = 0$. Therefore, the inputs of the atomic block FA_1 has the combination $(r_1^{(n-1)}, c_0^{(n)}, \overline{d}_1) = (0, 1, 0)$ and the atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0, 0, 0)$.

Proof 14: To prove that the inputs of the atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0, 1, 1)$.

The input assignment $R^{(0)} = 2^{n-1}$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 2^{n-1}$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-1}) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-1}$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-1}$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-1}$ and $(R^{(j-1)} = 2^{n-1}) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-1}$. When $j = n-1$, $R^{(n-1)} = 2^{n-1}$.

For $D = 2^{n-1} - 1$ to be true, all bits with index i $(0 \leq i \leq n-2)$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$, which is equal to one. As $R^{(n-1)} = 2^{n-1}$, all bits with index i $(0 \leq i \leq n-2)$ in $R^{(n-1)}$ are equal to zero, except for $r_{n-1}^{(n-1)}$, which is equal to one. From Equation (34) and circuit design (Figure 7), q_0 is equal to 1. Therefore, the inputs of the atomic block MUX_0 has the care combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (0, 1, 1)$.

Proof 15: To prove that the inputs of atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1, 0, 0)$.

The input assignment $R^{(0)} = 1$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 1$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 1) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 1$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 1$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 1$ and $(R^{(j-1)} = 1) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 1$. When $j = n-1$, $R^{(n-1)} = 1$.

For $D = 2^{n-1} - 1$ to be true, all bits with index i $(0 \leq i \leq n-2)$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$, which is equal to one. As $R^{(n-1)} = 1$, all bits with index i $(1 \leq i \leq n-1)$ in $R^{(n-1)}$ are equal to zero, except for $r_0^{(n-1)}$, which is equal to one. From Equation (34) and circuit design (Figure 7), q_0 is equal to zero. Therefore, the inputs of the atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1, 0, 0)$.

Proof 16: To prove that the inputs of the atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1, 1, 0)$.

The input assignment $R^{(0)} = 1$ and $D = 2^{n-1} - 2$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 1$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 1) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 1$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 1$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 1$ and $(R^{(j-1)} = 1) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 1$. When $j = n-1$, $R^{(n-1)} = 1$.

For $D = 2^{n-1} - 2$ to be true, all bits with index i $(1 \leq i \leq n-2)$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$ and $\overline{d}_0$, which are equal to one. As $R^{(n-1)} = 1$, all bits with index i

$(1 \leq i \leq n-1)$ in $R^{(n-1)}$ are equal to zero, except for $r_0^{(n-1)}$, which is equal to one. From Equation (34) and circuit design (Figure 7), q_0 is equal to zero. Therefore, the inputs of the atomic block MUX_0 has the combination $(r_0^{(n-1)}, t_0^{(n)}, q_0) = (1, 1, 0)$.

Proof 17: To prove that the inputs of the atomic block MUX_{n-2} has the combination $(r_{n-2}^{(n-1)}, t_{n-2}^{(n)}, q_0) = (0, 1, 1)$.

The input assignment $R^{(0)} = 3 \cdot 2^{n-2} - 1$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 3 \cdot 2^{n-2} - 1$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 3 \cdot 2^{n-2} - 1) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 3 \cdot 2^{n-2} - 1$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 3 \cdot 2^{n-2} - 1$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 3 \cdot 2^{n-2} - 1$ and $(R^{(j-1)} = 3 \cdot 2^{n-2} - 1) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 3 \cdot 2^{n-2} - 1$. When $j = n-1$, $R^{(n-1)} = 3 \cdot 2^{n-2} - 1$.

For $D = 2^{n-1} - 1$ to be true, all bits with index i $(0 \leq i \leq n-2)$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$, which is equal to one. As $R^{(n-1)} = 3 \cdot 2^{n-2} - 1$, all bits with index i $(0 \leq i \leq n-1)$ in $R^{(n-1)}$ are equal to one, except for $r_{n-2}^{(n-1)}$, which is equal to zero. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$ $(0 \leq i \leq n-3)$ is equal to one and $q_0 = 1$. Therefore, the inputs of atomic block MUX_{n-2} has the combination $(r_{n-2}^{(n-1)}, t_{n-2}^{(n)}, q_0) = (0, 1, 1)$.

Proof 18: To prove that the inputs of the atomic block MUX_{n-2} has the combination $(r_{n-2}^{(n-1)}, t_{n-2}^{(n)}, q_0) = (1, 1, 1)$.

The input assignment assignment $R^{(0)} = 3 \cdot 2^{n-2}$ and $D = 2^{n-1} - 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 3 \cdot 2^{n-2}$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1}D \cdot 2^{n-1}$. As $(R^{(0)} = 3 \cdot 2^{n-2}) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 3 \cdot 2^{n-2}$. Assume that for $j-1$ the condition holds, that is , $R^{(j-1)} = 3 \cdot 2^{n-2}$. $R^{(j)} = R^{(j-1)} - q_{n-j}D \cdot 2^{n-j}$, since $R^{(j-1)} = 3 \cdot 2^{n-2}$ and $(R^{(j-1)} = 3 \cdot 2^{n-2}) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 3 \cdot 2^{n-2}$. When $j = n-1$, $R^{(n-1)} = 3 \cdot 2^{n-2}$.

For $D = 2^{n-1} - 1$ to be true, all bits with index i $(0 \leq i \leq n-2)$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$, which is equal to one. As $R^{(n-1)} = 3 \cdot 2^{n-2}$, all bits with index i $(0 \leq i \leq n-3)$ in $R^{(n-1)}$ are equal to zero, except for $r_{n-1}^{(n-1)}$ and $r_{n-2}^{(n-1)}$, which are equal to one. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$ $(0 \leq i \leq n-2)$ is equal to zero and q_0 is equal to one. Therefore, the inputs of the atomic block MUX_{n-2} has the combination $(r_{n-2}^{(n-1)}, t_{n-2}^{(n)}, q_0) = (1, 1, 1)$.

Proof 19: To prove that the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-3$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0, 1, 1)$.

The input assignment assignment $R^{(0)} = 2^{n-1}$ and $D = 2^{n-2} + 1$ results in the combination discussed in the current proof. The current assignment obeys the input constraint $IC : 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 2^{n-1}$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1}D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-1}) < (D \cdot 2^{n-1} = 2^{2n-3} + 2^{n-1})$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-1}$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-1}$. $R^{(j)} = R^{(j-1)} - q_{n-j}D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-1}$ and $(R^{(j-1)} = 2^{n-1}) < (D \cdot 2^{n-j} = 2^{2n-2-j} + 2^{n-j})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-1}$. When $j = n-1$, $R^{(n-1)} = 2^{n-1}$.

For $D = 2^{n-2} + 1$ to be true, all bits with index i $(1 \leq i \leq n-3)$ in $\overline{D}$ are equal to one, except for $\overline{d}_{n-2}$ and $\overline{d}_0$, which are equal to zero. As $R^{(n-1)} = 2^{n-1}$, every bit in $R^{(n-1)}$ is equal to zero, except for $r_{n-1}^{(n-1)}$, which is equal to one. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$ $(0 \leq i \leq n-2)$ is equal to zero and q_0 is equal to one. Therefore, the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-3$ have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (0, 1, 1)$.

Proof 20: To prove that the inputs of all atomic blocks MUX_i , where $1 \leq i \leq n-3$, have the care combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1, 0, 0)$.

The input assignment $R^{(0)} = 2^{n-2} - 2$ and $D = 2^{n-1} - 2$ results in the combination discussed in the current proof. The current assignment obeys the input constraint IC : $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The condition: $R^{(j)} = 2^{n-2} - 2$ holds for $1 \leq j \leq n-1$. The proof uses induction to establish this condition.

For $j = 1$, $R^{(1)} = R^{(0)} - q_{n-1} D \cdot 2^{n-1}$. As $(R^{(0)} = 2^{n-2} - 2) < (D \cdot 2^{n-1} = 2^{2n-2} - 2^n)$, from the restoring division algorithm 2 it is clear that $q_{n-1} = 0$. Thus, it is true that $R^{(1)} = R^{(0)} = 2^{n-2} - 2$. Assume that for $j-1$ the condition holds, that is, $R^{(j-1)} = 2^{n-2} - 2$. $R^{(j)} = R^{(j-1)} - q_{n-j} D \cdot 2^{n-j}$, since $R^{(j-1)} = 2^{n-2} - 2$ and $(R^{(j-1)} = 2^{n-2} - 2) < (D \cdot 2^{n-j} = 2^{2n-1-j} - 2^{n-j+1})$, this results in $q_{n-j} = 0$. Therefore, $R^{(j)} = R^{(j-1)} = 2^{n-2} - 2$. When $j = n-1$, $R^{(n-1)} = 2^{n-2} - 2$.

For $D = 2^{n-1} - 2$ to be true, all bits with index i $(1 \leq i \leq n-2)$ in $\overline{D}$ are equal to zero, except for $\overline{d}_{n-1}$ and $\overline{d}_0$, which are equal to one. As $R^{(n-1)} = 2^{n-2} - 2$, every bit in $R^{(n-1)}$ is equal to zero, except for bits from index 1 to $n-3$, which are equal to one. From Equation (34) and circuit design (Figure 7), every carry bit $c_i^{(n)}$ $(0 \leq i \leq n-3)$ is equal to one and q_0 is equal to 0. Therefore, the inputs of all atomic blocks MUX_i, where $1 \leq i \leq n-3$, have the combination $(r_i^{(n-1)}, t_i^{(n)}, q_0) = (1, 0, 0)$.

The presence of the don't care cubes at the inputs of a few atomic blocks does not always avoid an exponential blow-up of the intermediate polynomial size. In some circuits, there aren't enough and required don't care cubes at the inputs of the atomic blocks. The lack of don't care cubes could hinder the minimization of the polynomial's size. The same scenario applies to the reduced restoring divider shown in this work. The following section illustrates an exponential blow-up of the intermediate polynomial size, even with the presence of don't care cubes.

5.6 Proof of the Exponential Size of Polynomial Representing the Reduced Restoring Divider

The current section presents a proof that extracts an exponential lower bound on the size of a co-factorized polynomial. The polynomial is co-factorized by setting the input variables $r_{n-1}^{(n-1)}$ and $r_{n-2}^{(n-1)}$ to zero. The exponential lower bound obtained on the size of the co-factorized polynomial must also hold for the size of the original polynomial as well because setting the input variables to zero would eliminate terms, thereby reducing the polynomial size. Setting the input variables to zero corroborates the goal of optimization of a polynomial, which aims to reduce the polynomial's size by minimizing the number of terms in a polynomial.

There are two don't care cubes $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}) = (1,1)$ and $(\overline{d}_{n-2}, r_{n-2}^{(n-1)}, c_{n-3}^{(n)}) = (1,1,1)$ at the inputs of atomic blocks FA_{n-1} and FA_{n-2}, respectively. Since the current proof sets $r_{n-1}^{(n-1)}$ and $r_{n-2}^{(n-1)}$ to zero, using the don't care cubes discussed here is ineffective. In other words, setting both $r_{n-1}^{(n-1)}$ and $r_{n-2}^{(n-1)}$ to zero maps the don't care monomials $v_1 r_{n-1}^{(n-1)} c_{n-2}^{(n)}$ and $v_2 \overline{d}_{n-2} r_{n-2}^{(n-1)} c_{n-3}^{(n)}$ to zero, thereby making the addition of these monomials to the original polynomial futile. This argument is also true for splits of don't care cubes $r_{n-1}^{(n-1)} c_{n-2}^{(n)}$ and $\overline{d}_{n-2} r_{n-2}^{(n-1)} c_{n-3}^{(n)}$, as the split cubes also contain the signals $r_{n-1}^{(n-1)}$ and $r_{n-2}^{(n-1)}$, which ultimately reduces the small cubes to zero. For the same reason as discussed above, the don't care cubes presented in Proofs 1 and 4 are also ineffective. The proof begins with the polynomial obtained after backward rewriting replaces $Stage_n^{MUX}$ with their corresponding equations (refer to Equation (21)):

$$\sum_{i=1}^{n-1} q_i 2^i D + [q_0 \cdot D + q_0 \cdot \sum_{i=0}^{n-2} t_i^{(n)} 2^i - 2^{n-1} t_{n-1}^{(n)} - q_0 \cdot R^{(n-1)}] + R^{(n-1)} - R^{(0)} \qquad (35)$$

where $T^{(n)} = \sum_{i=0}^{n-2} t_i^{(n)} 2^{(i)} - 2^{n-1} t_{n-1}^{(n)}$ and $R^{(n-1)} = \sum_{i=0}^{n-3} r_i^{(n-1)} 2^i$ ($\because r_n^{(n-1)} = r_{n-1}^{(n-1)} = r_{n-2}^{(n-1)} = 0$). In the current proof, the backward rewriting only considers replacing $T^{(n)}$; that is, the backward rewriting deviates from the topological order of replacing q_0 before replacing $T^{(n)}$. Since $\overline{d}_{n-1} = 1$ and $r_{n-1}^{(n-1)} = 0$, this results in $t_{n-1}^{(n)} = \overline{c}_{n-2}^{(n)}$. Substituting $t_{n-1}^{(n)} = 1 - c_{n-2}^{(n)}$ in the expression for $T^{(n)}$ results in $\sum_{i=0}^{n-2} t_i^{(n)} 2^i + 2^{n-1} c_{n-2}^{(n)} - 2^{(n-1)}$. The backward rewriting rewrites $c_{n-2}^{(n)}$ as:

$$c_{n-2}^{(n)} = [r_{n-2}^{(n-1)} - r_{n-2}^{(n-1)} d_{n-2} + c_{n-3}^{(n)} \\ - c_{n-3}^{(n)} r_{n-2}^{(n-1)} - c_{n-3}^{(n)} d_{n-2} + 2c_{n-3}^{(n)} r_{n-2}^{(n-1)} d_{n-2}] \tag{36}$$

Using the assumption $r_{n-2}^{(n-1)} = 0$, the above expression for $c_{n-2}^{(n)}$ simplifies to

$$c_{n-2}^{(n)} = c_{n-3}^{(n-1)} - c_{n-3}^{(n)} d_{n-2} \tag{37}$$

The backward rewriting expresses the term $t_{n-2}^{(n)}$ as

$$t_{n-2}^{(n)} = [1 - d_{n-2} - r_{n-2}^{(n-1)} - c_{n-3}^{(n)} + 2r_{n-2}^{(n-1)} c_{n-3}^{(n)} \\ - 2d_{n-2}(2r_{n-2}^{(n-1)} c_{n-3}^{(n)} - r_{n-2}^{(n-1)} - c_{n-3}^{(n)})] \tag{38}$$

The above equation is further simplified by using the assumption $r_{n-2}^{(n-1)} = 0$, which gives rise to a modified polynomial for $t_{n-2}^{(n)}$:

$$t_{n-2}^{(n)} = [1 - d_{n-2} - c_{n-3}^{(n)} + 2d_{n-2} c_{n-3}^{(n)}] \tag{39}$$

Equations (37) and (39) translate the expression $2^{(n-2)} t_{n-2}^{(n)} + 2^{(n-1)} c_{n-2}^{(n)} - 2^{(n-1)}$ to $2^{(n-2)} c_{n-3}^{(n)} - 2^{(n-2)} d_{n-2} - 2^{(n-2)}$. Consequently, Equation (35) simplifies to:

$$\sum_{i=1}^{n-1} q_i 2^i D + [q_0 \cdot D + q_0 \cdot \sum_{i=0}^{n-3} t_i^{(n)} 2^i + 2^{n-2} c_{n-3}^{(n)} - 2^{n-2} d_{n-2} - 2^{n-2} - q_0 \cdot R^{(n-1)}] \\ + R^{(n-1)} - R^{(0)} \tag{40}$$

In the above equation, the backward rewriting replaces $2^{n-3}[t_{n-3}^{(n)} + 2c_{n-3}^{(n)}\ ]$ by the full adder equation : $2^{n-3}[1 - d_{n-3} + r_{n-3}^{(n-1)} + c_{n-4}^{(n)}]$. Therefore, transforming the above equation to:

$$\sum_{i=1}^{n-1} q_i 2^i D + [q_0 \cdot D + q_0 \cdot \sum_{i=0}^{n-4} t_i^{(n)} 2^i + 2^{n-3} c_{n-4}^{(n)} - 2^{n-3} r_{n-3}^{(n-1)} - \sum_{i=n-3}^{n-2} d_i 2^i - 2^{(n-3)} \\ - q_0 \cdot R^{(n-1)}] + R^{(n-1)} - R^{(0)} \tag{41}$$

The backward rewriting replaces the remaining full adders up to FA_0 with the corresponding full adder equation (Equation (4)) to finally arrive at the following equation.

$$\sum_{i=1}^{n-1} q_i 2^i D + q_0[D + t_0^{(n)} + 2c_0^{(n)} + \sum_{i=1}^{n-3} r_i^{(n-1)} 2^i - \sum_{i=1}^{n-2} d_i 2^i - 2 - R^{(n-1)}] + R^{(n-1)} - R^{(0)} \quad (42)$$

The above equation simplifies to the following equation due to: $D = \sum_{i=0}^{n-2} d_i 2^i$ and $R^{(n-1)} = \sum_{i=0}^{n-3} r_i^{(n-1)} 2^{(i)}$.

$$\sum_{i=1}^{n-1} q_i 2^{(i)} D + q_0[t_0^{(n)} + 2c_0^{(n)} + d_0 - r_0^{(n-1)} - 2] + R^{(n-1)} - R^{(0)} \quad (43)$$

The expression: $[t_0^{(n)} + 2c_0^{(n)} + d_0 - r_0^{(n-1)} - 2]$ in the above equation will reduce to zero if the backward rewriting replaces $t_0^{(n)} + 2c_0^{(n)}$ by the full adder equation: $1 - d_0 + r_0^{(n-1)} + 1$. However, with $r_{n-1}^{(n-1)} = 0$ and $\overline{d}_{n-1} = 1$, $q_0 = c_{n-2}^{(n)}$. As already described in Equation(37), $c_{n-2}^{(n)} = c_{n-3}^{(n)} - c_{n-3}^{(n)} d_{n-2}$, in other words, $q_0 = c_{n-3}^{(n)} - c_{n-3}^{(n)} d_{n-2}$. The term P'_{n-2} denotes the polynomial for the carry bit $c_{n-3}^{(n)}$. The current proof uses the same analysis presented in Section 5.3. The polynomial for the carry bit $c_{n-3}^{(n)}$ can be constructed as follows: $P'_1 = r_0^{(n-1)} - r_0^{(n-1)} d_0 + [1 - r_0^{(n-1)} - d_0 + 2r_0^{(n-1)} d_0] c_{-1}^{(n)}$ and $P'_j = r_{j-1}^{(n-1)} - r_{j-1}^{(n-1)} d_{j-1} + [1 - r_{j-1}^{(n-1)} - d_{j-1} + 2r_{j-1}^{(n-1)} d_{j-1}] P'_{j-1}$, for $j = 2, \ldots, n-2$. The size of the polynomial P'_1 denoted as $|P'_1|$ is equivalent to: $|P'_1| = 2^1 + 2^2$. The size of the polynomial P'_j denoted as $|P_j|$ is given as follows.

$$|P_j| = 2 + 2^2 (\sum_{i=0}^{j-1} 2^{i+1}) = 8 \cdot 2^j - 6$$

Therefore, the size of the polynomial $|P'_{n-2}|$ representing the carry bit $c_{n-3}^{(n)}$ is equal to $2^{n+1} - 6$. The size of the polynomial representing q_0 is $|q_0| = 2^{n+2} - 12$. The following provides the size of the final polynomial denoted as $|P'|$ seen in Equation (43).

$$|P'| = (n-1)(n-1) + 5(2^{n+2} - 6) + n - 2 + 2n - 2$$
$$|P'| = 5(2^{n+2}) + n^2 + n - 57$$

It is correct to conclude that the final polynomial in Equation (43) has an exponential size. This proves that the co-factorized polynomial has an exponential size. Thus, the original non co-factorized polynomial also will have an exponential size.

5.7 Delayed Backward Rewriting

The previous section clearly demonstrates the ineffectiveness of don't care cubes and their extension: the splitting of don't care cubes due to the presence of specific terms, in the current case, term q_0. With the existing optimization, it is inevitable to avoid an exponential blow-up of intermediate polynomial size. Thus, the current section illustrates a heuristic that delays the replacement of specific terms or monomials, thereby circumventing the limitations of existing optimization. The fundamental principle is as follows: whenever a signal's occurrence in a polynomial exceeds a specified threshold, the backward rewriting delays the rewriting of such signal, which otherwise should have been replaced according to the topological order. Instead, the backward rewriting continues by replacing the next available monomial according to topological order. Rewriting of the delayed signal takes place only when its occurrence does not decrease any more.

By doing so, the backward rewriting avoids the replacement ofa signal multiple times, thereby reducing the probability of an exponential blow-up of the intermediate polynomial size. The following analysis justifies that delaying the rewriting of q_0 results in the intended polynomial without a significant increase in the intermediate polynomial size. The analysis starts with the polynomial obtained after the backward rewriting replaces $Stage_n^{MUX}$ (refer to Equation (21)).

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0 (\sum_{i=0}^{n-2} t_i^{(n)} 2^i - 2^{n-1} t_{n-1}^{(n)} - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^{n-1} r_{n-1}^{(n-1)} + D) - R^{(0)} \quad (44)$$

At this stage, according to topological order, the backward rewriting should replace q_0. However, q_0 appears nearly $3n$ times in the polynomial present in Equation (44), and replacing it could increase the current size of the polynomial. Therefore, the backward rewriting skips or delays the immediate replacement of q_0 and continues with the signal $t_{n-1}^{(n)}$ available in the topological queue. Section 5.3, Equation (23) illustrates the polyno-

mial obtained after replacing the signal $t_{n-1}^{(n)}$.

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0 (\sum_{i=0}^{n-2} t_i^{(n)} 2^i + 2^{n-1} c_{n-2}^{(n)} \\ - 2^n r_{n-1}^{(n-1)} c_{n-2}^{(n)} - 2^{n-1} - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^n r_{n-1}^{(n-1)} + D) - R^{(0)} \tag{45}$$

The backward rewriting utilizes the don't care cube $(r_{n-1}^{(n-1)} c_{n-2}^{(n)}) = (1, 1)$ available at this stage to minimize the size of the intermediate polynomial. In Equation (45), the monomial q_0 associates with the monomial $r_{n-1}^{(n-1)} c_{n-2}^{(n)}$, which represents the mentioned don't care cube. To optimize the size of the intermediate polynomial, the monomial $r_{n-1}^{(n-1)} c_{n-2}^{(n)}$ is split into smaller cubes $v_1(q_0 r_{n-1}^{(n-1)} c_{n-2}^{(n)}) + v_2(1 - q_0)(r_{n-1}^{(n-1)} c_{n-2}^{(n)})$, which further simplifies as $(v_1 - v_2)(q_0 r_{n-1}^{(n-1)} c_{n-2}^{(n)}) + v_2(r_{n-1}^{(n-1)} c_{n-2}^{(n)})$. The modified don't care monomial is added to Equation (45).

This addition enables the optimization of the polynomials, as discussed in Section 5.4.2. The resulting polynomial is:

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0 (\sum_{i=0}^{n-2} t_i^{(n)} 2^i + 2^{n-1} c_{n-2}^{(n)} \\ + (v_1 - v_2 - 2^n)(r_{n-1}^{(n-1)} c_{n-2}^{(n)}) + v_2(r_{n-1}^{(n-1)} c_{n-2}^{(n)}) - 2^{n-1} \\ - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^n r_{n-1}^{(n-1)} + D) - R^{(0)} \tag{46}$$

Assigning $v_2 = 0$ and $v_1 = 2^n$ optimizes the above equation to:

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0 (\sum_{i=0}^{n-2} t_i^{(n)} 2^i + 2^{n-1} c_{n-2}^{(n)} - 2^{n-1} \\ - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^n r_{n-1}^{(n-1)} + D) - R^{(0)} \tag{47}$$

In the above equation, $2^{n-2}t_{n-2}^{(n)} + 2^{n-1}c_{n-2}^{(n)}$ resembles a one-bit full adder equation (Equation (4)), hence the backward rewriting replaces $2^{n-2}t_{n-2}^{(n)} + 2^{n-1}c_{n-2}^{(n)} = 2^{n-2}(1 - d_{n-2} + r_{n-2}^{(n-1)} + c_{n-3}^{(n)})$, which results in

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0(\sum_{i=0}^{n-3} t_i^{(n)} 2^i + 2^{n-2} c_{n-3}^{(n)} + 2^{n-2} r_{n-2}^{(n-1)} - 2^{n-2} d_{n-2} - 2^{n-2} - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^n r_{n-1}^{(n-1)} + D) - R^{(0)} \tag{48}$$

Similarly, backward rewriting replaces $2^{n-3}t_{n-3}^{(n)} + 2^{n-2}c_{n-3}^{(n)} = 2^{n-3}(1 - d_{n-3} + r_{n-3}^{(n-1)} + c_{n-4}^{(n)})$ and remaining full adders with their respective equation (4) to result in:

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0(\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - \sum_{i=0}^{n-2} d_i 2^i - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^n r_{n-1}^{(n-1)} + D) - R^{(0)} \tag{49}$$

In the above equation, $D - \sum_{i=0}^{n-2} d_i 2^i = 0$ and $\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - \sum_{i=0}^{n-2} r_i^{(n-1)} 2^i = 0$. Thus, the equation simplifies to

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + q_0(2^n r_{n-1}^{(n-1)}) - R^{(0)} \tag{50}$$

The occurrence of q_0 in the specification polynomial monotonically decreases, and the size of the intermediate polynomial remains quadratic. After replacing $Stage_n^{SUB}$, the occurrence of q_0 no longer decreases, consequently, promoting the replacement of q_0 as $1 - t_{n-1}^{(n)}$. Where $t_{n-1}^{(n)} = 1 - r_{n-1}^{(n-1)} - c_{n-2}^{(n)} + 2r_{n-1}^{(n-1)}c_{n-2}^{(n)}$ and as a result of which, $q_0 = r_{n-1}^{(n-1)} + c_{n-2}^{(n)} - 2r_{n-1}^{(n-1)}c_{n-2}^{(n)}$. This polynomial of q_0 translates the above equation to:

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i - 2^{n-1} r_{n-1}^{(n-1)}) + 2^n r_{n-1}^{(n-1)} - 2^n r_{n-1}^{(n-1)} c_{n-2}^{(n)} - R^{(0)} \tag{51}$$

In the current stage, the process of optimising the above polynomial by adding a don't care monomial $v_1(r_{n-1}^{(n-1)}c_{n-2}^{(n)})$ is equivalent to removing the monomial $r_{n-1}^{(n-1)}c_{n-2}^{(n)}$ present in

the specification polynomial. As a result of which, the specification polynomial translates to:

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-2} r_i^{(n-1)} 2^i + 2^{n-1} r_{n-1}^{(n-1)}) - R^{(0)} \tag{52}$$

From the above analysis, delaying the replacement of a signal that occurs often and replacing it only when its occurrence no longer reduces helps to maintain the quadratic polynomial size. In the best case, this approach can also lead to the intended polynomial at a specific cut. The delayed backward rewriting becomes another optimization technique to circumvent an exponential blow-up of intermediate polynomial size.

5.8 Analysis of $Stage_{n-1}$ of a Reduced Restoring Divider

The previous section illustrated a heuristic that yields the intended polynomial at the cut between $Stage_{n-1}^{MUX}$ and $Stage_n^{SUB}$ without causing a significant increase in the polynomial size. However, this is not always the case for stages ($Stage_j$) that occur before $Stage_n$ in a reduced restoring divider circuit. The current section presents the analysis of $Stage_{n-1}$ to highlight the limitations of delayed rewriting and don't care optimization. The analysis of $Stage_{n-1}$ continues with the polynomial (Equation (52)) obtained after rewriting $Stage_n$. The Backward rewriting obtains the polynomial between $stage_{(n-1)}^{MUX}$ and $stage_{(n-1)}^{SUB}$ by replacing every bit of $R^{(n-1)}(\sum_{i=0}^{n-1} r_i^{(n-1)} 2^{(i)})$, where $1 \leq i \leq n-1$, by the mux equation (11) : $q_1 t_{i-1}^{(n-1)} + (1-q_1) r_i^{(n-2)}$ and replacing $r_0^{(n-1)}$ with $r_0^{(n-2)}$. The polynomial simplifies further by rearranging the term associated with q_1 together, resulting in the following expression.

$$\sum_{i=2}^{n-1} q_i 2^i \cdot D + q_1(2D + \sum_{i=0}^{n-2} t_i^{(n-1)} 2^{i+1} - \sum_{i=1}^{n-1} r_i^{(n-2)} 2^i) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^i - R^{(0)} \tag{53}$$

Similar to the previous section, the backward rewriting delays the replacement of the signal q_1 and continues by replacing $t_{n-2}^{(n-1)}$. Where, $t_{n-2}^{(n-1)} = \overline{d}_{n-2} \oplus r_{n-1}^{(n-2)} \oplus c_{n-3}^{(n-1)}$ and is equivalent to

$$\begin{aligned} t_{n-2}^{(n-1)} = \overline{d}_{n-2} + r_{n-1}^{(n-2)} + c_{n-3}^{(n-1)} - 2r_{n-1}^{(n-2)}c_{n-3}^{(n-1)} - 2\overline{d}_{n-2}r_{n-1}^{(n-2)} \\ - 2\overline{d}_{n-2}c_{n-3}^{(n-1)} + 4\overline{d}_{n-2}r_{n-1}^{(n-2)}c_{n-3}^{(n-1)} \end{aligned} \tag{54}$$

Substituting the polynomial for $t_{n-2}^{(n-1)}$ in Equation (53) results in the following polynomial:

$$\begin{aligned} &\sum_{i=2}^{n-1} q_i 2^i \cdot D + q_1(2D + \sum_{i=0}^{n-3} t_i^{(n-1)} 2^{i+1} + 2^{n-1}(\overline{d}_{n-2} + r_{n-1}^{(n-2)} + c_{n-3}^{(n-1)} - 2r_{n-1}^{n-2}c_{n-3}^{(n-1)} \\ &- 2\overline{d}_{n-2}r_{n-1}^{(n-2)} - 2\overline{d}_{n-2}c_{n-3}^{(n-1)} + 4\overline{d}_{n-2}r_{n-1}^{(n-2)}c_{n-3}^{(n-1)}) - \sum_{i=1}^{n-1} r_i^{(n-2)} 2^i) \\ &+ \sum_{i=0}^{n-1} r_i^{(n-2)} 2^i - R^{(0)} \end{aligned} \tag{55}$$

The backward rewriting avails the don't care assignment $(\overline{d}_{n-2}, r_{n-1}^{(n-2)}, c_{n-3}^{(n-1)}) = (1,1,1)$ and deploys splitting of the don't care cube $(\overline{d}_{n-2}r_{n-1}^{(n-2)}c_{n-3}^{(n-1)})$ in a similar way shown in the previous section to optimize the above equation. This optimization removes the term $\overline{d}_{n-2}r_{n-1}^{(n-2)}c_{n-3}^{(n-1)}$ in the above equation to yield the following intermediate polynomial.

$$\begin{aligned} &\sum_{i=2}^{n-1} q_i 2^i \cdot D + q_1(2D + \sum_{i=0}^{n-3} t_i^{(n-1)} 2^{i+1} + 2^{n-1}(\overline{d}_{n-2} + r_{n-1}^{(n-2)} + c_{n-3}^{(n-1)} - 2r_{n-1}^{(n-2)}c_{n-3}^{(n-1)} \\ &- 2\overline{d}_{n-2}r_{n-1}^{(n-2)} - 2\overline{d}_{n-2}c_{n-3}^{(n-1)}) - \sum_{i=1}^{n-1} r_i^{(n-2)} 2^i) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^i - R^{(0)} \end{aligned} \tag{56}$$

The backward rewriting replaces the expression $2^{n-2}t_{n-3}^{(n-1)} + 2^{n-1}c_{n-3}^{(n-1)}$ with the full adder equation: $2^{n-2}[\overline{d}_{n-3} + r_{n-2}^{(n-2)} + c_{n-4}^{(n-1)}]$. Additionally, backward rewriting also replaces the term $c_{n-3}^{(n-1)}$ associated with other signals: $\overline{d}_{n-2}$ and $r_{n-1}^{(n-2)}$. The polynomial expression for the term $c_{n-3}^{(n-1)}$ is equivalent to $r_{n-2}^{(n-2)}\overline{d}_{n-3} + r_{n-2}^{(n-2)}c_{n-4}^{(n-1)} + \overline{d}_{n-3}c_{n-4}^{(n-1)} - 2\overline{d}_{n-3}r_{n-2}^{(n-2)}c_{n-4}^{(n-1)}$. For better readability, the term k denotes the overall expression obtained after replacing

$c_{n-3}^{(n-1)}$. These replacements transform the above equation to the following polynomial.

$$\sum_{i=2}^{n-1} q_i 2^i \cdot D + q_1(2D + \sum_{i=0}^{n-4} t_i^{(n-1)} 2^{i+1} + 2^{n-2} c_{n-4}^{(n-1)} + \sum_{i=n-3}^{n-2} \overline{d}_i^{(n-1)} 2^{i+1} + 2^{n-1}(k) - \sum_{i=1}^{n-3} r_i^{(n-2)} 2^i) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^i - R^{(0)} \quad (57)$$

The backward rewriting next replaces $2^{n-3} t_{n-4}^{(n-1)} + 2^{n-2} c_{n-4}^{(n-1)}$ by the full adder equation: $2^{n-3}[\overline{d}_{n-4} + r_{n-3}^{(n-2)} + c_{n-5}^{(n-1)}]$ and the term $c_{n-4}^{(n-1)}$ (present in k) by the expression $r_{n-3}^{(n-2)} \overline{d}_{n-4} + r_{n-3}^{(n-2)} c_{n-5}^{(n-1)} + \overline{d}_{n-4} c_{n-5}^{(n-1)} - 2\overline{d}_{n-4} r_{n-3}^{(n-2)} c_{n-5}^{(n-1)}$. Similarly, backward rewriting replaces the remaining full adders with their respective full adder equations along with replacing the carry term c_i^{n-1} present in k, where $0 \leq i \leq n-5$, with their respective equations to arrive at the following polynomial.

$$\sum_{i=2}^{n-1} q_i 2^i \cdot D + q_1(2D + \sum_{i=0}^{n-2} \overline{d}_i 2^{i+1} + 2c_{-1}^{(n-1)} + 2^{n-1}(k)) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^i - R^{(0)} \quad (58)$$

Substituting $\overline{d}_i = 1 - d_i$, $2D - 2\sum_{i=0}^{n-2} d_i 2^i = 0$, and $\sum_{i=0}^{n-2} 2^{i+1} + 2c_{-1}^{(n-1)} = 2^n (\text{as } c_{-1}^{n-1} = 1)$ in the above expressions, gives rise to the following expression .

$$\sum_{i=2}^{n-1} q_i 2^{(i)} . D + q_1(2^n + 2^{n-1}(k)) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^{(i)} - R^{(0)} \quad (59)$$

Backward rewriting replaces q_1 as $1 - t_{n-1}^{(n-1)}$, which is equivalent to $q_1 = r_n^{(n-2)} + c_{n-2}^{(n-1)} - 2r_n^{(n-2)} c_{n-2}^{(n-1)}$. Substituting the polynomial for q_1 in the above equation results in

$$\sum_{i=2}^{n-1} q_i 2^{(i)} \cdot D + (r_n^{(n-2)} + c_{n-2}^{(n-1)} - 2r_n^{(n-2)} c_{n-2}^{(n-1)})(2^n + 2^{n-1}(k)) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^{(i)} - R^{(0)} \quad (60)$$

As seen before, the backward rewriting utilizes the don't care splitting optimization to remove every term associated with the don't care cube $r_n^{(n-2)} c_{n-2}^{(n-1)}$. It then continues by

replacing $c_{n-2}^{(n-1)}$ as $r_{n-1}^{(n-2)}\overline{d}_{n-2}+r_{n-1}^{(n-2)}c_{n-3}^{(n-1)}+\overline{d}_{n-2}c_{n-3}^{(n-1)}-2\overline{d}_{n-2}r_{n-1}^{(n-2)}c_{n-3}^{(n-1)}$. The expression for $c_{n-2}^{(n-1)}$ further simplifies to $r_{n-1}^{(n-2)}\overline{d}_{n-2}+r_{n-1}^{(n-2)}c_{n-3}^{(n-1)}+\overline{d}_{n-2}c_{n-3}^{(n-1)}$ due to the presence of don't care cube $\overline{d}_{n-2}r_{n-1}^{(n-2)}c_{n-3}^{(n-1)}$. However, at this stage, the term k is a function of $< r_{n-2}^{n-2}\dots r_1^{n-2} >$ and $< d_{n-3}\dots d_0 >$ and the size of k is significantly large. Additionally, the presence of the term $c_{n-3}^{(n-1)}$ in the expression for $c_{n-2}^{(n-1)}$ will trigger an exponential blow-up of the intermediate polynomial size upon its replacement (see *Lemma 1* and *Lemma 2* in [16]). Though delaying the replacement of high occurrence signal q_1 might give a slight advantage over replacing it according to the topological order, due to the lack of intended don't care assignments, delaying signals will not avoid exponential blow-up of the polynomial size. In $Stage_n$, the presence of don't care cube $(r_{n-1}^{(n-1)}, c_{n-2}^{(n)}) = (1,1)$ and constant propagation of $d_{n-1} = 0$ enable the backward rewriting to replace $Stage_n^{SUB}$ without causing a substantial increase in the polynomial size. However, in $Stage_{n-1}$, the backward rewriting begins with a polynomial without the sign bit $r_n^{(n-1)}$ belonging to the partial remainder $R^{(n-1)}$. The absence of this sign bit disables the opportunity of utilising the don't care cube $(r_n^{(n-2)}, c_{n-2}^{(n-1)}) = (1,1)$ and constant propagation $d_{n-1} = 0$, which otherwise would replace $Stage_{n-1}^{SUB}$ in a similar way as shown in Section 5.7, without causing an exponential increase in the polynomial size. Unfortunately, the optimization techniques discussed in the current work fail to address the challenge shown in the current section. Therefore, improving the current optimization techniques is essential to deduce a computationally inexpensive automatic verification tool.

6 Verification of $0 \leq R < D$

So far, current work has focused mainly on verifying the condition $R^{(0)} = Q \cdot D + R$. However, it is also critical to verify the condition $0 \leq R < D$, which ensures the division always yields a unique quotient and remainder. Although backward rewriting could be employed to verify this condition, this approach would fail because the backward rewriting starts with a polynomial representation for $0 \leq R < D$, which already has exponential size [3]. Nevertheless, there is a BDD that represents $0 \leq R < D$, thus allowing a feasible approach to verify $0 \leq R < D$. As mentioned in Section 5.4.1, the current work uses BDD-based image computation to verify *VC2*.

Since verifying *VC2* relies on BDD-based image computation, it is key to illustrate the representation of the input constraint $0 \leq R^{(0)} < D \cdot 2^{n-1}$ using BDD. The construction

of BDDs that represent the image, input constraint (*IC)*, and the condition *VC2* involves using a static variable ordering, where bits r_i and d_i, as well as $r^{(0)}_{i+n-1}$ and d_i are arranged side by side. In this variable ordering, the evaluation begins with the bits of higher indices [3]. Since the dividend $R^{(0)}$ has a bit width of $2n-1$ and the n bit divisor D is left-shifted by the $n-1$ positions, the BDD representing the input constraint (IC) consist of nodes with variables ranging from $r^{(0)}_{2n-2}$ to $r^{(0)}_{n-1}$ and d_{n-1} to d_0. Thus, this BDD ignores the bits in $R^{(0)}$ with indices below $n-1$. Since $d_{n-1}=1$ violates the condition $D < 2^{n-1}$, BDD models this condition by directing the high edge of the node corresponding to bit d_{n-1} to terminal node zero and the low edge of this node to the node corresponding to bit $r^{(0)}_{2n-2}$. Similarly, $r^{(0)}_{2n-2}=1$ violates the condition $R^{(0)} \geq 0$, the BDD models this condition by directing the high edge of the node corresponding to bit $r^{(0)}_{2n-2}$ to terminal node zero and the low edge of this node to the node corresponding to bit d_{n-2}. The BDD verifies condition $R^{(0)} < D \cdot 2^{n-1}$ by comparing bits of $R^{(0)}$ and D, starting with higher indices, specifically $r^{(0)}_{2n-3}$ and d_{n-2}. The following three cases describe this comparison.

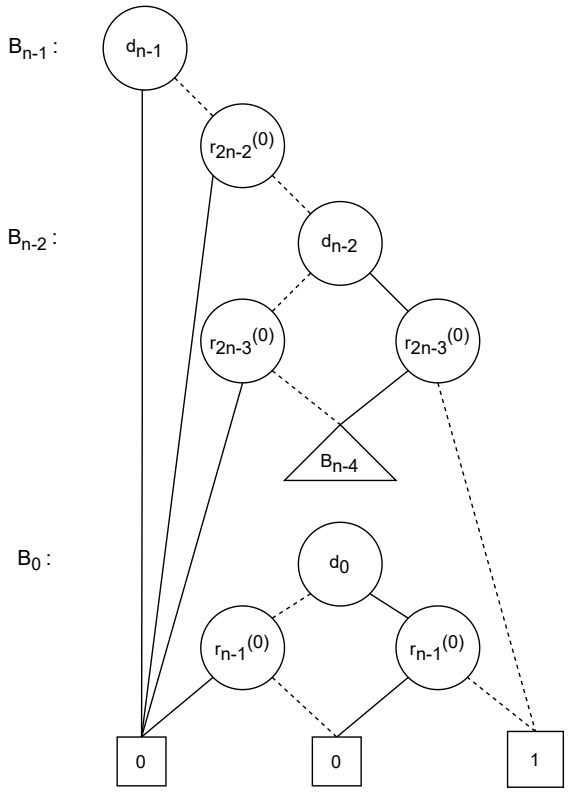

Figure 9: BDD representing the input constraint: $0 \leq R^{(0)} < D \cdot 2^{n-1}$

Case 1: If the divisor bit is equal to one and the dividend bit is equal to zero, the condition $R^{(0)} < D \cdot 2^{n-1}$ is satisfied. Therefore, there is no need to compare the subsequent bits. The BDD achieves this comparison by connecting a high edge between the node corresponding to bit d_{n-2} and the node corresponding to bit $r^{(0)}_{2n-3}$. It then connects a low edge from node corresponding to bit $r^{(0)}_{2n-3}$ to terminal node 1.

Case 2: If the divisor bit is equal to zero and the dividend bit is equal to one, the condition $R^{(0)} < D \cdot 2^{n-1}$ is not fulfilled. In this case, there is no need to compare the subsequent bits. The BDD models this case by directing the low edge of the node corresponding to bit d_{n-2} to the node corresponding to bit $r^{(0)}_{2n-3}$. It then connects a high edge from the node corresponding to bit $r^{(0)}_{2n-3}$ to terminal node 0.

Case 3: If the divisor bit and the dividend bit are identical, the comparison proceeds to the next subsequent bits. The BDD expresses this case by directing both edges of the node corresponding to bit $r^{(0)}_{2n-3}$ to the node corresponding to the subsequent bit based on the static variable ordering.

The construction of BDD follows these three cases until it reaches the comparison between the least significant bits of D and $R^{(0)}$, specifically, the bits d_0 and $r^{(0)}_{n-1}$. At this stage, the comparison slightly differs from that discussed previously. The reason behind this difference is that the dividend $R^{(0)}$ must be strictly less than the divisor D ($\because R^{(0)} < D \cdot 2^{n-1}$). The condition $R^{(0)} < D \cdot 2^{n-1}$ is only satisfied if $d_0 = 1$ and $r^{(0)}_{n-1} = 0$ and BDD models this by directing the high edge from the node corresponding to bit d_0 to the node corresponding to bit $r^{(0)}_{n-1}$ and then connecting the low edge from this node to terminal node 1. Figure 9 illustrates the cases discussed above with a high-level BDD representation of the input constraint (IC), using dotted lines for low edges and solid lines for high edges. Figure 10 shows the BDD that represents the condition $0 < R < D$. This BDD also follows the same principle as discussed above to compare the bits of the final remainder R and the divisor D and represent the condition *VC2*. The only difference here is that R and D have the same bit width, $n - 1$.

As already mentioned in Section 5.4.1, the application of BDD-based image computation is not only restricted to obtaining the don't care cubes at the inputs of atomic blocks but also plays a pivotal role in verifying condition *VC2*. The BDD-based image computation produces various images of IC under different slices. Performing the image computation on the final set of atomic blocks produces the final and complete image IMG_f of the IC

under the whole circuit considered for verification [3]. This image IMG_f represents the possible output space of the final remainder R and is, therefore, vital for the verification of condition *VC2*. The final step in this verification involves checking whether IMG_f implies $0 \leq R < D$ [3]. In other words, $((\neg IMG_f) \vee (0 \leq R < D))$ should result in a constant one in order to confirm condition $0 \leq R < D$ holds.

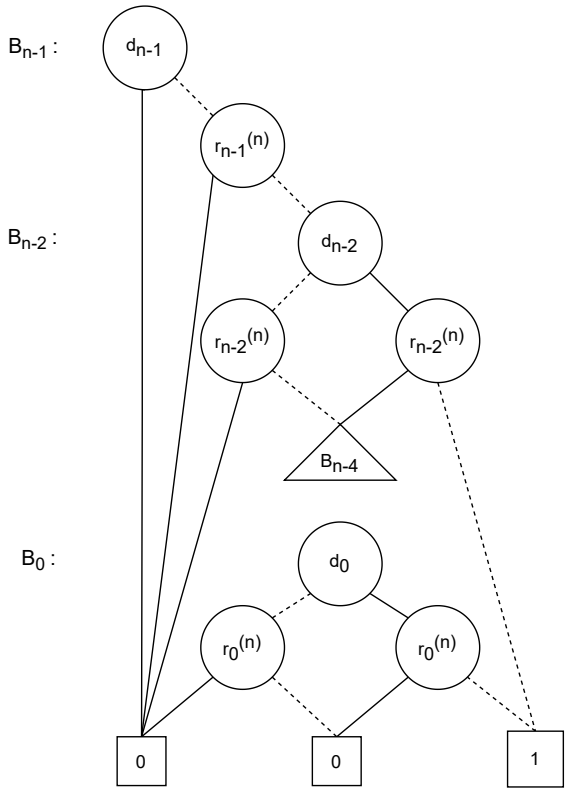

Figure 10: BDD representing the condition $R < D$

7 Evaluation of the Final Polynomial

As already mentioned in Section (5.3), it is critical to evaluate the final polynomial obtained after the backward rewriting process terminates. Observations made in [3] show that applying a non-optimized SCA-based method to an optimized divider [3, fig 3] with a small width results in a non-zero polynomial. Though the typical SCA-based verification would have concluded that the implementation does not satisfy the specification, this

is not necessarily true for optimized dividers. A non-zero final polynomial could still indicate a correct implementation of a divider if it evaluates to zero for all inputs satisfying the input constraint $IC := 0 \leq R^{(0)} < D \cdot 2^{n-1}$. The same argument also holds for the case of reduced restoring divider. [3, Alg. 3] delineates an algorithm that evaluates the final polynomial by bitwise comparing $R^{(0)}$ and D, starting with the most significant bit. The comparison between $R^{(0)}$ and D follows the same principle as described in Section (6). The algorithm successively evaluates the final polynomial for a given size (n) of the divider as the following. For each iteration i, where ($0 \leq i \leq n-2$), the algorithm first simulates case 1 (Section 6) by assigning d_i and $r^{(0)}_{i+n-1}$ with constant values 1 and 0, respectively. If the polynomial does not reduce to zero, the final polynomial violates the input constraint; therefore, the algorithm terminates by returning constant zero, indicating faulty implementation of the divider. If the polynomial reduces to zero, the algorithm simulates case 3 (Section 6) by assigning $d_i = r^{(0)}_{i+n-1}$ and continues with the succeeding bit. The algorithm terminates by returning constant one when it has evaluated case 1 and case 3 mentioned in Section 6 for all the bits of $R^{(0)}$ and D, thereby concluding that the implementation of the divider satisfies the specification.

8 Implementation

This section presents two algorithms: splitting of the don't care cube and delayed optimized backward rewriting. These two optimization techniques are key components of the automatic verification tool.

8.1 Implementation of Splitting of Don't Care Cube

Section 5.4 already provides a theoretical overview of the splitting of don't care cubes and its role in minimizing the size of polynomials. The algorithm 3 delineates the implementation of this optimization technique in an automatic verification tool. The algorithm 3 aims to identify the don't care split cubes that could be present in the specification polynomial at some stage of the backward rewriting. The fundamental principle of the algorithm 3 is to determine the monomial m_s that is in conjunction with the don't care polynomial. Additionally, the algorithm 3 also determines the presence of the *pure don't care* polynomial. To understand this principle, the current section reintroduces the example presented in Section 5.4. Theoretically, polynomial

$K(a,b,c,d) = 3ad+2b-8cb+5ac-5abc+a-ab$ is optimized by adding two smaller cubes : $v_1(a-ab)c + v_2(a-ab)(1-c)$ of the original don't care cube $dc_1 = a - ab$. However, the algorithm 3 iterates through the polynomial K to first find the presence of a pure don't care polynomial $a - ab$ and then identifies the monomial c that is in conjunction with the don't care cube $a - ab$. Consequently, the algorithm adds polynomials $v_1(a - ab)$ and $v_2(ac - abc)$ to the polynomial $K(a, b, c, d)$.

The verification tool optimizes the resulting polynomial $K(a,b,c,d) = 3ad + 2b - 8cb + ac(5 + v_2) - abc(5 + v_2) + a(v_1 + 1) - ab(v_1 + 1)$ by assigning $v_1 = -1$ and $v_2 = -5$. This optimization results in $K(a,b,c,d) = 3ad+2b-8cb$, which is equivalent to the polynomial that is obtained after optimization using splitting of the don't care cube shown in Section 5.4.

The verification tool translates the don't care assignment at the inputs of an atomic into a don't care polynomial (*dc_poly*), which serves as one of the inputs to the algorithm 3. Along with *dc_poly*, the algorithm 3 takes the individual signals present in the *dc_poly* contained in a set *dc_signals*, and intermediate specification polynomial SP_i as its inputs. The algorithm either identifies the don't care cubes contained in the specification polynomial (SP_i) and incorporates them into the polynomial or returns the message "no dc-split cubes" if the intended don't care cube is not present in the specification polynomial (SP_i). For the discussed example, the inputs to the algorithm would be $dc\_poly = a - ab$, $dc\_signals = \{a, b\}$ and $SP_i = K(a, b, c, d)$. The initial part of the algorithm determines whether a pure don't care cube is present in SP_i by calling the function `findExact` (line 8) for every monomial $m_i \in dc\_poly$ (line 7). The function `findExact` returns m_i if m_i is present in SP_i; otherwise, it returns *null*. If `findExact` returns null (line 8), the algorithm exits the loop early by assigning the Boolean *all_monomials_found* to false (line 9), which otherwise is initialized to true. This termination indicates that there is no pure don't care cube present in SP_i. The flag *all_monomials_found* being equal to true after the completion of the loop (line 14) confirms the presence of every monomial $m_i \in dc\_poly$ in SP_i. As a result of this, the algorithm invokes the function `addSingleDCPolynomial` to add the pure don't care polynomial *dc_poly* to SP_i (line 15).

The next stage of the algorithm determines the presence of split cubes of don't care polynomial, if any. The algorithm first determines the type of *dc_poly* by invoking the function `check_dcType` (line 3), which returns either type1 or type2 depending on the first

term of *dc_poly*. The algorithm requires *dc_poly* to be arranged in a graded lexicographical order. The function `check_dcType` categorises *dc_poly* as type2 if the first term of *dc_poly* is a constant one; otherwise it categorises as type1. For example, `check_dcType` $(a - ab)$ will return type1, whereas `check_dcType` $(1 - a - b + ab)$ would return type2. The later part of this section discusses the reason for such a categorization. Suppose the type of the *dc_poly* is type1 (line17).

In that case, the algorithm identifies all monomials in SP_i that are in conjunction with the first monomial m_1 of *dc_poly* (line 18). The algorithm realizes this query by invoking the function `findContaining`, which retrieves and stores the monomials in a collection $Conjunction\_monomials\_type1$ (line 19). This approach guarantees that the algorithm not only obtains the terms directly associated with m_1 but also additional terms that are associated with other monomials of *dc_poly*. The function call `findContaining` with m_1 as the argument retrieves additional terms that are associated with other monomials of *dc_poly* because the first term m_1 is associated with every other term m_i $(i \neq 1)$ of *dc_poly*. For example, the function call `findContaining`$(K(a, b, c, d), a)$ would return $\{3ad, 5ac, -5abc, a, -ab\}$. The algorithm terminates with the message "no dc-split cubes" when `findContaining` returns null, thereby indicating that the current polynomial SP_i does not contain any split cubes of the queried don't care cube. If $Conjunction\_monomials\_type1$ is not empty, the algorithm invokes the function `removeVars_extractMonomial` (line 23) with $Conjunction\_monomials\_type1$, *dc_signals* and size(*dc_poly*) as the arguments. This function performs two crucial tasks: first, it removes all the signals $s_i \in$ *dc_signals* associated with each element of $Conjunction\_monomials\_type1$; second, it removes the elements in the modified $Conjunction\_monomials\_type1$ whose occurrence count does not match with the number of monomials in *dc_poly*. `removeVars_extractMonomial` finally consolidates all the terms whose count matches the size of *dc_poly*. As a result of which, the function identifies terms that fully cover the (*dc_poly*), indicating the presence of a complete split don't care cube. For example, calling the function `removeVars_extractMonomial` with arguments $\{3ad, 5ac, -5abc, a, -ab\}$, (a, b) and `size`$(dc\_poly) = 2$ will first remove (a, b) from $\{3ad, 5ac, -5abc, a, -ab\}$ to result in $\{3d, 5c, -5c\}$. In the next step, the function removes the term d from $\{3d, 5c, -5c\}$, as the count of d is less than the `size`$(a - ab)$. The $Conjunction\_monomials\_type1$ will contain a monomial c because the count of monomial c matches the size of *dc_poly*. In the case that $Conjunction\_monomials\_type1$ becomes empty, it is a clear indication that the polynomial does not contain any split cubes of the queried don't care cube.

Hence, the algorithm terminates with the message "no dc-split cubes". Conversely, if $Conjunction\_monomials\_type1$ contains one or more terms, the algorithm multiplies each term t_i with the *dc_poly* to give rise to a don't care split polynomial (*dc_split_poly*) (line 26). The algorithm adds the (*dc_split_poly*) by calling the function `addSingleDCPolynomial`.

It is crucial to highlight here why the algorithm begins the search for pure don't care cubes prior to the split don't care cubes. To understand this approach, consider a polynomial $K^{'}(a, b, c, d) = 4a - ac + 2a - 2ab$ with the don't care polynomial $a - ab$. Assume that the algorithm omits line 7 to line 16 and terminates at line 33. The outcome of this truncated algorithm would be the message "no dc-split cubes" because $Conjunction\_monomials\_type1$ would become empty after being parsed by the function `removeVars_extractMonomial`. Hence, to avoid such an incorrect outcome, the algorithm 3 always identifies pure don't care cube before identifying split don't care cubes.

The same principle as discussed above also applies to type2 don't care polynomial. However, there are two minor modifications in the algorithm to search the don't care split cubes of type2 don't care polynomial effectively. As mentioned earlier in this section, the first term of the type2 don't care polynomial is always a constant 1, due to which, calling the function `findContaining` with constant one as one of the function's argument would return all the monomials present in SP_i. This approach would make the search very challenging and computationally expensive. To make the search feasible, the algorithm invokes the function `findContaining` for each signal s_i present in *dc_signals* (line 35-36). By doing so, the algorithm obtains all the monomials that are associated with every term of *dc_poly*. The algorithm uniquely collects all the terms returned by the function `findContaining` in a set $Conjunction\_monomials\_type2$. Apart from these modifications, the rest of the algorithm follows the same principle as discussed previously. The algorithm presented in this section guarantees that it identifies and incorporates every possible don't care cube available in the current specification polynomial.

Algorithm 3 Splitting of don't care cube

1: **Input:** Don't care polynomial *dc_poly*, don't care signals *dc_signals*, Intermediate specification polynomial SP_i
2: **Output:** Identified pure DC-polynomials and DC-split cubes, or "no dc-split cubes" if none were found.
3: *dc_type* := `check_dcType`(*dc_poly*);
4: $Conjunction\_monomials\_type1 := \emptyset$;
5: $Conjunction\_monomials\_type2 := \emptyset$;
6: $all\_monomials\_found$:= **True**;
7: **for** each monomial $m_i \in$ *dc_poly* **do**
8: *Exact_monomial* := `findExact`(SP_i, m_i);
9: **if** *Exact_monomial* = null **then**
10: $all\_monomials\_found$:= **False**;
11: **exit** loop;
12: **end if**
13: **end for**
14: **if** $all\_monomials\_found$ **then**
15: `addSingleDCPolynomial`(*dc_poly*);
16: **end if**
17: **if** *dc_type* = `type1` **then**
18: m_1 := first monomial in *dc_poly*;
19: $Conjunction\_monomials\_type1$:= `findContaining`(SP_i, m_1);
20: **if** $Conjunction\_monomials\_type1 = \emptyset$ **then**
21: **return** no dc-split cubes;
22: **else**
23: `removeVars_extractMonomial`($Conjunction\_monomials\_type1$, `size`(*dc_poly*), *dc_signals*);
24: **if** $Conjunction\_monomials\_type1 \neq \emptyset$ **then**
25: **for** each $t_i \in$ `Conjunction_monomials_type1` **do**
26: *dc_split_poly* := `multiplyDCPoly`(t_i, *dc_poly*);
27: `addSingleDCPolynomial`(*dc_split_poly*);
28: **end for**
29: **else**
30: **return** no dc-split cubes;
31: **end if**
32: **end if**
33: **end if**
34: **if** *dc_type* = `type2` **then**
35: **for** each $s_i \in$ *dc_signals* **do**
36: $Conjunction\_monomials\_type2$:= `findContaining`(SP_i, s_i);
37: **if** $Conjunction\_monomials\_type2 = \emptyset$ **then**
38: **return** no dc-split cubes;
39: **end if**
40: **end for**
41: `removeVars_extractMonomial`(Conjunction_monomials_type2, `size`(*dc_poly*), *dc_signals*);
42: **if** $Conjunction\_monomials\_type2 \neq \emptyset$ **then**
43: **for** each $t_i \in$ `Conjunction_monomials_type2` **do**
44: *dc_split_poly* := `multiplyDCPoly`(t_i, *dc_poly*);
45: `addSingleDCPolynomial`(*dc_split_poly*);
46: **end for**
47: **else**
48: **return** no dc-split cubes;
49: **end if**
50: **end if**

8.2 Implementation of Optimized Delayed Backward Rewriting

This section presents an algorithm that forms the core component of the verification tool designed to verify the divider circuits. The algorithm leverages two optimization techniques: *Don't Care optimization* (Section 5.4) and *Delayed Backward Rewriting* (Section 5.7) to realize an efficient verification tool. The utilization of these optimization techniques differs slightly from the theoretical approach presented in their respective sections. Unlike in Section 5.7 (Equation (45)), where the backward rewriting immediately utilizes the don't care cube to optimize the intermediate specification polynomial, the algorithm 4 only employs don't care optimization when the size of intermediate polynomial exceeds a specific threshold. The reason behind such an approach is that not every don't care cube at the inputs of an atomic block is necessary for optimizing the polynomial. Moreover, in some scenarios, the optimization of polynomials involves a combination of don't care cubes at the inputs of the same or different atomic blocks. The algorithm leverages such information by not making use of the don't care cubes immediately but by saving them whenever it encounters them at the inputs of the atomic blocks during backward rewriting. Whenever there is a significant increase in the polynomial size, the algorithm backtracks to the recently saved don't care cube and optimizes the polynomial. Similarly, the algorithm does not immediately delay the rewriting of a signal or a monomial with a high occurrence count; instead, it rewrites the signal with its corresponding gate polynomial. This is because, in some cases, the rewriting of a high-occurrence signal could trigger cancellations of monomials present in the specification polynomial, thereby reducing the size of the polynomial. However, in some cases, it is also plausible that replacing a high occurrence signal could cause a substantial increase in the size of the intermediate polynomial. In such situations, the algorithm backtracks to the point before the replacement of a high-occurrence signal and delays its replacement. The algorithm replaces the delayed signal when its occurrence count does not decrease any more. The following part of this section explains in detail how the algorithm handles the backtracking of two optimization techniques.

The algorithm begins with a gate-level netlist of the divider circuit. To perform efficient rewriting, the algorithm first identifies all the non-trivial and trivial atomic blocks. Then, it computes the topological order $\prec_{top}$ on the detected atomic blocks using the heuristic adopted in [3]. Before beginning the backward rewriting with the specification polynomial $SP^{init} = Q \cdot D + R - R^0$, the algorithm computes the don't care cubes at the

inputs of the atomic blocks (see Section 5.4.1) (line 7). The algorithm employs a single stack ST to store the backtrack points for both optimizations. By doing so, backward rewriting can utilize the combination of both optimization techniques when necessary. The algorithm also maintains a list *occurrence_list* that collects all atomic blocks with a high occurrence count. The *occurrence_list* is empty at the beginning, and the algorithm updates it at regular intervals by invoking the function `Update_occurrenceList` (line 14). The function `Update_occurrenceList` appends atomic blocks with occurrence count exceeding the specific threshold (*occurrence_threshold*) to the *occurrence_list*. The variables *atomic_counter* and *atomic_count_threshold* determine the intervals during which the algorithm calls the function `Update_occurrenceList`. The algorithm increments the *atomic_counter* after backward rewriting replaces an atomic block (line 12), and when the *atomic_counter* reaches the threshold (*atomic_count_threshold*), the algorithm invokes the function `Update_occurrenceList` and resets the counter to zero (line 14).

The algorithm checks whether an atomic block belongs to *occurrence_list* before the backward rewriting replaces the atomic block (line 9). Suppose the algorithm encounters an atomic block that belongs to the *occurrence_list*, in that case, the algorithm saves the backtrack point by pushing the current polynomial SP_i, the current iteration i, and the type of optimization (in this case skip_signal) to the stack ST (line 10). The algorithm simultaneously pushes the current atomic block a_i to the stack $Signal\_ST$ (line 10), which is required to delay the rewriting of the atomic block a_i when necessary. The algorithm continues by replacing the atomic block with its gate polynomial (line 12). Suppose the atomic block contains don't care cubes at its inputs, in that case, the algorithm again saves a backtrack point by pushing the replaced polynomial SP_{i-1}, the current iteration i, and the type of optimization (in this case dc) to the stack ST (line 27).

Whenever there is a significant increase in the polynomial size (line 16), the algorithm backtracks to the most recently saved backtrack point. It pops the stack ST to retrieve the saved specification polynomial, iteration and the type of optimization (line 17). If the optimization type is dc, the algorithm immediately optimizes the polynomial with the available don't care cubes (line 19). In case the optimization type is *skip_signal*, then the algorithm pops the most recently saved atomic block from $Signal\_ST$ and skips the rewriting of the popped atomic block in the current iteration, thereby delaying its replacement (line 21).

Once the algorithm delays the rewriting of an atomic block, it adds the atomic block to the list *Skipped_atomicBlock* and resets its count to zero (line 23). The list stores the current iteration '*i*', which is used during the rewriting of the delayed block, and tracks the occurrence count of each skipped atomic block. Additionally, the algorithm activates *track_skipped_signal*, enabling the monitoring of the count for the skipped atomic block. As the algorithm continues with the backward rewriting, it continuously monitors the count of the delayed atomic block with the help of the function `Rewrite_skip_signal`. When the count of such delayed atomic blocks does not reduce, the algorithm immediately rewrites the delayed atomic block with its corresponding gate polynomial (line 33). While rewriting a delayed atomic block, the algorithm immediately uses the don't care optimization if the atomic block consists of don't care cubes at its inputs. This modification avoids unnecessary backtracking, which otherwise, in the worst case, can deteriorate the performance of the verification tool.

Once the algorithm completes rewriting every atomic block of a divider circuit, it invokes the `evaluate` function (line 39) to evaluate the final polynomial SP_0. As discussed in Section 7, the evaluation of the final polynomial involves checking if the polynomial reduces to zero for all input combinations that satisfy the input constraint: $0 \leq R^{(0)} < D \cdot 2^{n-1}$. The algorithm terminates by returning one if the final polynomial, upon evaluation, reduces to zero, thereby confirming that the implementation satisfies the specification. Otherwise, it returns a zero, suggesting a faulty implementation.

Algorithm 4 Optimized delayed backward rewriting

Input: Specification polynomial SP^{init}, Input Constraint IC, Circuit CUV with atomic blocks $a_1 \prec_{top} \cdots \prec_{top} a_m$ in topological order $\prec_{top}$
Output: 1 iff specification holds for all inputs satisfying IC
$SP_m := SP^{init}$; $oldsize := size(SP_m)$; $i := m$
$ST := \emptyset$; $Skipped\_atomicBlock := \emptyset$
$atomic\_counter := 0$; $Signal\_ST := \emptyset$
$track\_skipped\_signal :=$ False
$(dc(a_1), \ldots, dc(a_m)) :=$ `Compute_DC`(CUV, IC);
while $i > 0$ **do**
  if $a_i \in occurrence\_list$ **then**
    push(ST, $(SP_i, i,$ skip_signal$)$); push(ST, $(Signal\_ST, a_i)$); $oldsize := size(SP_i)$
  end if
  $SP_{i-1} :=$ `Rewrite`(SP_i, a_i); $atomic\_counter = atomic\_counter + 1$;
  if $atomic\_counter = atomic\_count\_threshold$ **then**
    `Update_occurrenceList(`$occurrence\_list$`,` $occurrence\_threshold$`)`; $atomic\_counter := 0$;
  end if
  if $size(SP_{i-1}) >$ threshold $\times$ $oldsize$ **and** $ST \neq \emptyset$ **then**
    $(SP, j, type) :=$ pop(ST); $i := j$;$SP_{i-1} := SP$;
    if $type =$ dc **then**
      $SP_{i-1} := Opt\_DC(SP_{i-1}, dc(a_i))$;
    else if $type =$ skip_signal **then**
      $a_i :=$ pop($Signal\_ST$); $SP_{i-1} :=$ `Rewrite`$(SP_{i-1}; Skip(a_i))$;
      $track\_skipped\_signal :=$ True;
      $Skipped\_atomicBlock[a_i].count := 0$; $Skipped\_atomicBlock[a_i].i := i$;
    end if
  else
    if $dc(a_i) \neq$ empty **then**
      push(ST, $(SP_{i-1}, i,$ dc$)$); $oldsize := size(SP_{i-1})$;
    end if
    for each atomic block $a_i \in Skipped\_atomicBlock$ **do**
      if `Rewrite_skip_signal`$(a_i,$ $Skipped\_atomicBlock[a_i]$.count$) =$ True **then**
        $i := Skipped\_atomicBlock[a_i].i$;
        $Skipped\_atomicBlock[a_i].count := 0$;
        $SP_{i-1} :=$ `Rewrite_With_Opt_DC`(SP_{i-1}, a_i);
      end if
    end for
  end if
  $i := i - 1$
end while
Return `evaluate`(SP0);

9 Experimental Results

The current section presents the experimental evaluation performed on one core of an Intel Xeon CPU E5-2643 with 3.3 GHz and 62 GiB of main memory. The run time of the experiments is limited to 24 CPU hours. As already mentioned, the BDD-based image computations required for computing the don't care cubes are realized with the CUDD 3.0.0 package. Gurobi ILP solver solves the integer linear programming problem for don't care optimization of polynomials.

This section presents the results of two experiments. Starting with Experiment 1, which deploys two optimization methods: optimized backward rewriting [3] and optimized delayed backward rewriting (ODR) on the last stage ($Stage_n$) of the reduced restoring divider. Table 3 illustrates the results obtained from Experiment 1. Column 1 of Table 3 represents the bit width of the reduced restoring divider. Column 2 of Table 3 mentions the starting size of the specification polynomial corresponding to the bit width of the restoring divider. From Equation (20), the size of the specification polynomial for an n-bit divider is equivalent to n^2+2n-2. Column 3 of Table 3 provides the expected polynomial size after backward rewriting replaces the last stage $Stage_n$. From Equation (52), the size of the polynomial obtained after rewriting $Stage_n$ is equivalent to n^2+n-1. Column 4 of Table 3 provides the size of the polynomial obtained after method [3] (optimized backward rewriting) replaces $Stage_n$. Column 5 of Table 3 provides the peak size of the polynomial obtained while method [3] (optimized backward rewriting) replaces $Stage_n$. Similarly, Column 6 of Table 3 provides the size of the polynomial obtained after optimized delayed backward rewriting replaces $Stage_n$. Column 7 of Table 3 provides the peak size of the polynomial obtained while optimized delayed backward rewriting replaces $Stage_n$. Table 3 illustrates a clear comparison between the two optimization methods. It highlights the limitation of the optimization method presented in [3], which fails to avoid exponential blow-up of the intermediate polynomial size. Furthermore, optimized backward rewriting could not complete the rewriting of $Stage_n$ for $n > 12$ bit reduced restoring divider because this method leads to memory explosion. However, optimized delayed backward rewriting successfully rewrites $Stage_n$ for all reduced restoring divider up to bit width $n = 512$-bit without causing exponential blow-up of intermediate polynomial size. Although optimized delayed backward rewriting works well for bigger reduced restoring divider circuits, the BDD-base image computation fails to extract the don't care cubes for restoring divider with bit width greater than 16 bit.

In such cases, simulations that do not violate the input constraint $IC = 0 \leq R^0 < D \cdot 2^{n-1}$ compute the don't care cubes at the inputs of the atomic blocks. However, these don't care cubes are then verified with the help of the theoretical proofs presented in Section 5.5 to determine whether they are indeed true don't care cubes or not.

n	**Start Poly. Size**	$Stage_n$ **Poly. Size**	[3] **Poly. Size**	[3] **Peak Poly. Size**	**ODR Poly. Size**	**ODR Peak Poly. Size**
4	22	19	72	305	19	51
8	78	71	13724	36926	71	143
12	166	155	3495408	9437326	155	267
16	286	271	MO	MO	271	489
24	622	599	MO	MO	599	753
32	1086	1055	MO	MO	1055	1346
48	2398	2351	MO	MO	2351	2400
64	4222	4159	MO	MO	4159	4791
96	9406	9311	MO	MO	9311	10263
128	16638	16511	MO	MO	16511	17783
256	66046	65791	MO	MO	65791	68343
512	263166	262655	MO	MO	262655	267767

Table 3: Comparison of results obtained from the backward rewriting of $Stage_n$

Experiment 2 focuses on deploying the two previously mentioned optimization techniques to verify the reduced restoring divider. Table 4 provides a detailed result obtained from verifying the reduced restoring divider using optimized delayed backward rewriting. Column 1 of Table 4 represents the bit-width of the reduced restoring divider. Column 2 of Table 4 represents the starting polynomial, which is similar to col.2 of Table 3. Column 3 of Table 4 presents the total time in CPU seconds taken by the verification tool to complete the verification process. This total time includes the time required to compute the don't-care cubes, the time spent searching for these don't-care cubes within the intermediate specification polynomial, and the total time spent on backward rewriting. Column 4 of Table 4 highlights the maximum time in CPU seconds taken by verification tool to query a don't care cube including the split cubes of the queried don't care cube using the algorithm 3. Column 5 of Table 4 represents the peak polynomial size obtained during the backward rewriting. The last column of Table 4 represents the final

polynomial size before the tool evaluates the final polynomial under the input constraint (IC). Although the evaluation reduces the final polynomial to zero, thereby confirming that the verification is successful, optimized delayed rewriting fails to avoid exponential blow-up of the intermediate polynomial size. This observation is confirmed by the values present in Column 5 of Table 4, which clearly indicates the substantial increase in the polynomial size corresponding to the bit width of the divider under verification. Moreover, the verification tool with optimized delayed backward rewriting fails to complete the verification of reduced restoring dividers with bit-width greater than 7-bit. With an increase in the intermediate polynomial size, the search space to query don't care cubes in the current polynomial also increase. As a result of this, the time taken to search don't care cubes contained in the intermediate specification polynomial also increases. Column 4 of Table 4 supports this observation.

n	**Start Poly. Size**	**Overall Time**	**Max. Time for DC Search**	**Peak Poly. Size**	**Final Poly. Size**
4	22	0.587s	$<$ 0.01s	1911	125
5	33	2.98s	0.345s	32158	1046
6	46	87.77s	38.45s	517632	8879
7	61	11573.1s	7961.62s	8310566	74289
8	TO	TO	TO	TO	TO

Table 4: Verifying optimized reduced restoring divider

Figure 11 provides the comparison between the peak polynomial size obtained by the two optimization techniques: optimized backward rewriting (DC opt. without delayed rewriting [3]) and optimized delayed backward rewriting (DC opt. with delayed rewriting) when employed to verify the optimized reduced restoring divider. Though in comparison, the peak polynomial size obtained by employing optimized backward rewriting is approximately twice the peak size obtained by employing optimized delayed backward rewriting, both the techniques fail to avoid exponential blow-up of the intermediate polynomial size. Hence, a more robust and efficient optimization method is necessary to verify larger reduced restoring dividers without encountering any memory explosion problem.

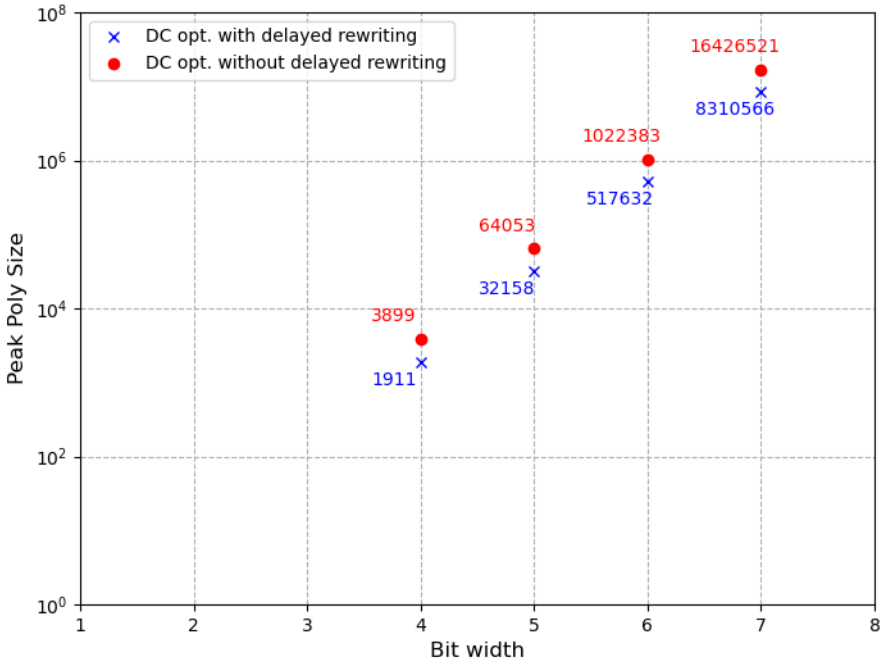

Figure 11: Peak sizes for n-bit reduced restoring divider

10 Conclusion and Outlook

In this thesis, the optimized reduced restoring divider was analyzed and verified. To confirm the correct implementation of a divider circuit, two conditions were crucially verified: $R^{(0)} = Q \cdot D + R$ and $0 \leq R < D$. The first condition was verified using polynomial-based rewriting, while the second condition was confirmed using BDD-based image computation. The optimized reduced restoring divider circuit, considered in this work, was realized using an n-bit carry-ripple adder (CRA) and an n-bit multiplexer. Polynomial-based backward rewriting replaced the non-trivial atomic blocks specifically, the full adder and one-bit multiplexer present in the CRA and n-bit multiplexer, respectively with their corresponding gate polynomials. However, as discussed in Section 5.3, basic backward rewriting failed to avoid the exponential blow-up in the size of intermediate polynomials.

The first optimization technique introduced in the thesis was *don't care* optimization. This technique derived a smaller polynomial P' from the original polynomial P using satisfiability don't cares. By enforcing the input constraint IC: $0 \leq R^{(0)} < D \cdot 2^{n-1}$ on the inputs of the divider, specific assignments at the inputs of certain atomic blocks were eliminated. These assignments, considered as *don't care* cubes, were used to optimize the intermediate specification polynomial. In the case of the reduced restoring divider, the presence of *don't care* cubes at the inputs of atomic blocks FA_{n-1}, FA_{n-2}, MUX_{n-1}, and MUX_{n-2} in $Stage_n$ was demonstrated in Section 5.5. However, the don't care optimization alone could not avoid an exponential blow-up of intermediate polynomial size. The setback of this optimization technique when employed during verification of the reduced restoring divider was shown in Section 5.6.

The current thesis introduced another optimization technique known as delayed backward rewriting, which deviates from the pre-determine topological order and delays the rewriting of a specific signal with high occurrence count to avoid replacing the signal multiple times. As a result of which, the probability of increase in the size of intermediate polynomial reduces. Though this optimization works well for the last stage, that is, $Stage_n$ of the reduced restoring divider, it fails to avoid the memory explosion. The experimental results shown in Section 9 supports the analysis presented in Section 5.8, which highlights the drawback of both the optimization techniques. The lack of don't care cubes and optimized reduced restoring divider's structure makes their verification challenging. Therefore, a much robust optimization techniques are required to combat the difficulties posed by the divider.

There are two possible solutions on hand that could possibly help in verifying the optimized divider circuit effectively. As mentioned in Section 5.8, the absence of the sign bit $r_n^{(n-1)}$ in $R^{(n-1)}$ during rewriting $Stage_{n-1}$ takes away the opportunity to use the don't care cubes present at the inputs of atomic block FA_{n-1} present in $Stage_{n-1}^{SUB}$. As all the partial remainders of a restoring divider must be positive, the sign bit $r_n^{(n-1)}$ of the partial remainder $R^{(n-1)}$ must be equal to zero. This requirement can be used to extend the polynomial representing $R^{(n-1)}$ in Equation(52) . In other words, the specification polynomial after rewriting $Stage_n$ can be rewritten as

$$\sum_{i=1}^{n-1} q_i 2^i \cdot D + (\sum_{i=0}^{n-1} r_i^{(n-1)} 2^i - 2^n r_n^{(n-1)}) - R^{(0)}$$

The above equation resembles Equation (20), due to which, the delayed backward rewriting illustrated in Section 5.7 must also work in the exact same way when employed to rewrite $Stage_{n-1}$. A similar extension of the partial remainder must be made in every predecessor stage to take advantage of the delayed backward rewriting, which then would verify the reduced restoring divider without leading to an exponential blow-up of the polynomial size. Although this approach is simple and straightforward, it is merely a practical approach. The automatic verification tool implemented to verify the divider circuits tend to remove atomic blocks associated with signals that have no destination. As a result of this, the tool removes the most significant multiplexer of each $Stage_j$, where $1 \leq j \leq n-1$, which is associated with the sign bit of the partial remainder $R^{(j)}$. Therefore, implementing the above discussed optimization technique is quite challenging. Moreover, extending the partial remainder is equivalent to widening the divider circuit, doing so eliminates the essence of verifying an optimized divider circuit.

Instead of widening the circuit, another possible solution is to use the don't care cubes to extend the polynomial in such a way that it enables the availability of the don't care cubes at the input of specific atomic block. Then there is a possibility that rewriting the extended polynomial may not lead to exponential blow up of the intermediate polynomial size. In Equation (53), $q_1 \cdot \sum_{i=0}^{n-2} t_i^{(n-1)} 2^{i+1}$ can be extended by adding a don't care cube: $v_1 \cdot q_1 t_{n-1}^{(n-1)}$. Assigning $v_1 = -2^n$, transforms Equation (53) to the following
:

$$\sum_{i=2}^{n-1} q_i 2^i \cdot D + q_1(2D + \sum_{i=0}^{n-2} t_i^{(n-1)} 2^{i+1} - 2^n t_{n-1}^{(n-1)} - \sum_{i=1}^{n-1} r_i^{(n-2)} 2^i) + \sum_{i=0}^{n-1} r_i^{(n-2)} 2^i - R^{(0)}$$

The above equation resembles Equation (44) and also provides the opportunity to utilize the don't care cube $(r_n^{(n-2)}, c_{n-2}^{(n-1)}) = (1, 1)$ present at the inputs of the atomic block FA_{n-1} of $Stage_{n-1}^{SUB}$. Therefore, a similar backward rewriting illustrated in the rewriting of $Stage_n^{SUB}$ can also be followed to replace $Stage_{n-1}^{SUB}$ along with the delayed rewriting of the signal q_1. Doing so will yield $\sum_{i=2}^{n-1} q_i 2^i D + \sum_{i=0}^{n} r_i^{(n-2)} 2^i - R^{(0)}$ as the final polynomial without any exponential blow-up of intermediate polynomial size. The following discussion provides a high-level overview of how this optimization technique could be realized in a verification tool.

1. The tool still employs delayed backward rewriting, which delays the rewriting of signals with a high occurrence count.

2. When the backward rewriting replaces a signal t that occurs together with the delayed signal q in a monomial, then some information on q is introduced during the rewriting process. This step is essential, as it could introduce don't care cubes that might extend the polynomial as intended.

   (a) To add the required don't care cubes to the intermediate specification polynomial, signals s' that are transitive fan-in of the signal q are considered. Amongst these signals, candidate signals s are chosen in such a way that the transitive fan-in of candidate signals s share gates with the transitive fan-in of signal t.

   (b) The tool must compute the don't care conditions on s and q and then add don't care cube with s and q.

3. Instead of using the usual don't care optimization with backtracking, the tool must use delayed don't care optimization (DDCO) [4]. Unlike the usual don't care optimization that immediately optimizes polynomials with respect to a don't care cube as soon as the polynomial contains the input variables of the cube, delayed don't care optimization only adds don't care terms to the polynomial, but delays the optimization until a later time. In other words, DDCO does not use don't care optimization with backtracking but delays the don't care optimization by a specific number of steps [4].

4. In the current case, only steps that affect the don't care terms are counted, that is, a delay of d for a don't care cube with integer variable v_1 means that there is at least one term that is associated with v_1 into which there have been d substitutions after the don't care term was introduced to the intermediate specification polynomial.

5. There is a possibility that the optimization would assign integer variable v_1 with a suitable value that makes the extension of the polynomial fruitful.

In the case of $Stage_{n-1}$ of the reduced restoring divider, the addition of the don't care term $v_1 \cdot q_1 t_{n-1}^{(n-1)}$ to the polynomial takes place according to Step 2. From Steps 3 and 4, v_1 should be assigned the value -2^n, as this would simplify the polynomial after rewriting FA_{n-2} of $Stage_{n-1}^{SUB}$. However, it is essential that the don't care cube $(r_n^{(n-2)}, c_{n-2}^{(n-1)}) = (1, 1)$ must be optimized first.

Additionally, the BDD-based image computation for verifying the second condition $0 \leq R < D$ of the reduced restoring divider was successful for circuits up to 16 bit. The divider circuits with larger bit width could not be verified using this approach, as the size of the BDD increases. The BDD-based image computation for larger divider circuits exceeded the allowed time, due to which, the computation was terminated. The aim still remains to verify the second condition using the BDD-based image computation without causing an increase in the size of the BDD. Overall, the goal is to find a simple yet efficient optimization technique that achieves verification of optimized dividers without causing any memory explosion.

Acknowledgements

I would like to thank all the people who have supported me during the master's thesis and made this work possible.

First and foremost, I would like to thank ...

- Prof. Dr. Christoph Scholl for the kind allocation of the master's thesis topic, support and guidance during the master's thesis.
- My supervisor M. Sc. Alexander Konrad, for the guidance, advice and insight into relevant topics during the thesis.
- My family and friends for their continuous support during my studies.

References

[1] M. Ciesielski, A. Yasin, and J. Dasari, "Functional verification of arithmetic circuits: Survey of formal methods," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2022, pp. 94–99.

[2] A. Yasin, T. Su, S. Pillement, and M. Ciesielski, "Functional verification of hardware dividers using algebraic model," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 257–262.

[3] C. Scholl, A. Konrad, A. Mahzoon, D. Große, and R. Drechsler, "Verifying dividers using symbolic computer algebra and don't care optimization," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1110–1115.

[4] A. Konrad, C. Scholl, A. Mahzoon, D. Große, and R. Drechsler, "Divider verification using symbolic computer algebra and delayed don't care optimization," in *2022 Formal Methods in Computer-Aided Design (FMCAD)*, 2022, pp. 1–10.

[5] K. Hamaguchi, A. Morita, and S. Yajima, "Efficient construction of binary moment diagrams for verifying arithmetic circuits," in *Computer-Aided Design, International Conference on.* Los Alamitos, CA, USA: IEEE Computer Society, nov 1995, pp. 78–82. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCAD.1995.479995

[6] A. Yasin, T. Su, S. Pillement, and M. Ciesielski, "Formal verification of integer dividers:division by a constant," 07 2019, pp. 76–81.

[7] H. Haghbayan and B. Alizadeh, "A dynamic specification to automatically debug and correct various divider circuits," *Integration, the VLSI Journal*, vol. 53, 12 2015.

[8] C. Scholl and A. Konrad, "Symbolic computer algebra and sat based information forwarding for fully automatic divider verification," 07 2020, pp. 1–6.

[9] D. A. Cox, J. Little, and D. O'Shea, *Geometry, Algebra, and Algorithms.* Cham: Springer International Publishing, 2015, pp. 1–47. [Online]. Available: https://doi.org/10.1007/978-3-319-16721-3_1

[10] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 2131–2142, 2016.

[11] M. Ciesielski, T. Su, A. Yasin, and C. Yu, "Understanding algebraic rewriting for arithmetic circuit verification: A bit-flow model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1346–1357, 2020.

[12] D. Harris and S. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[13] B. Becker, *Vorlesungsskript: Technische Informatik*. Institut für Informatik, Arbeitsgruppe für Rechnerarchitektur, Albert-Ludwigs-Universität Freiburg, 2020, unveröffentlicht.

[14] C. Scholl and R. Wimmer, *Verification of Digital Circuits*. Institute of Computer Science, Albert-Ludwigs-Universität Freiburg, 2024, ch. Binary Decision Diagrams, pp. 1–84.

[15] K. H. Rosen, *Discrete Mathematics and Its Applications*, 7th ed. New York: McGraw-Hill Education, 2012.

[16] C. Scholl, "Some notes on arithmetic verification," Freiburg im Breisgau, Germany, 2024, unpublished. [Online]. Available: mailto:scholl@informatik.uni-freiburg.de

[17] A. Mahzoon, D. Große, and R. Drechsler, "Revsca: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[18] C. Scholl and R. Wimmer, "Verification of digital circuits: Chapter 6a - verification of sequential circuits," Lecture Notes, 2024, slides 25–41. [Online]. Available: mailto:scholl@informatik.uni-freiburg.de,mailto:wimmer@informatik.uni-freiburg.de

[19] ——, "Verification of digital circuits: Chapter 6b - verification of sequential circuits: Optimizations," Lecture Notes, 2024, slides 9–19. [Online]. Available: mailto:scholl@informatik.uni-freiburg.de,mailto:wimmer@informatik.uni-freiburg.de