Comprehensive Search for ECO Rectification Using Symbolic Sampling

Victor N. Kravets
IBM Thomas J. Watson Research Center
New York, USA

Nian-Ze Lee   Jie-Hong R. Jiang
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei, Taiwan

ABSTRACT

The task of an engineering change order (ECO) is to update the current implementation of a design according to its revised specification with minimum modification. Prior studies show that the amount of design modification majorly depends on the selection of *rectification points*, i.e., the input pins of gates whose functionality should be rectified with some patch circuitry. In realistic ECOs, as the netlist of the current implementation has been heavily optimized to meet design objectives, it is usually structurally dissimilar to the netlist of a revised specification, which is synthesized only by lightweight optimization. This paper proposes an ECO solution for optimized designs, which is robust against structural dissimilarity caused by design optimization. It locates candidate rectification points in a sampling domain, which significantly improves the scalability of rectification search. To synthesize the circuitry of patches, a structurally independent rewiring formulation is proposed to reuse existing logic in the implementation. Based on the proposed method, a newly developed engine is evaluated on the engineering changes arising in the design of microprocessors. Its ability to derive patches of superior quality is demonstrated in comparison to industrial tools.

1 INTRODUCTION

The engineering change order (ECO) is an important practice in the design methodology of modern semiconductor chips which incrementally updates the current implementation relative to a revised specification. The changes in the specification are often due to a functional bug or the design's evolved functionality. When the ECO is applied in a late design stage, close to chip tape-out for manufacturing, its incrementality of updates prevents the potential mask re-spinning and thereby avoids extra manufacturing costs. An incremental update is also common in a typical chip development cycle, driven by design closure and the often-cited behavioral instability of modern automation tools. Rather than re-running the entire tool-chain as the design evolves, the functional rectification through ECO is realized in the current implementation. It saves on the design effort and avoids dealing with unpredictable tool outcomes.

By the time design reaches sign-off, its structural properties have been aggressively tuned to meet the fed-forward constraints and to find sufficient trade-off among optimization objectives, including timing, area, power dissipation, and routing congestion. The objectives are met much due to the logic transformations performed prior to placement as well as the later downstream physical optimization of a given technology. Thus the restructuring effects of logic sharing and duplication complicate the isolation of functional updates as design may have multiple outputs and include logic paths that are not part of the ECO. The rectification points must be chosen carefully in such path-entangled designs to avoid unnecessary increase of patch sizes.

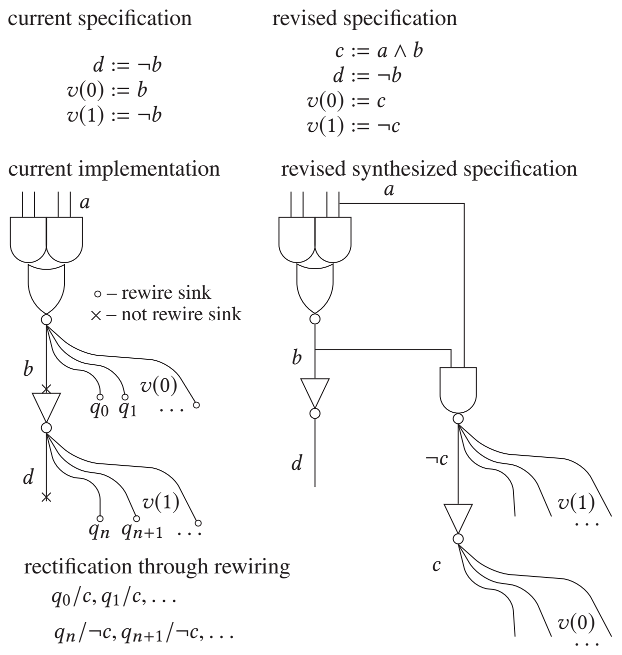

Figure 1: Selection of rewiring sink pins for a solution. Reconnecting all but one sink of nets b and d to c and $\neg c$ respectively rectifies the design.

The potential significance of accurately identifying candidate rectification points is illustrated in Figure 1, where only a partial design is shown. The depicted implementation at its left-hand side contains two nets b and d, each with an arbitrary, and potentially large, number of fanouts (sinks). The revised specification creates a new signal c by *and*-ing signals a and b, and redefines functions of the single-bit multi-sink signals $v(0)$ with sinks $q_0, q_1, \ldots$ and $v(1)$ with sinks $q_n, q_{n+1}, \ldots$. However, the revision does not affect another signal d that depends on b, and thus the signal must be preserved during rectification. Choosing all but one sink of nets b and d as candidate rectification points enables correction of $v(0)$ and $v(1)$ while protecting the logic at a remaining sink of each net. Overlooking such a solution and selecting candidate rectification points past the sinks of $v(0)$ and $v(1)$ would make the rectification costlier.

The difficulty to incrementally update intensively optimized designs motivates an approach that is (i) robust to the structural dissimilarity between heavily restructured existing implementation and lightweight synthesized specification, and (ii) powerful to derive patches with a minimal impact on the already attained quality of the current implementation. We postulate that rather than explicitly synthesizing the rectification logic to perform the design correction, the current implementation and the intermediate representation

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06...$15.00
https://doi.org/10.1145/3316781.3317790

of a revised specification already contain logical structures needed to achieve high-quality updates. Thereby we propose a systematic approach that enumerates functionally (in contrast to structurally) derived candidate rectification points and matches them implicitly against already existing logic to produce updates of the current implementation. The scalability of the proposed method is attained by casting its computations to a *symbolic sampling domain* in which candidate rectifications are analyzed and produced. To demonstrate the value-add of the developed formulation, it is implemented and tuned in new ECO engine, dubbed `syseco` (for `sy`mbolic `s`ampling in `ECO`). It is evaluated on the realistic ECOs arising in the design of microprocessors, with no restriction to an error model.

The main results of this paper include:

- A new rectification flow for optimized designs: It finds rectification points functionally to achieve resilience to structural dissimilarity, and performs ECO reusing existing logic from either current implementation or an intermediate representation of new specification.
- A scalable symbolic sampling technique: It is applied to symbolically pose and solve Boolean reasoning queries that are relevant to the rectification flow, thereby making the solution search more comprehensive and scalable.
- A thorough experimental evaluation in an industrial setting: The ECO patches generated using the new techniques measure 5x smaller on average, compared to a modern industrial tool. The multiple output patches are obtained on a diverse variety of real ECOs. The patch quality also meets a designer's estimate for an ideal update to the current implementation.

The remainder of the paper is structured as follows. Section 2 gives an account of prior work, relevant to ECO. The terminology and formalism of studied problem are given in Section 3. The choices in computing an ECO patch are stated in Section 4. Section 5 introduces the notion of symbolic sampling and describes its use in the search for a design update. Experimental results are given in Section 6. Finally, Section 7 concludes this paper and highlights potential future directions.

2 PRIOR WORK

Given a netlist C representing the current implementation, which is synthesized from the original specification $\mathcal{S}$, the functional revisions considered in this work, which modify $\mathcal{S}$ to the revised specification $\mathcal{S}'$, are restricted to changing the *combinational logic* of the design only, in contrast to sequential rectification [10]. Such *(combinational) ECOs*, as we shall assume in the sequel, are of significant relevance to industrial practice. Their automated solutions have been actively researched since mid 1980s. Prior ECO solutions vary considerably in the way of locating the rectification points and how their update logic is determined. They can be roughly classified into two fashions: *structural* and *functional* approaches. We remark that, however, these two fashions are not orthogonal to each other and can be exploited at the same time for a hybrid approach.

Structural approaches update C using a form of signal correspondence derived from structural similarity between C and the netlist C' synthesized from $\mathcal{S}'$. In [2], the process that incrementally updates C is supplemented with the original specification $\mathcal{S}$ in addition to the revised specification $\mathcal{S}'$. The proposed solution detects structural difference between $\mathcal{S}$ and $\mathcal{S}'$, and then updates C by relating it to $\mathcal{S}$. Its drawback is that the quality of patches hinges on the extent of functionally equivalent signals between $\mathcal{S}$ and C. It also presumes the sufficiently low-level representation of specifications $\mathcal{S}$ and $\mathcal{S}'$ for the accurate isolation of their structural differences. The approach in [8] emphasizes structural similarities between C and C'. It derives a patch boundary matching signals of C and C' from both primary inputs and outputs, thus making the logic implementation of an update readily available. Such an approach however, places a stability burden on synthesis tools to retain the structural similarity of functionally unchanged portions in the specification, making the process volatile in practice. The authors in [14] make the process more stable using the concept of a hint, which is supplemented manually to deduce a difference between $\mathcal{S}$ and $\mathcal{S}'$. A hint is then synthesized into C', annotating functionally changed signals to limit the matching search between C and C'. The work in [7] enhances the matching procedure by introducing a new "lock-step" upstream traversal of structurally similar current and revised implementations. In contrast, our proposed approach is *structure independent*, that is, it does not rely on structure-driven traversal that attempts to match candidate rectification points against the specification. Hence our method is more robust against structural dissimilarity.

Functional approaches are based on rigorous Boolean reasoning and impose no assumption on structural similarity between C and C' in identifying rectification points and constructing patch logic. One early formulation [9] relies on the Boolean equation solver to rectify a design implementation under a single-fault assumption modifying a gate function. The work in [11, 18] places the update logic on exterior of the current implementation, thus preserving its already generated layout. The approach in [6] addresses the multiple error ECOs, incrementally correcting a failing output using a pruning heuristic to locate candidate rectification points and binary-decision diagrams (BDDs) [3] to synthesize an update. Although effective on the studied test-cases, the heuristic could be overly conservative when handling diverse ECOs arising in practice, leading to needlessly large patches. A formulation based on satisfiability (SAT) to diagnose rectification points is studied in [15], and further explored in the context of rectification logic synthesis by [12]. The approach relies on the explicit netlist replication to model possible assignments to input signals, thus creating a conjunction of SAT instances. The theory of Craig interpolation [4] was popularized to derive the patch logic from a companion SAT formulation. The use of Craig interpolation in ECO was first introduced in [19] to perform a single fault rectification, and then later extended to iterative concretization of a patch over the provided set of multiple rectification points [5, 17, 20]. The construction of patches with interpolation-based approaches benefits from the *conjunctive normal form* (CNF) representation of a revised specification. Methods that are not restricted by an ECO type and are independent of a revised specification representation are still the area of on-going research. The work in [13] gives a brief overview of the relevant efforts, and studies synthesis of a single-point rectification using Boolean learning in the *exists-forall* (EF) formulation.

Apart from functional ECO for logic rectification, there are prior efforts that focus on other ECO issues, such as timing and spare cell usage, which are out of the scope of this paper.

3 RECTIFICATION FORMULATION

The ECO problem statement and the proposed rewire-based rectification formulation are given in this section.

3.1 Design representation

An informal terminology is provided below, to facilitate the presentation of the studied problem. The combinational logic of a design is implemented by a *Boolean circuit* C, whose outputs are computed as a function of its input values at the moment. Individual inputs and outputs have unique labels that are used to establish the behavioral correspondence between two circuits C and C'. The logic operations in a circuit are modeled using *gates*. These operations are performed on a gate's binary inputs, producing a single binary output. A *net* in the circuit connects gates at their input and output *pins*, and may also connect to the input or output pins of a circuit itself. A net carries information from its single *source* pin to the downstream *sink* pins. We assume that a circuit remains *well-formed*, i.e., all of its pins are connected to a net, and it has no topological cycles.

3.2 ECO problem statement

Consider the combinational logic of a design with n inputs and k outputs, whose behavior is described by a multi-output Boolean function $\mathbf{f} \equiv (f_1, \dots, f_k) : \mathbb{B}^n \to \mathbb{B}^k$. For a changed specification $\mathbf{f}' \equiv (f'_1, \dots, f'_k)$, the *ECO problem* asks to rectify the circuit implementation C of $\mathbf{f}$ to meet the changed specification. To solve the ECO problem, it suffices to search for a vector of pins $(p_1, \dots, p_m)$ at gate inputs or possibly at circuit outputs in C, and change the functions of their driving nets. Let $\mathbf{h} \equiv (h_1, \dots, h_k)$ denote the *composition function* that is computed at outputs of C while treating the pins $(p_1, \dots, p_m)$ as free circuit inputs. The ECO problem is then stated in the form of a *functional decomposition* as:

$$\text{Find } \mathbf{h} \text{ and } \mathbf{r} \text{ such that } \mathbf{f}' = \mathbf{h}(\mathbf{r}), \tag{1}$$

where $\mathbf{h}$ composes individual functions of $\mathbf{r} \equiv (r_1, \dots, r_m)$ at the inputs that correspond to pins $(p_1, \dots, p_m)$. We refer to $\mathbf{r}$ as a *rectification function*, highlighting the behavioral change in the specification that it captures. The pins $(p_1, \dots, p_m)$ are referred to as *rectification points*.

3.3 Rewire-based rectification formulation

To minimize the design perturbation and hence realize high-quality ECO, instead of synthesizing the rectification function $\mathbf{r}$ from scratch, we restrict the search space of $\mathbf{r}$ to existing nets within the current implementation C and synthesized specification C', and rectify the design by replacing the driving nets of rectification points by other nets. This restriction does not affect the completeness: It accommodates any revision because a circuit output is also a rectification point, with its rectification function r being the revised function f', realized at the corresponding output in C'. The *rewire-based rectification* is formally stated as follows.

The *rewire* operation $\mathcal{R}$ disconnects a vector of pins $(p_1, \dots, p_m)$ from their driving nets and connects them to the corresponding nets $(s_1, \dots, s_m)$. The operation is denoted as $\mathcal{R} \equiv p_1/s_1, \dots, p_m/s_m$. A topological constraint is imposed on pins involved in the rewire operation: No path should connect any pair of the pins. This restriction simplifies the rectification analysis and retains well-formed (acyclic) structure of the current implementation. The incremental update through rewiring leads to the formulation that is central to the algorithmic flow of design rectification described in this paper.

Proposition 1. *Given a current implementation C and a new synthesized specification in the form of a circuit C', for every corresponding pair of outputs, the* rewire-based rectification *finds the rectification points $p_1, \dots, p_m$ such that their individual rectification functions $r_1, \dots, r_m$ are realized by corresponding nets $s_1, \dots, s_m$ of either C or C'.*

When the rewire-based rectification produces a rewire operation $\mathcal{R} \equiv p_1/s_1, \dots, p_m/s_m$, we say $\mathcal{R}$ *rectifies* logic of C. We note that, if a net s belongs to circuit C', then its logic copy is instantiated in C and the rewire operation is performed by connecting a clone of s.

4 COMPREHENSIVE RECTIFICATION SEARCH

This section provides a computational rigor for achieving the rewire-based rectification in three steps: identifying feasible rectification point-sets, selecting their candidate rewiring nets, and combining thereof into rewiring choices that rectify a circuit output.

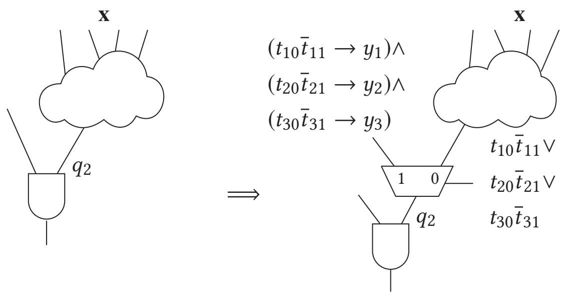

Figure 2: Parameterized selection of a rectification point using binary expansion encoding of a pin index: $\mathbf{t}_i^2 \equiv \bar{t}_{i0}t_{i1}$ encodes the decision of choosing pin q_2 as i-th rectification point, $1 \le i \le 3$.

4.1 Boolean constructs notation

The following terminology is introduced to describe the rewire-based rectification. The vector $\mathbf{x} \equiv (x_1, \dots, x_n)$ denotes variables at circuit inputs. As rectification points are treated as free circuit inputs, they are associated with variables $\mathbf{y} \equiv (y_1, \dots, y_m)$. The composition function at each output of the circuit is denoted as $h(\mathbf{x}, \mathbf{y})$. A *cube* is a conjunction of *literals*, each being a variable or the negation (complement) of a variable. If removing any literal from a cube voids its containment by some given function, it is *prime* relative to that function. A *minterm* with respect to a variable vector $\mathbf{v}$ is a cube with the presence of a literal of every variable in $\mathbf{v}$. We denote the binary code of integer i with variables $\mathbf{v} = (v_1, v_2, v_3)$ as $\mathbf{v}^i$, assuming the "big endian" ordering of i's bits, e.g., $\mathbf{v}^3$ refers to $\overline{v}_1 v_2 v_3$. An *assignment* to the variable vector $\mathbf{v}$ is denoted as $\hat{\mathbf{v}}$. A *truth (satisfying) assignment* is an assignment to the variables of a cube/function that evaluates the cube/function to true.

4.2 Feasible rectification point-sets

To rectify an output of the circuit, we wish to select at most m rectification points out of a set of M sink pins $\{q_0, \dots, q_{M-1}\}$ of the circuit. A technique to implicitly enumerate all subsets of $\{q_0, \dots, q_{M-1}\}$ that are feasible rectification point-sets with maximum size m, is instrumented as follows:

(1) For each rectification point y_i, the parametric variables $\mathbf{t}_i$ are allocated. The truth assignment to a minterm $\mathbf{t}_i^j$ then selects pin q_j identified by j for rectification point y_i. One possible way to encode the selection of a pin is to use the binary representation of its identifier, although other encodings are possible.

(2) For a pin q_j, the expression $\mathbf{t}_1^j \vee \dots \vee \mathbf{t}_m^j$ determines whether the pin is selected for any rectification point or not. If a term $\mathbf{t}_i^j$ in the expression is true, then q_j gets disconnected from its original net and is connected to the free input y_i. (Multiple selections of q_j are merged into a single rectification point.) On the other hand, if the above expression is false, the pin is not selected for any rectification point, and retains its existing connection.

This parameterized selection is realized within the current implementation netlist using the multiplexer logic. A multiplexer is introduced for each pin, with the *selection* signal being $\mathbf{t}_1^j \vee \dots \vee \mathbf{t}_m^j$, the *data-0* input being the original net of the pin, the *data-1* input being the expression $(\mathbf{t}_1^j \to y_1) \wedge \dots \wedge (\mathbf{t}_m^j \to y_m)$, and the output connected to the pin. Figure 2 shows an example of how the circuitry is modified to realize the parameterized selection for a given pin q_2 when three rectification points are considered.

The parameterized function $h(\mathbf{x}, \mathbf{y}, \mathbf{t})$ is then computed on the augmented netlist, encompassing diagnostics for rectification points. The characteristic function $\mathcal{H}(\mathbf{t})$ of all feasible rectification point-sets of maximum size m can be expressed as

$$\mathcal{H}(\mathbf{t}) \equiv \forall \mathbf{x} \exists \mathbf{y} (h(\mathbf{x}, \mathbf{y}, \mathbf{t}) = f'(\mathbf{x})) \tag{2}$$

With Eq. (2), we enumerate prime cubes of $\mathcal{H}(\mathbf{t})$ and use them as seeds to construct an explicit list of candidate rectification point-sets. If a selected prime cube depends on variables $\mathbf{t}_i$ and it contains $\mathbf{t}_i^j$, then rectification point y_i admits choice of pin q_j.

Example 1. *Let the specification in Figure 1 be extended with*

$$\mathbf{w}_{out} := \mathit{GATE}(\mathbf{w}_{in1}, v(0)) \vee \mathit{GATE}(\mathbf{w}_{in2}, v(1)),$$

where GATE *denotes the bitwise* and*-ing of a word with a single-bit signal; the logical $\vee$ is bitwise. We assume that $\mathbf{w}_{in1}$ and $\mathbf{w}_{in2}$ are n-bit words, $\mathbf{w}_{out} \equiv (w_0, \ldots, w_{n-1})$ are circuit outputs, and an output w_k $(0 \leq k \leq n-1)$ depends on sinks q_k and q_{n+k}, which carry the k-th bit value of $\mathbf{w}_{in1}$ and $\mathbf{w}_{in2}$, respectively. Suppose that the rectification points are to be chosen from the circuit pins $\{q_0, \ldots, q_n, \ldots, q_{2n-1}, \ldots\}$. The decision variables $\mathbf{t}_i$ for each rectification point i $(1 \leq i \leq m)$ select one of these pins based on the truth assignment that represents the binary code of a pin subscript. When $m = 2$, the existence of rectification for an output w_k is established by Eq.* (2)*, which yields:*

$$\mathcal{H}_k(\mathbf{t}_1, \mathbf{t}_2) \equiv \mathbf{t}_1^k \mathbf{t}_2^{n+k} \vee \mathbf{t}_1^{n+k} \mathbf{t}_2^k$$

The two product terms in the expression provide identical solutions: Both encode the same pair of indices $\{k, n+k\}$ that represent rectification points $(p_1, p_2) \equiv (q_k, q_{n+k})$ for a given output w_k.

As the number of variables required to identify a pin among M pins is $\lceil \log_2 M \rceil$, the total number of newly allocated variables when using binary representation of a pin index is $m \cdot \lceil \log_2 M \rceil$ for m rectification points.

4.3 Candidate rewiring nets

For a candidate rectification point associated with some pin q, a set of candidate rewiring nets is also determined. A candidate rewiring net s is chosen from both the current implementation and the synthesized specification. We rely on structural filtering, followed by a functional heuristic to find candidate rewiring nets. If structural input dependence at output of f' contains transitive fanins of net s, then net s is chosen as a candidate rewiring net. The functional heuristic assesses behavioral difference between functions of the original net at pin q and candidate rewiring net s in the *error domain* $\mathbb{E} \equiv \{\hat{\mathbf{x}} \mid f(\hat{\mathbf{x}}) \neq f'(\hat{\mathbf{x}})\}$. Specifically, for a pin q in the logic of f from C, the *rectification utility* of a candidate rewiring net s is computed as $|\{\hat{\mathbf{x}} \mid \hat{\mathbf{x}} \in \mathbb{E} \wedge q(\hat{\mathbf{x}}) \neq r(\hat{\mathbf{x}})\}|/|\mathbb{E}|$, where $q(\mathbf{x})$ and $r(\mathbf{x})$ are functions of the original net at pin q and candidate rewiring net s, respectively. This ratio exploits the heuristic that the more pronounced difference between functions of a candidate rectification point and a rewiring net is more likely to rectify the errors of $\mathbb{E}$.

4.4 Candidate rewiring choices

A method that finds a rectification function $\mathbf{r} \equiv (r_1, \ldots, r_m)$, whose components represent functions of the rewiring nets $(s_1, \ldots, s_m)$, is described below. According to the statement in Eq. (1), the rewire operation $p_1/s_1, \ldots, p_m/s_m$ rectifies the current implementation of f if and only if f' and $h(\mathbf{r})$ are consistent. The consistency can be checked by directly applying the rewire operation and evaluating the equivalence, or analytically as stated below.

We denote the consistent assignments induced by $\mathbf{r}$ for input variables $\mathbf{x}$ and rectification point variables $\mathbf{y}$ by:

$$R(\mathbf{x}, \mathbf{y}) \equiv (y_1 = r_1(\mathbf{x})) \wedge \cdots \wedge (y_m = r_m(\mathbf{x}))$$

The necessary and sufficient condition for the existence of a rectification function $\mathbf{r}$ is stated in the following theorem.

Theorem 1. *The function* $\mathbf{r}$ *rectifies the implementation at an output if and only if its composition function $h(\mathbf{x}, \mathbf{y})$ satisfies both of the implications below:*

$$L(\mathbf{x}, \mathbf{y}) \Rightarrow h(\mathbf{x}, \mathbf{y}) \tag{3}$$

$$h(\mathbf{x}, \mathbf{y}) \Rightarrow U(\mathbf{x}, \mathbf{y}) \tag{4}$$

where $L(\mathbf{x}, \mathbf{y}) \equiv f'(\mathbf{x}) \wedge R(\mathbf{x}, \mathbf{y})$ and $U(\mathbf{x}, \mathbf{y}) \equiv f'(\mathbf{x}) \vee \neg R(\mathbf{x}, \mathbf{y})$.

We examine possible rectification choices by matching rectification points to the candidate rewiring nets. Suppose that a rectification point y_i is allocated an ordered set of candidate rewiring nets $S_i \equiv (s_{i0}, s_{i1}, \ldots)$. The choices of candidate rewiring nets are encoded using decision variables $\mathbf{c} \equiv (\mathbf{c}_1, \ldots, \mathbf{c}_m)$, which parameterizes $R(\mathbf{x}, \mathbf{y})$ as:

$$R(\mathbf{x}, \mathbf{y}, \mathbf{c}) \equiv \bigwedge\nolimits_i (\mathbf{c}_i^0 \rightarrow y_i = r_{i0}(\mathbf{x})) \wedge (\mathbf{c}_i^1 \rightarrow y_i = r_{i1}(\mathbf{x})) \wedge \cdots$$

Note that $r_{i0}, r_{i1}, \ldots$ are the functions of candidate rewiring nets $s_{i0}, s_{i1}, \ldots$, respectively. The selection of a candidate rewiring net from S_i is induced by an assignment to $\mathbf{c}_i$. Substituting $R(\mathbf{x}, \mathbf{y}, \mathbf{c})$ in Eq. (3) and (4), and universally quantifying $\mathbf{x}$ and $\mathbf{y}$, we obtain a characteristic function of all valid rewire operations:

$$\Xi(\mathbf{c}) \equiv \forall \mathbf{x}, \mathbf{y} (L(\mathbf{x}, \mathbf{y}, \mathbf{c}) \Rightarrow h(\mathbf{x}, \mathbf{y}) \wedge h(\mathbf{x}, \mathbf{y}) \Rightarrow U(\mathbf{x}, \mathbf{y}, \mathbf{c}))$$

An assignment to $\mathbf{c}$ determines a choice of a rewire operation that changes $f(\mathbf{x})$ to $f'(\mathbf{x})$.

Example 2. *We extend Example 1 to illustrate the selection of rewiring nets for pins $\{q_k, q_{n+k}\}$ to rectify the logic of output w_k. Let $S_1 \equiv (v(0), c, \neg c)$ and $S_2 \equiv (v(1), c, \neg c)$ be rewiring candidates for q_k and q_{n+k}, respectively. The function of rewiring choices R is then*

$$(\mathbf{c}_1^0 \rightarrow y_1 = r_{v(0)}) \wedge (\mathbf{c}_1^1 \rightarrow y_1 = r_c) \wedge (\mathbf{c}_1^2 \rightarrow y_1 = r_{\neg c}) \wedge$$

$$(\mathbf{c}_2^0 \rightarrow y_2 = r_{v(1)}) \wedge (\mathbf{c}_2^1 \rightarrow y_2 = r_c) \wedge (\mathbf{c}_2^2 \rightarrow y_2 = r_{\neg c})$$

Substituting it into the computational form of $\Xi(\mathbf{c})$, and universally quantifying out the input and rectification variables we obtain

$$\Xi_k(\mathbf{c}_1, \mathbf{c}_2) \equiv \mathbf{c}_1^1 \vee \mathbf{c}_2^2$$

which represents rewiring $\mathcal{R} \equiv q_k/c, q_{n+k}/\neg c$ that rectifies w_k.

5 APPLICATION OF SYMBOLIC SAMPLING

The computational efficiency of the proposed symbolic approaches in Section 4 is potentially hindered by design complexity. This section introduces a scalable implementation for the proposed computations, based on the notion of *symbolic sampling*, and outlines the full algorithmic flow for design rectification.

5.1 Casting computation to symbolic sampling domain

Given a circuit C with input variables $\mathbf{x} \equiv (x_1, \ldots, x_n)$, a *sampling domain* is a set of assignments to variables $\mathbf{x}$. The number of sampled assignments in a domain trades off the desired degrees of precision versus computational complexity. Given a sampling domain with N assignments $\{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_N\}$, a set of $\mathbf{z}$ variables of size $\lceil \log_2 N \rceil$ is introduced to encode those assignments. A *sampling function* $\mathbf{g} \equiv (g_1, \ldots, g_n) : \{\hat{\mathbf{z}}\} \rightarrow \{\hat{\mathbf{x}}\}$ is used to capture the sampling domain, and is computed as the matrix product:

$$\left[\mathbf{z}^0, \ldots, \mathbf{z}^{N-1}\right] \begin{bmatrix} — & \hat{\mathbf{x}}_1 & — \\ — & \hat{\mathbf{x}}_2 & — \\ & \vdots & \\ — & \hat{\mathbf{x}}_N & — \end{bmatrix} = [g_1(\mathbf{z}), \ldots, g_n(\mathbf{z})]$$

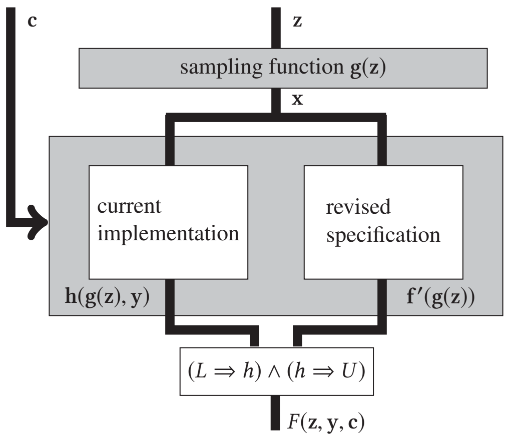

Figure 3: Computation of $\Xi(\mathbf{c}) \equiv \forall \mathbf{z}, \mathbf{y} F(\mathbf{z}, \mathbf{y}, \mathbf{c})$ in the sampling domain. The original domain $\mathbf{x}$ is overloaded with the sampling function $\mathbf{g}(\mathbf{z})$.

where the multiplicand is an $N\times n$ binary matrix comprised of the sampling domain assignments. The input variables $\mathbf{x}$ of circuit C are then overloaded with the sampling function $\mathbf{g}(\mathbf{z})$ to cast the original circuit operation from its exact domain of $\mathbf{x}$ to the sampling domain of $\mathbf{z}$.

Although reasoning in the sampling domain yields a super-set of candidates and hence require a validation step, it makes the computation robust for the designs of high complexity. The accuracy of computations in the domain benefits from selecting samples carefully. For design rectification, the computation yields fewer false positives when sampled assignments are from the error domain $\mathbb{E}$. Figure 3 illustrates the working of symbolic sampling, casting the computation of feasible rewire operations from Section 4.4 to the sampling domain.

For implementation, we use BDDs to represent the sampling domain, which offers efficient operations for abstracting variables and counting assignments. The judicious choice of total variables used to represent the domain leads to the reduced complexity of the abstraction operations, as well as to efficient counting of consistent value assignments to inputs. The contained memory footprint of the domain makes computations robust and independent of the design size and its logical complexity. The choices of rewire operations computed in the sampling domain are subsequently validated with a resource-constrained SAT solver during the search for a preferred solution.

5.2 Overall algorithmic flow

The proposed rewire-based design rectification algorithm *RewireRectification* casts computation steps from Section 4 to a sampling domain. Its working is summarized as follows. Given the current implementation C and the revised specification C', *RewireRectification* iterates over the corresponding output pairs (p_o, p'_o) that remain non-equivalent, and:

(1) selects error samples to construct a sampling domain (Section 5.1)
(2) enumerates feasible rectification point-sets (Section 4.2)
(3) assigns candidate rewire nets for each rectification point (Section 4.3)
(4) finds choices of rewire operations $\mathcal{R}$ for each feasible rectification point-set (Section 4.4)
(5) uses SAT solving to validate $\mathcal{R}$ constructed in the sampling domain

The process finds candidate rectification points $(p_1, \dots, p_m)$ for the individual output pairs (p_o, p'_o), sorted in the increasing order of their logical complexity. Such local context ensures that the computation scales well with the design size increase, although the single-output view may occasionally overlook candidates that are more economical for multiple outputs. In contrast to the rectification points, the selection of candidate rewiring nets $(s_1, \dots, s_m)$ is done in the global context, across all the outputs that depend on the constructed rectification $\mathcal{R} \equiv p_1/s_1, \dots, p_m/s_m$: (i) the candidate rectification is pruned if it "damages" already rectified outputs, and (ii) the rectification is favored if it corrects the largest number of outputs. Thus, in spite of candidate rectification points being discovered for a single representative output, the impact of a constructed $\mathcal{R}$ is assessed on all its depending outputs.

Since for a given m the derived $\mathcal{R}$ may over-approximate the number of needed rectification points, a net that is already connected to each candidate pin p_j is also included as a trivial rewiring candidate when computing $\Xi(\mathbf{c})$ (Step 3). As an additional post-processing step, the patch inputs are refined through a sweeping technique that reuses already existing current implementation logic, thereby reducing the patch size.

6 EXPERIMENTAL RESULTS

The presented concepts and algorithms were used to implement a design rectification engine `syseco`. The engine is written in C++, and its source code has about 30,000 lines in total length. The line count excludes implementation of the internal data model to represent a design logic, the satisfiability solver MiniSAT [16], and the in-house BDD package.

To assess the benefits of the developed engine, we obtained a suite of 11 test cases comprised of real ECOs in industrial designs. Each of the designs comes with an optimized implementation of the original specification, and VHDL description of the revised specification. We synthesize the netlist C' from the revised specification, producing its technology-independent representation. An ECO update is then realized on C relative to C'.

Table 1 lists characteristics of the derived combinational netlists for the test cases, identified numerically in the first column of the table. For each of the representatives, the table gives counts for the following attributes of their logic: input and output ports; gates, nets, and net sinks in the original implementation. The number of outputs affected by the revised specification, and their percentage relative to the total outputs in the design, are in the last two columns of the table.

Table 2 gives a broader characteristic of the test cases in the suite. We consulted designers to obtain an expected patch size for each of the representatives; column 2 provides the advised measurements in terms of a modern library cell count. The data is helpful as it gives a practical goal that may go beyond what a known automated solution achieves. For a wider variety of reference points, we also compute patches for each of the test cases using the default setting of a commercially available tool. The attributes of generated patches are in columns 3-6 of the table, they count the patch inputs, outputs, gates, and nets. They offer guidance and are not meant to conclude optimization quality of the tool, as we did not seek technical advice from its vendor.

Table 1: Characteristics of ECO test cases.

	Design implementation					Revised part	
	inputs	outputs	gates	nets	sinks	outputs	%
1	21047	11811	238961	260008	473008	1344	11.3
2	66	37	313	384	735	25	67.5
3	25124	23404	379784	404910	815995	1920	8.2
4	4836	1071	49941	54778	81962	158	14.7
5	524	296	2607	3131	6063	136	45.9
6	11721	3956	82872	94595	163066	15	0.3
7	5472	3655	59993	65466	123852	350	9.5
8	2541	1106	24302	26844	46737	220	19.8
9	651	530	4923	5574	10114	27	5.0
10	2157	1666	15827	17984	33877	108	6.4
11	6941	3930	53052	59993	104375	128	3.2

Table 2: Comparison of the patch attributes from four different sources: a designer's estimate, a commercial tool, DeltaSyn and `syseco`.

	Designer's estimate	Commercial tool				DeltaSyn					`syseco`				
	technology cells	inputs	outputs	gates	nets	inputs	outputs	gates	nets	time, *h:m:s*	inputs	outputs	gates	nets	time, *h:m:s*
1	800	2210	2113	5992	8204	7397	6328	9417	16816	>92:00:00[a]	1117	367	1025	1945	01:20:09
2	30	62	162	243	308	50	39	129	180	00:00:20	32	32	15	41	00:00:39
3	300	2620	2324	5371	7994	5070	4789	6117	11188	>92:00:00[a]	358	115	177	457	00:11:44
4	15	75	58	137	214	56	11	62	118	00:12:06	34	8	26	54	00:02:14
5	10	70	38	309	38	25	9	15	40	00:00:23	0	12	0	1	00:01:24
6	50	68	17	71	14	164	20	139	387	02:26:09	106	9	76	175	00:06:05
7	1000	97	235	212	311	1837	994	4502	6341	06:33:34	189	85	163	277	00:42:03
8	100	249	289	215	467	341	365	315	658	01:09:36	244	237	32	256	00:01:33
9	100	120	79	203	326	447	183	570	1019	00:27:18	62	17	56	111	00:00:23
10	20	113	102	81	195	240	116	240	480	00:00:31	190	108	95	213	00:00:16
11	20	194	150	325	522	320	347	513	835	00:19:24	9	3	3	11	00:02:23
										average reduction ratios relative to DeltaSyn:	**0.35**	**0.47**	**0.17**	**0.21**	

[a] The test case timed out after 92 hours of execution. The patch was then created using the tool's lightweight setting.

In Table 2, patches generated by `syseco` are compared against the DeltaSyn tool. The implementation of the core reasoning engine invoked in the DeltaSyn experiments is based on the patent description in [1]. The columns 7-10 characterize DeltaSyn patches using the same attributes as the four columns for the "commercial tool"; similarly, attributes of the `syseco` patches are given in columns 12-15. The averages of reduction ratios achieved by `syseco` over DeltaSyn for each of these attributes are at the bottom of the table. The runtime consumed by the rectification in both of the engines is also provided in columns 11 and 16 of the table. On the two larger test cases, DeltaSyn exceeded the allocated time, and their rectification was achieved with a lightweight, based on [8], setting of the tool that tends to produce an inferior solution. The results in the table convey following observations:

- `Syseco`-generated patches have fewer gates and nets, with the average reduction ratios being 0.17 and 0.21. The tool's patches are superior to those generated by DeltaSyn on all of the test cases.
- The number of patch outputs in the `syseco`-generated patches is twice fewer than in those produced by DeltaSyn. The reduction points to the benefits of the implemented rewire-based rectification.
- Despite the minimal performance tuning invested in `syseco` implementation, its runs consume less time overall when compared to DeltaSyn. The tool scales well on the larger test cases, where DeltaSyn times out.

The sizes of generated patches also conform to the designer's estimate, conveyed in terms of patch size in the second column of Table 2. Benefits of the `syseco` are further apparent when comparing its patches to the results of a commercial tool (columns 3-6 in the table).

The `syseco` tool has been used in the timing closure loop of high-performance designs whose functional specification undergone a revision. Table 3 illustrates the impact of DeltaSyn and `syseco` patches on the timing of several of those designs. The table compares the tools in terms of the gate count in the produced patches, and their impact on the design slack, measured after each design is placed and routed. The observed timing improvements (i.e., increased slack) with `syseco` are a result of the level-driven optimization decisions that the tool uses as an additional qualitative measure when selecting rewire operations.

Table 3: Rectification impact on design slack.

	DeltaSyn patch		`syseco` patch	
	gates	slack,ps	gates	slack,ps
12	268	-27	47	-14
13	208	-44	207	-36
14	640	-100	236	-66
15	378	0.11	244	1.43

7 CONCLUSIONS AND FUTURE WORK

The rewire-based rectification studied in this work offers a robust platform to perform incremental design updates and assumes no restriction on the error type. Its novel symbolic formulation derives from functional decomposition and is equipped with the domain sampling technique that makes comprehensive solution search scalable. For future work, further improvements to the presented flow lie in rectification logic synthesis and in sampling domain selection. The concept of symbolic sampling has potential application to other problems, e.g., to post-silicon debug.

REFERENCES

[1] E. Arbel, D. Geiger, V. N. Kravets, S. Krishnaswamy, R. Puri, and H. Ren. Logic modification synthesis. U.S. Patent 8,365,114, issued Jan. 29, 2013.

[2] D. Brand, A. Drumm, S. Kundu, and P. Narain. Incremental synthesis. In *Proc. ICCAD*, pp. 14-18, 1994.

[3] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677-691, 1986.

[4] W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *The Journal of Symbolic Logic*, 22(3): 250-268, 1957.

[5] A.-Q. Dao, N.-Z. Lee, L.-C. Chen, M. P.-H. Lin, J.-H. R. Jiang, A. Mishchenko, and R. K. Brayton. Efficient computation of ECO patch functions. In *Proc. DAC*, pp. 51:1-51:6, 2018.

[6] S.-Y. Huang, K.-C. Chen, and K.-T. Cheng. AutoFix: A hybrid tool for automatic logic rectification. *IEEE Trans. CAD*, 18(9): 1376-1384, 1999.

[7] S.-L. Huang, W.-H. Lin, and C.-Y. Huang. Match and replace: A functional ECO engine for multi-error circuit rectification. *IEEE Trans. CAD*, 32(3): 467-478, 2013.

[8] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri. DeltaSyn: An efficient logic difference optimizer for ECO synthesis. In *Proc. ICCAD*, pp. 789-796, 2009.

[9] J. C. Madre, O. Coudert, and J. P. Billon. Automating the diagnosis and the rectification of digital errors with PRIAM. In *Proc. ICCAD*, pp. 30-33, 1989.

[10] N.-Z. Lee, V. N. Kravets, and J.-H. R. Jiang. Sequential engineering change order under retiming and resynthesis. In *Proc. ICCAD*, pp. 109-116, 2017.

[11] I. Pomeranz and S. M. Reddy. On diagnosis and correction of design errors. In *Proc. ICCAD*, pp. 500-507, 1993.

[12] H. Riener and G. Fey. Exact diagnosis using Boolean satisfiability. In *Proc. ICCAD*, pp. 53-58, 2016.

[13] H. Riener, R. Ehlers, and G. Fey. CEGAR-based EF synthesis of Boolean functions with an application to circuit rectification. In *Proc. ASP-DAC*, pp. 251-256, 2017.

[14] H. Ren, R. Puri, L. Reddy, S. Krishnaswamy, C. Washburn, J. Earl, and J. Keinert. Intuitive ECO synthesis for high performance circuits. In *Proc. DATE*, pp. 1002-1007, 2013.

[15] A. Smith, A. G. Veneris, M. F. Ali, and A. Viglas. Fault diagnosis and logic debugging using Boolean satisfiability. *IEEE Trans. CAD*, 24(10): 1606-1621, 2005.

[16] N. Sörensson and N. Eén. Minisat v1. 13: A SAT solver with conflict- clause minimization. In *Proc. SAT Competition*, pp. 53-54, 2005.

[17] K.-F. Tang, P.-K. Huang, C.-N. Chou, and C.-Y. Huang. Multi-patch generation for multi-error logic rectification by interpolation with cofactor reduction. In *Proc. DATE*, pp. 1567-1572, 2012.

[18] Y. Watanabe and R. K. Brayton. Incremental synthesis for engineering changes. In *Proc. ICCD*, pp. 40-43, 1991.

[19] B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. R. Jiang. A robust functional ECO engine by SAT proof minimization and interpolation techniques. In *Proc. ICCAD*, pp. 729-734, 2010.

[20] H.-T. Zhang and J.-H. R. Jiang. Cost-aware patch generation for multi-target function rectification of engineering change orders. In *Proc. DAC*, pp. 96:1-96:6, 2018.