Learning to Automate the Design Updates From Observed Engineering Changes in the Chip Development Cycle

Victor N. Kravets
IBM T. J. Watson Research Center
New York, NY
kravets@us.ibm.com

Jie-Hong R. Jiang
National Taiwan University
Taipei, Taiwan
jhjiang@ntu.edu.tw

Heinz Riener
EPFL
Lausanne, Switzerland
heinz.riener@epfl.ch

***Abstract*—The behavioral revisions to the design are frequent in the late stage of the semiconductor chip development. Quite often, their realization emphasizes incrementality that seeks minimum perturbation of the existing implementation. This tutorial paper poses the engineering change order (ECO) problem as the functional decomposition and proposes its solution in the form of the Boolean equations. We hope that the sufficient generality of the statement will be useful in extending the existing state-of-the-art design revision techniques. To assist in this process, we present an observed variety of design revisions encountered in the chip development cycle. The knowledge of such frequent and realistic ECOs is essential in advancing a tool's ability to yield compact implementation updates. We believe that sharing our experience of practical ECOs would benefit the research community in developing an open-source tool.**

I. Introduction

The engineering change order (ECO) is an essential practice in the chip development cycle that incrementally updates the current implementation relative to a revised specification. Such revisions are often a result of the functional bugs discovered late in the cycle or may target an improved operation of the current implementation. To avoid the often volatile task of rerunning the design automation tool-chain or the costly regeneration of masks, the ECO asks to rectify the current circuitry of design incrementally, with minimal incurred penalty to its already attained quality. The research to automate the ECO task spans several decades and continues contributing to the maturity of available software tools.

Various factors may inhibit the potential quality of the automated updates, including the relative complexity of the revision, the design closure effort already invested in the current implementation, the stability of tools used to obtain the implementation, among others. These and other factors drive practical decisions when crafting an automated rectification approach. The ultimate challenge, however, is making an ECO tool that separates these concerns, and reliably handles a broad set of chip design scenarios.

In recent years the research community made notable contributions towards the functional ECO methods. The advances in Boolean satisfiability (SAT) engines now enable answers to propositional queries for larger designs. The ECO methods can localize errors and offer their correction under certain assumptions; both are crucial steps in automating the ECO solutions. The popularized application of satisfiability solvers also extends into domains of quantified Boolean logic. Its use offers concise statements of rectification [9]; whose synthesized patch depends less on the acquired structure of a revised specification. However, the general-purpose solvers, even for the quantified Boolean logic, is still a developing subject in research. The limited scalability of the solvers gives prominence to the ECO algorithms that integrate the medium of heuristic reasoning. These algorithms also enable the controlled rectification quality outside of a solver.

This paper poses the ECO problem in the form of functional decomposition. The rigor of Boolean equations is used to derive a closed-form representation that captures the functional rectification choices. The formulation does not depend on the representation form of the revised specification. The stated equations, however, raise the scalability concerns even when applied to small designs. We, therefore, propose to overload the said computation with a sampling domain that yields an over-approximated solution. Validating the presented hypotheses while learning from their falsifying counter-examples leads to a rectification answer or proves that it is not possible. The work in [10] studies a constrained variant of this approach that uses the synthesized revised specification.

We illustrate the scope of the design rectification problem with the examples of ECOs observed during the chip development cycle. The encountered examples are described in the context of their connecting logic. The knowledge of such realistic ECOs is vital in the development of a robust rectification tool, to yield compact implementation updates. We believe that sharing our experience with practical ECOs would benefit the research community in developing ever-improved, open-source tools.

The remainder of the paper is organized as follows. Section II states the task of ECO using the rigor of Boolean reasoning. The symbolic sampling approach to assist in solving the stated computational forms is given in Section III. Section IV describes the types of revisions in specification during the chip development. Section V concludes this paper summarizing potential future directions.

978-3-9819263-4-7/DATE20/©2020 EDAA

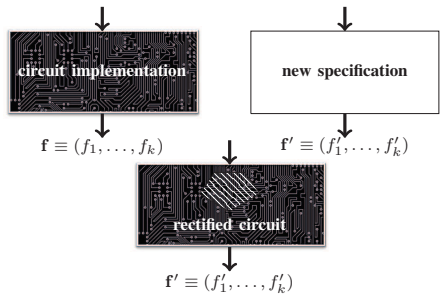

Figure 1. ECO problem asks to rectify implementation of $\mathbf{f}$ to meet new specification $\mathbf{f}'$.

II. Rectification Using Boolean Reasoning

This section uses Boolean reasoning to formulate the ECO problem for the design rectification. It states the computational methods for finding a rectification.

A. Functional ECO formulation

The work in [10] formulates the functional ECO problem using the notion of functional decomposition. The formulation is re-stated below, establishing the terminology for further discussion.

Consider a Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^k$ that describes behaviour of an n-input k-output combinational logic. For a changed specification $\mathbf{f}' \equiv (f'_1, \ldots, f'_k)$, the *ECO problem* asks to *rectify* the k-output circuit implementation $\mathcal{C}$ of $\mathbf{f} \equiv (f_1, \ldots, f_k)$ to meet the revision (see Figure 1 illustration). To solve the ECO problem, it suffices to search for a vector of pins $(p_1, \ldots, p_m)$ at gate inputs or possibly at circuit outputs in $\mathcal{C}$, changing the functions of their driving nets. Let $\mathbf{h} \equiv (h_1, \ldots, h_k)$ denote the *composition function* that is computed at the outputs of $\mathcal{C}$ while treating the pins $p_1, \ldots, p_m$ as free circuit inputs. The circuit rectification then finds $\mathbf{h}$ and $\mathbf{r}$ such that $\mathbf{f}' = \mathbf{h}(\mathbf{r})$, where $\mathbf{h}$ composes individual functions of $\mathbf{r} \equiv (r_1, \ldots, r_m)$ at the inputs that correspond to pins $p_1, \ldots, p_m$.

We refer to $\mathbf{r}$ as a *rectification function*, highlighting the behavioral change in the specification that it captures. The pins $(p_1, \ldots, p_m)$ are referred to as *rectification points*. They define outputs of a *patch* that updates the current implementation $\mathcal{C}$. Optimization also selects the patch inputs from existing signals in the circuit implementation, with the common goals that include reduced size of patch, and its improved connectivity.

B. Validation of candidate rectification

For the given functions $\mathbf{f}'$ and $\mathbf{h}$, we can determine valid choices for the rectification function $\mathbf{r}$ from the analytically derived consistency of equation $\mathbf{f}' = \mathbf{h}(\mathbf{r})$.

Let $\mathbf{y} \equiv (y_1, \ldots, y_m)$ be variables of the m rectification points, and let f' and h denote the index-corresponding single-output functions from $\mathbf{f}'$ and $\mathbf{h}$. Given a rectification function $\mathbf{r}(x)$, the following characteristic function expresses consistent input-output behavior of its circuit implementation:

$$R(\mathbf{x}, \mathbf{y}) \equiv (y_1 = r_1(\mathbf{x})) \wedge \cdots \wedge (y_m = r_m(\mathbf{x})) \qquad (1)$$

It also serves as the characteristic function of a *care-set* that determines the choice of h: the values of h must agree with the values of f' in the set, and can be arbitrary outside of it. With this observation, the function r rectifies an output of h if and only if the following inequality holds for all assignments to $\mathbf{x}$ and $\mathbf{y}$:

$$\underbrace{f'(\mathbf{x}) \wedge R(\mathbf{x}, \mathbf{y})}_{L(\mathbf{x},\mathbf{y})} \leq h(\mathbf{x}, \mathbf{y}) \leq \underbrace{f'(\mathbf{x}) \vee \neg R(\mathbf{x}, \mathbf{y})}_{U(\mathbf{x},\mathbf{y})} \qquad (2)$$

Using the short-hand notation for the lower and upper bounds, the above inequality reduces to a single equation:

$$\phi(\mathbf{x}, \mathbf{y}) \equiv \neg L(\mathbf{x}, \mathbf{y}) \wedge \neg h(\mathbf{x}, \mathbf{y}) \vee h(\mathbf{x}, \mathbf{y}) \wedge U(\mathbf{x}, \mathbf{y}) = \mathbf{1} \quad (3)$$

The universal consistency for all input values of $\phi(\mathbf{x}, \mathbf{y})$, i.e. the equation tautology, determines validity of the presented rectification function or else prunes it. For the general case of a k-output circuit, the consistency of the product computed over all the given outputs validates the rectification: $\prod_{i=1}^{k} \phi_i(\mathbf{x}, \mathbf{y}) = \mathbf{1}$.

C. Synthesis of rectification

We continue by deriving a closed-form computation for the functional choices of a patch. The stated computational form makes no use of structure acquired by the representation of a revised specification $\mathbf{f}'$, treating it in this sense as a black box. It synthesizes the patch expressing its functional choices using internal signals in the circuit implementation $\mathcal{C}$. Thus any of the signals from $\mathcal{C}$ that keeps it well-formed is a legitimate patch input candidate, although potentially superfluous.

Let $\mathbf{s} \equiv (s_1, \ldots, s_K)$ be variables that uniquely identify the K candidate patch inputs, and let the $t_i(\mathbf{x})$ functions ($1 \leq i \leq K$) describe their individual behavior. Analogous to the equation (1), the input-output consistency of these signals is captured by

$$T(\mathbf{x}, \mathbf{s}) \equiv (s_1 = t_1(\mathbf{x})) \wedge \cdots \wedge (s_K = t_K(\mathbf{x})) \qquad (4)$$

The meaning of inequality (2) extends when its $R(\mathbf{x}, \mathbf{y})$ function is replaced with the $T(\mathbf{x}, \mathbf{s})$ function. The substitution introduces an implicit relation between the $\mathbf{s}$ and $\mathbf{y}$ variable values that control the consistency of the inequality. The resulting generalized inequality can be solved for the consistent values of $\mathbf{x}$ and $\mathbf{y}$ obtaining their explicit characterization $\mathcal{R}(\mathbf{s}, \mathbf{y})$. Indeed, let us reduce the inequality to an equation $\psi(\mathbf{x}, \mathbf{s}, \mathbf{y})$, analogous to how $\phi(\mathbf{x}, \mathbf{y})$ derives from (2). The solution $\mathcal{R}(\mathbf{s}, \mathbf{y})$ must hold independent of all $\mathbf{x}$ values, and is computed as

$$\mathcal{R}(\mathbf{s}, \mathbf{y}) \equiv \forall \mathbf{x} \psi(\mathbf{x}, \mathbf{s}, \mathbf{y}) \qquad (5)$$

For $\mathcal{R}(\mathbf{s}, \mathbf{y})$ to yield a valid patch function, its relation must be *total* in the space of $\mathbf{s}$:

$$\exists \mathbf{y} \mathcal{R}(\mathbf{s}, \mathbf{y}) = \mathbf{1}.$$

There could be a variety of distinct patch functions that are characterized by $\mathcal{R}(\mathbf{s}, \mathbf{y})$. To solve the relation for a patch function, the process of *determinization* (see, e.g., [2] [8]) is applied to $\mathcal{R}(\mathbf{s}, \mathbf{y})$). It yields a deterministic relation $R(\mathbf{s}, \mathbf{y})$ compatible with $\mathcal{R}(\mathbf{s}, \mathbf{y})$ that identifies a unique output

response to each input of the patch. A practical pre-processing step in the determinization is to eliminate as many candidate patch inputs as possible, with the justifiable hope that it leads to reduced patch size, or other measure. The elimination can be done through the universal quantification of a carefully chosen subset of the $\mathbf{s}$ variables while ensuring that the resulting relation remains total.

The approach in [6] describes a tailored heuristic for eliminating candidate patch inputs in the context of SAT-solving. The solution avoids the explicit construction of rectification choices. The SAT-based formulation, however, is not stated in the closed form. It constructs the patch one output at-a-time, successively deriving the permitted output behavior in the form of an interpolant [5] [12] from a companion satisfiability (SAT) problem. The approach uses the CNF to represent a revised specification, and its innate structure may influence the choice of the interpolant function that ultimately determines the patch quality.

An optimal selection of patch inputs that works well on the ECOs encountered in practice is not a solved problem yet. Although the existing approaches for the optimal support computation of incompletely specified functions [11] can be adopted when the patch characterizations $\mathcal{R}(\mathbf{s}, \mathbf{y})$ have a small number of variables (i.e., a few dozen).

D. Rectification points enumeration

A set of m pins, designated by the independent variables $\mathbf{y}$, forms the rectification points if and only if the following equation is consistent:

$$\eta(\mathbf{x}, \mathbf{y}) \equiv (\forall \mathbf{x} \exists \mathbf{y} (\mathbf{f}'(\mathbf{x}) = \mathbf{h}(\mathbf{x}, \mathbf{y}))) = 1 \qquad (6)$$

The formula states that for every assignment to the $\mathbf{x}$ variables, there must be an assignment to the $\mathbf{y}$ variables, which makes values of $\mathbf{f}'$ and $\mathbf{h}$ equal. The equation consistency may also be demonstrated by finding the *Skolem functions* for variables $\mathbf{y}$ such that substituting $\mathbf{y}$ with their respective Skolem functions in equation (6) yields a tautology. In this way, the Skolem functions serve as the rectification function $\mathbf{r}$, which amounts to a more complex problem of solving the Section II-A functional decomposition equation $\mathbf{f}' = \mathbf{h}(\mathbf{r})$. Deriving a patch for already validated rectification point-sets while pruning the falsified sets, reduces the ECO automation effort. However, enumeration of the pin-sets that give a feasible rectification remains computationally challenging.

Heuristic approaches (such as in [7]) that assess a point's controllability of outputs are prone to conservative decision making when $m > 1$. The formulation in [15] uses implicit enumeration, instrumenting the diagnostics with decision variables that determine the rectification point-sets. The approach is stated in the context of error localization using SAT-solving. It is further studied with the application to rectification logic synthesis in [13]. In the other work [10], the search space encoding draws an analogy with the classical facility location problem (see, e.g., [3]), that assigns a customer (i.e., rectification variable y_i) to a facility (i.e., circuit pin p_j). The work uses auxiliary variables that parameterize the $\eta(\mathbf{x}, \mathbf{y})$ computation to obtain a characteristics function of rectification point-sets.

The inherent combinatorial complexity of enumerating the rectification points may also be addressed algorithmically. Instead of the simultaneous revision of all outputs, the iterative rectification of individual non-equivalent output pairs (or of their carefully selected subsets) reduces the complexity. It divides the search into more manageable subproblems, whose deduced patches tend to have a small number of rectification points. They may also be structurally shared with other outputs, possibly yielding their simultaneous correction of logic.

III. Application of Sampling

This section discusses a sampling method to improve the scalability of the computational forms for fulfilling an ECO.

A. Symbolic sampling

The computational forms stated in Section II may not scale well on the larger designs. Their binary-decision diagrams (BDD) representation is particularly sensitive to the design size parameters, such as the number of input variables $\mathbf{x}$. As a result, for example, the construction of characteristic function $T(\mathbf{x}, \mathbf{y})$ for the candidate patch inputs could become prohibitive even when the number of candidate inputs K is small.

While adapting the studied Boolean reasoning notions to the SAT-solving is possible, the *symbolic sampling* method, proposed in [10], retains the derived computational structures, and solves the derived equations in their earlier stated form. Given a formula that contains input variables $\mathbf{x} \equiv (x_1, \ldots, x_n)$, a *sampling domain* is a set of assignments to $\mathbf{x}$. The assignments may also be viewed as bit patterns. For a sampling domain with N bit patterns $\{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_N\}$, a set of $\mathbf{z}$ variables of size $\geq \lceil \log_2 N \rceil$ is introduced to encode those patterns. The encoding is defined by a vector of mutually orthogonal functions $(e_0(\mathbf{z}), \ldots, e_{N-1}(\mathbf{z}))$. In a straight-forward encoding choice, each function e_i picks a single assignment to $\mathbf{z}$ that valuates it to the truth. A *sampling function* $\mathbf{g} \equiv (g_1, \ldots, g_n)$ then uses a chosen encoding to map the $\mathbf{z}$ assignments to the $\mathbf{x}$ assignments, and is computed as the matrix product:

$$\big[e_0(\mathbf{z}), \ldots, e_{N-1}(\mathbf{z})\big] \begin{bmatrix} — & \hat{\mathbf{x}}_1 & — \\ — & \hat{\mathbf{x}}_2 & — \\ & \vdots & \\ — & \hat{\mathbf{x}}_N & — \end{bmatrix} = \big[g_1(\mathbf{z}), \ldots, g_n(\mathbf{z})\big]$$

where the multiplicand is an $N \times n$ binary matrix comprised of the sampling domain patterns.

The overloading of variables $\mathbf{x}$ with the sampling function $\mathbf{g}(\mathbf{z})$ casts the formula operation from its exact domain of $\mathbf{x}$ to the sampling domain of $\mathbf{z}$. This way, the symbolic sampling method trades off the desired degrees of precision versus computational complexity. It makes computations more robust as the formula no longer depends on the design inputs $\mathbf{x}$. For example, the re-stated characteristic function $T(\mathbf{g}(\mathbf{z}), \mathbf{y})$ represents well with BDDs, as opposed to the original function in formula (4) that depends on the design-specific $\mathbf{x}$ variables. Similarly, replacing $\mathbf{x}$ with the sampling domain $\mathbf{g}(\mathbf{z})$, over-approximates the search for rectification points in equation (6).

B. Learning a rectification

The work in [10] gives a symbolic formulation that parameterizes equation (3) with choices for the rectification function construction. Thus, the formula solutions are the hypotheses that need validation. The formulation uses the pre-synthesized representation of the revised specification to obtain the "building blocks" r_i for the potential rectification functions $\mathbf{r}$. Their enumeration prioritizes functions that maximize logic sharing with the current circuit implementation, and hence lead to smaller patches.

In general, if the formula has variables $\mathbf{x}$ universally quantified, then its behavior gets over-approximated in the sampling domain. Indeed, the replacement of variables $\mathbf{x}$ in formula (5) yields a new relation $\breve{\mathcal{R}}$ that contains the original $\mathcal{R}$:

$$\breve{\mathcal{R}}(\mathbf{s},\mathbf{y}) \equiv \forall \mathbf{z} \psi(\mathbf{g}(\mathbf{z}),\mathbf{s},\mathbf{y})$$
$$\breve{\mathcal{R}}(\mathbf{s},\mathbf{y}) \succeq \mathcal{R}(\mathbf{s},\mathbf{y}).$$

The computed function $\breve{\mathcal{R}}$ describes an incomplete specification of the candidate rectification function. The specification can be concretized with an iterative process, to yield a rectification solution. The determinized relation R of $\breve{\mathcal{R}}$ is a hypothesis that serves counter-examples in the form of $(\hat{\mathbf{s}},\hat{\mathbf{y}})$ assignments, if falsified. They constrain the $\breve{\mathcal{R}}$ relation in the subsequent selections of R. The refinement loop continues until the correct rectification function is found, whose existence is guaranteed by the initial choice of $\breve{\mathcal{R}}$. Such an iterative process follows the steps of counterexample-guided inductive synthesis (CEGIS), originating from abstraction refinement (CEGAR) [4]. The application of CEGIS to the ECO problem was studied by [14] to synthesize Boolean functions in sum-of-products form with a bounded number of terms. This approach relies on the incremental solving feature of modern SAT solvers that enables the solver to reuse information from previous solver calls. Each obtained counter-example is ruled out by strengthening the relation until finally a valid patch is found.

IV. Learnt Engineering Changes

This section describes a collection of industrial ECOs encountered in the course of chip development. The corresponding test cases of the studied ECOs are available online.[1] The test cases come as Verilog files that list equations for their NAND2/INV netlist representation and are derived from the experimental setting in [10], keeping the numeric value in their names. They capture the combinational portion of the design that is relevent to their ECO. The suite is devoid of any technology-specific information, thereby simplifying its use and facilitating its portability. The test cases point to valuable considerations for developing a capable ECO engine. They also emphasize the significance of conditions in which an update is needed. Even a small change to an HDL specification may lead to a substantial perturbation of the current implementation when realizing the update.

[1]ECO Benchmark Suite, https://doi.org/10.5281/zenodo.3588388

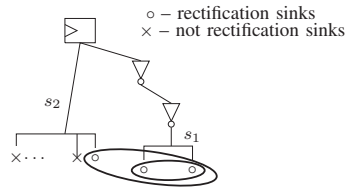

Figure 2. The relevant part of the circuit implementation in example 2. The s_1 and s_2 wires represent the HDL variable s. Not all of these sinks are the rectification points. In the design, two rectification points (in the small ellipse) revise 15 of the outputs. The revision of the remaining 113 outputs uses all three rectification points.

1. *Signal introduced into an expression.* This `design3` example has a very simple ECO that asks to compose a latch output with the existing expression, rewriting the statement

$$c \texttt{ <= } a \texttt{ and not } b$$

as

$$c \texttt{ <= } a \texttt{ and not } b \texttt{ and } d$$

The requested change resides in the entity that gets instantiated 48 times, each instance influencing 40 of outputs `design3`. The circuit implementation underwent the aggressive optimization of the ABC tool [1], which makes finding good rectification points challenging. This example illustrates how a simple change may require a patch with over a hundred of inputs and outputs. A robust tool should handle such a situation efficiently, observing the relative independence of rectification points.

2. *Logic introduced into an expression.* This `design11` rectification is similar to the previous ECO. It extends the HDL expression

$$y \texttt{ <= ... or (} s \texttt{ ...}$$

adding a simple logic. The introduced logic uses latch outputs $\mathbf{q}$`(0 to 2)` that represent the 1-hot encoding of three states restricting operational semantics of the state bit s:

$$y \texttt{ <= ... or ((} s \texttt{ and not (} \mathbf{q}\texttt{(0) or } \mathbf{q}\texttt{(1) or } \mathbf{q}\texttt{(2)) ...}$$

We can rectify an implementation of the initial expression composing the inverted s and $\mathbf{q}$ signals into a single 4-input `nor` gate. The challenging part is to find the correct set of pins to form the rectification points. Figure 2 illustrates the distribution of the signal s sinks in the optimized circuit implementation. The signal s in the figure is represented with two wires s_1 and s_2. This example emphasizes the significance of using sinks rather than nets to define candidate rectification points.

3. *Logically deep and replicated change.* In this `design1` example, the ECO replaces

$$f \texttt{ <= } a \texttt{ and } b \texttt{ and (} c \texttt{ and not } d\texttt{)}$$

with

$$f \texttt{ <= } a \texttt{ and } b \texttt{ and ((} c \texttt{ and not } d\texttt{) or } e\texttt{)}$$

The introduced signal e extends the controlling semantics of the f function, forcing an FSM transition to a particular state. Similar to example 1, the design places it within an entity that gets instantiated four dozen times:

```
for i in 0 to 47 generate
    machine : ENTITY ... fsm(i) ...
```

The aggressive optimization of a flat design leads to a much-varied realization of the multiple instances, testing a tool's vulnerability to a structural difference in the implementation. The logical depth of the rectification points in the ECO emphasizes the benefit of the "sweeping" technique that explores functionally equivalent signals to reduce the patch at its inputs.

4. *Boolean operator changed.* This `design2` example depicts the significance of selecting rectification points carefully. The update modifies a Boolean operation in two of its HDL expressions, replacing the logic of

```
d <= (a and not b) and not c
e <= ( ... ) or not c
```

with

```
d <= (a and not b) or not c
e <= ( ... ) or c
```

The two changed signals d and e are inputs to a transition relation function of a finite-state machine (FSM). The HDL description specifies the transition relation explicitly listing its terms. It is defined on the present-state variables **s**, the two inputs `d` and `e`, and the next-state variables **t** as

```
with (s & d & e) select
   t <= "100...011" when "000...0010",
        "100...001" when "000...0011",
         ...
```

Although the requested change is relatively simple, the context of the change places a special emphasis on accurate selection of rectification points. During synthesis and optimization, the logic of the d and e signals may get absorbed into the transition relation, making it difficult to isolate the rectification points.

5. *Spare latches used.* This `design3` is the first on the list that makes use of spare latches, whose data inputs are tied initially to a constant, extending the sequential behavior of a design. The ECO replaces constant inputs of the two reserved latches with a simple next-state logic. The present-state signals of the two latches are used to correct the logic of 12 outputs changing functions at the sinks of two nets, that correct 8 and 4 outputs independently. The remaining "error" outputs are rectifiable through three small functional changes at the sinks of a particular net. We observed three such rectification nets in total, each respectively correcting 139, 4, and 1 output. The last single output is timing-critical, whose circuit structure is much-optimized with an in-house tool. As a result, its update may use more gates than the other corrected outputs. This example emphasizes the robust handling of the aggressively optimized critical paths.

6. *Signal dropped from an expression.* In this `design5` example a signal is removed from the expression, leading to its simplification. Its ECO changes the behavior of a 6-bit wide signal, replacing the logic of the individual bit i ($0 \le i \le 5$):

```
d(i)<=and_reduce(not(a_i(0 to 1)) or
                 b(0 to 1)) or not(c(i))
```

with the simpler variant

```
d(i)<=and_reduce(b(0 to 1)) or not(c(i))
```

Such simplification of a logic expression has the compact realization that replaces the variable with a constant. The Fig. 3 schematics illustrate the ECO update for this example, replacing its $\mathbf{a}_i$ signals with the positive voltage VDD signal. The top portion (a) in the figure depicts the multi-sink signals $\mathbf{a}(i)$ shared with current implementation already. The rectification must omit the changing circuit locations that are not part of the requested ECO. This example emphasizes the significance of using constants for the simplified operation of a design.

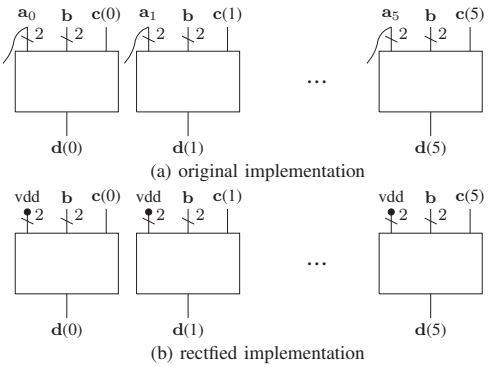

Figure 3. In the ECO of example 6, a signal is dropped for a logic expression. It is fulfilled reconnecting sinks of the 2-bit signals $\mathbf{a}_i$ ($0 \le i \le 5$) to VDD.

7. *Change in the next state decision-making.* The schematic shown in Figure 4 depicts the context of the next state change for this `design6` example. In the schematic, the ECO is isolated to a functional change of blocks $\mathbf{r_1}$ and $\mathbf{r_2}$. The two blocks are controlled by the incoming multi-bit signals $\mathbf{s}_1$ and $\mathbf{s}_2$, whose logic gathers the relevant state of the design, producing the encoding of its deduced meaning at the outputs of $\mathbf{r_1}$ and $\mathbf{r_2}$.

There is a total of 18 next-state signals to which the 6-bit output values of $\mathbf{r_1}$ and $\mathbf{r_2}$ are getting forwarded to complete the decision-making. The operations on the 6-bit signals are

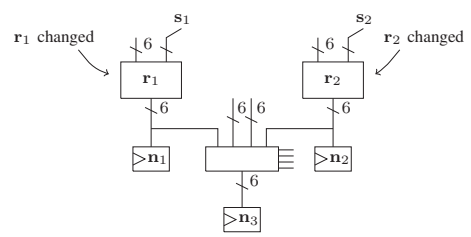

Figure 4. The ECO of example 7 asks to change functions $\mathbf{r}_1$ and $\mathbf{r}_2$, to correct the design behavior.

performed bit-by-bit, thereby forming bit slices of their logic. The revision in the block $\mathbf{r}_1$ changes 3 out of 6 next-state functions of the $\mathbf{n}_1$ latches. Similarly, the $\mathbf{r}_2$ revision changes all 6 next-state functions of the $\mathbf{n}_2$ latches. Collectively, the $\mathbf{r}_1$ and $\mathbf{r}_2$ revision changes all 6 functions of the $\mathbf{n}_3$ latches. The latched logic of $\mathbf{n}_3$ is more complex than of the other next-state signals, as its additional inputs are used to account for the broader state of the design.

This example illustrates how a separate rectification of more straightforward functions may combine to automatically correct the larger ones.

8. *Bit slices depend on the change.* This `design7` example contains several ECOs. Each of the revisions has a simple statement in HDL, and their combinational logic is independent from one another. The difficulty in their implementation however, is in the large number of outputs that have their function changed. The most challenging revision among the requested ECOs extends the functionality of an 8-bit signal $\mathbf{b} \equiv (x_0, \ldots, x_7)$ padding it with an additional bit a

$$\mathbf{b}\texttt{(0 to 8) <= (}x_0\texttt{ \& ... \& }x_7\texttt{ \& }a\texttt{)}$$

In the revised statement below, when the signal a value is 1, it prevents any of the $\mathbf{b}$ string-matches:

```
y1 <= EQ(b, "1---0--00")
y2 <= EQ(b, "01---00-0")
y3 <= EQ(b, "00010--00") or
      EQ(b, "0-0101-00")
```

While only a few outputs in the design depend on the y_1 variable, there is a total of five 64-bit register banks that rely on the other y_2 and y_3 variables. As a result, the logic of their functions is shared bit-wise, yielding a large number of bit slices that depend on the signals y_2 and y_3. This implies a large number of the rectification to revise the next-state logic of the wide register banks. The example illustrates, how operations on the wide signals can lead to a large number of rectification points. A robust tool should be able to handle this dimension of rectification complexity robustly.

9. *Sequential behaviour overriding using spare latches.* This `design8` example uses spare latches to define the conditional override of design behavior. In Figure 5, the combinational block A has its current implementation function extended. The figure schematic omits signals that are not relevant to the revision. The ECO task extends the sequential behavior of the design, introducing the next-state logic B to spare latches, whose data inputs are tied initially to a constant. The 16 outputs of the purposed latches get connected to block A, conditionally substituting data of the current implementation. The multiplexer decision is driven by a single bit signal whose value gets deduced from the design state. This example emphasizes robust handling of the purposed latches, of their new logic that extends design behavior.

V. Conclusions and Future Work

The automation of the design revisions in the chip development cycle remains a challenging problem. The ECO examples in this paper short-list the revision types that a versatile tool should handle. By no means the list of the presented types is complete. However, the exhibit lays the groundwork objectives in designing the ECO algorithms that are scalable. Their synthesized rectification patches should adhere to the already attained quality of the current implementation. The varying requirements for the logic depth emphasize the robust handling of arbitrary logic complexity. At the same time, the automated rectification should account for a design flow constrain where the structural similarity between the current circuit and its synthesized revised specification is not an available option. Although the Boolean reasoning methods described in the paper do not offer readily available solutions to these concerns, their succinct capture is one of the intentions. We hope that the research community finds its use in academic tool development.

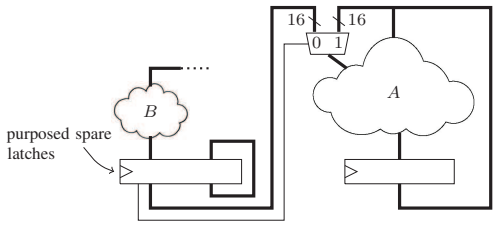

Figure 5. In the ECO of example 9, the spare latches purposed with the next-state logic B extend the functionality of block A in current implementation.

References

[1] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification, http://www.eecs.berkeley.edu/~alanmi/abc/.

[2] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Proc. DATE*, pp. 1188-1193, 2007.

[3] A. Bumb. *Approximation algorithms for facility location problems.* PhD Thesis, Univ. Twente, The Netherlands, 2002.

[4] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, Counterexample-guided abstraction refinement. In *Proc. CAV*, pp. 154–169, 2000.

[5] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.

[6] 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.

[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] J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung. Interpolating functions from large Boolean relations. In *Proc. ICCAD*, pp. 779-784, 2009.

[9] J.-H. R. Jiang, V. N. Kravets, and N.-Z. Lee. Engineering Change Order for Combinational and Sequential Design Rectification. In *Proc. DATE*, 2020.

[10] V. N. Kravets, N.-Z. Lee, J.-H. R. Jiang. Comprehensive search for ECO rectification using symbolic sampling. In *Proc. DAC*, pp. 71:1-71:6, 2019.

[11] B. Lin. Efficient symbolic support manipulation. In *Proc. ICCD*, pp. 513-516, 1993

[12] K. L. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV*, pp. 1-13, 2003.

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

[14] 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.

[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.