УДК 004.272.2 DOI: 10.17587/it.24.755-762

А. Л. Стемпковский, д-р техн. наук, академик РАН, науч. руководитель,
Д. В. Тельпухов, канд. техн. наук, зав. отд.,
Р. А. Соловьев, канд. техн. наук, вед. науч. сотр., **Ю. В. Битков,** науч. сотр.,
Институт проблем проектирования в микроэлектронике РАН (ИППМ РАН)
e-mail: nofrost@inbox.ru

Разработка методов автоматизации ресурсоориентированной функциональной коррекции логических схем

В современном процессе проектирования СБИС нередко возникают ситуации, когда неправильная работа схемы обнаруживается лишь на этапе заключительной верификации. Хуже, когда ошибки проявляются даже после заключительного этапа проектирования или производства микросхем. Исправление ошибок на таких поздних стадиях требует огромных трудовых и финансовых затрат. Для минимизации этих затрат были разработаны методы внесения функциональных изменений на поздних стадиях проектирования (engeneering change order, ECO) для внесения правок в окончательный проект вместо того, чтобы выполнять полное перепроектирование.

В работе предложены методы автоматизации функциональной коррекции схем на базе структурного анализа, формальных методов и симуляционных подходов. Реализованы программные средства, демонстрирующие высокую эффективность на ряде реальных задач.

Ключевые слова: *engeneering change order (ECO), функциональная коррекция, конфликты, ICCAD Contest*

Введение

Методы внесения функциональных изменений на поздних стадиях проектирования *ECO* (*engeneering change order*), описанные в литературе, можно разделить на несколько базовых типов.

Методы на основе использования моделей ошибок [1—3]. Обычно рассматриваются только простые ошибки типа неверных соединений проводников или замены логических элементов. С помощью этих методов ищут ошибки только конкретных типов и применяют заранее известные решения для их исправления. В рамках такого узкого круга случаев задача решается быстро и эффективно. Однако этих моделей ошибок обычно недостаточно для представления всех функциональных ошибок в реальных проектах [4].

Методы, основанные на анализе структуры схем [5—7]. Эффективность данной группы методов базируется на том допущении, что на практике ошибки на уровне регистровых передач обычно сравнительно небольшие и локализованы в одной небольшой области схемы. Для исправления схемы в этих методах используется сравнение структуры схемы с эталоном. Структурный анализ схем работает только в том случае, если схемы имеют большое число общих частей.

Методы, основанные на синтезе [8—12]. Данные методы не опираются на информацию о структуре эталонной схемы, вместо этого используются моделирование и методы функциональной верификации. Чаще всего методы этой группы работают в два этапа. На первом этапе находят те узлы, исправление которых способно исправить проект. На втором этапе запускается ресинтез для этих узлов. В то время как основным недостатком данной группы методов является слабая масштабируемость и зачастую ориентация на исправление только единичного сигнала, их преимуществом является полная автоматизация.

В данной работе рассматривается задача *ECO*, основанная на методе учета ресурсов при построении корректирующей схемы (патча), предложенная в работе [13]. В этой работе был предложен метод, основанный на учете некоторой физической информации о запасных

ячейках (*spare cells*) и расчете стоимости соединений после технологического маппинга. Таким образом, исправление ошибок в проекте предлагается проводить с учетом некоторой функции стоимости соединений. Это позволяет впервые автоматизировать процесс нахождения исправлений не только с учетом наименьшего числа логических элементов, но также с учетом полной длины межсоединений.

В маршруте промышленного проектирования функциональный инструмент *ECO*, например *Cadence Encounter Conformal ECO Designer* [14], широко применялся в течение многих лет. Базовой характеристикой корректирующей схемы в рамках данных средств является площадь патча. Несмотря на то что размер патча является важной метрикой его качества, для практического решения задачи *ECO* необходимо рассматривать и другие физические аспекты, включая обеспечение заданных временных и мощностных параметров.

В ряде теоретических исследований [4, 6, 8, 11, 12, 15—17] были предложены различные типы алгоритмов для генерации патчей. Основное внимание в работах уделялось минимизации размера патча. Однако сгенерированные патчи могут оказаться непригодными для решения промышленных задач ввиду слишком больших накладных расходов на их физическую реализацию.

1. Постановка задачи

В работе рассматривается второй этап маршрута *ECO* — этап исправления ошибок. На первом этапе — этапе обнаружения ошибок, части проекта, ответственные за ошибку, уже были идентифицированы. Это означает, что в логической схеме существует несколько висячих узлов, которые называются таргетами (*targets*). Необходимо с помощью некоторого набора узлов-кандидатов сформировать подсхемы для этих таргетов таким образом, чтобы основная схема стала эквивалентна эталонной схеме. При этом нужно учитывать обобщенный весовой параметр, который имеет каждый узел-кандидат в схеме, что является своего рода метрикой накладных расходов для его использования в патче.

Формализуем задачу, введя некоторые обозначения. Даны две схемы: F — рассматриваемая схема и G — эталонная схема; набор таргетов t_1, t_2, ..., t_n; набор весов w_{g_i}, для каждого узла-кандидата g_i, $i = 1...k$ из F. Задача: сгенерировать функции патча с минимальной стоимостью в определенном наборе таргетных точек в F таким образом, чтобы исправленная схема F' и схема G стали эквивалентны (рис. 1). Стоимость патча вычисляется суммированием весов используемых узлов-кандидатов.

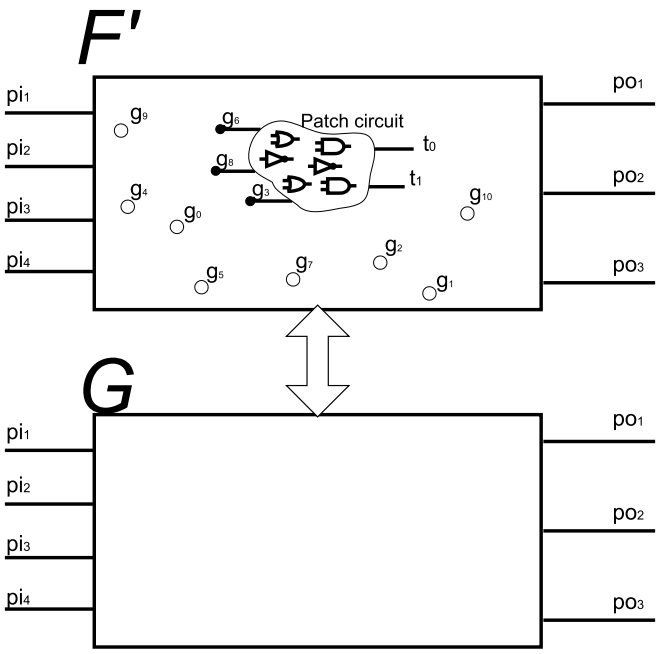

Рис. 1. Постановка задачи автоматизированной функциональной коррекции

Базисом для разрабатываемых методов служит формальный механизм проверки эквивалентности, связанный с представлением схемы в виде направленного ациклического графа, состоящего из инверсий и конъюнкций (And-Inverter Graph (AIG)) с последующим решением задачи выполнимости булевых формул (SAT). Данный механизм реализован в САПР с открытым исходным кодом — *ABC*, разработанном в Калифорнийском университете в Беркли (США) [18]. Все финальные и промежуточные проверки на эквивалентность реализованы с помощью встроенной функции *cec*. Также, не ограничивая общности рассуждений, будем считать, что используются только двух- и одновходовые вентили, поддерживаемые синтаксисом языка *Verilog*.

2. Структурный анализ схемы

Разбиение на подзадачи. Первый этап работы алгоритма заключается в структурном анализе схемы для сокращения вычислительной

нагрузки на последующих этапах. В частности, в рамках такого анализа делается попытка разбиения задачи на подзадачи меньшего размера в случае наличия нескольких таргетов.

Структурный анализ начинается с построения выходных конусов распространения сигнала для выходов каждого таргета и анализа первичных выходов схемы, которые попадают под влияние этих таргетов. Те выходы схемы, которые не попали под влияние таргетов, должны быть заведомо эквивалентны соответствующим выходам схемы *G*. Этот факт проверяется на начальном этапе, и в случае положительного исхода эти выходы исключаются из дальнейшего рассмотрения. В противном случае делается вывод о том, что никакой патч не сможет исправить эту ошибочную схему, что сигнализирует об ошибке на первом этапе маршрута *ECO*.

Далее, на основе данных о влиянии таргетов на выходы схемы строится целевая карта чувствительности (*target sensitivity map*, *TSM*). Эта структура формируется посредством группировки таргетов относительно влияния их на конкретные выходы и представляет собой функцию, определенную на конкретных наборах таргетов и отображающую их в наборы выходов: TSM: $T_i \rightarrow PO_j$, где $T_i = \{t_{i_1}, t_{i_2}, \ldots, t_{i_k}\}$ — некоторый набор таргетов; $PO_j = \{po_{j_1}, po_{j_2}, \ldots, po_{j_l}\}$ — некоторый набор выходов, po_j — соответствующий выход схемы. Такое отображение означает, что каждый выход из набора PO_j зависит от каждого таргета из набора T_i. Аргументы функции *TSM* уникальны: отдельные таргеты в них могут повторяться, в то время как po_j в значениях функции повторяться не могут. Ниже представлен пример *TSM* для одной реальной задачи, описанный в структуре данных языка *Python*. Третья строчка, к примеру, в этом словаре говорит о том, что на выход ‘g82’ влияют таргеты ‘t_0’, ‘t_2’, ‘t_7’.

```
{(‘t_0’, ‘t_1’): [‘g65’],
(‘t_0’, ‘’t_1’, ‘t_2’, ‘t_3’): [‘g66’],
(‘t_0’, ‘t_2’, ‘t_7’): [‘g82’],
(‘t_1’, ‘t_2’): [‘g67’],
(‘t_1’, ‘t_4’): [‘g83’],
(‘t_5’, ‘t_6’, ‘t_7’): [‘g89’],
(‘t_7’): [‘g85’, ‘g87’, ‘g88’, ‘g86’, ‘g84’]}
```

Используя *TSM*, можно легко определить замкнутые непересекающиеся классы таргетов, каждый из которых влияет только на некоторое подмножество выходов схемы, в то время как другие таргеты не влияют на них. Если существует больше одного такого класса, то задача разбивается на подзадачи, которые решаются параллельно и независимо, в том смысле что финальную проверку на эквивалентность для каждой такой подзадачи можно проводить лишь для соответствующих этому классу выходов.

Стратегия выбора таргетов. На основе *TSM* также реализована стратегия последовательного выбора таргетов для формирования патча. Базовая процедура нахождения патча, которая будет описана в последующих разделах, существенно ограничена обработкой только одного таргета. Поэтому чрезвычайно важно выбрать правильную последовательность обработки таргетов для наиболее эффективного нахождения патчей. Суть стратегии заключается в том, чтобы нахождение патча для каждого следующего таргета позволяло проверить на эквивалентность как можно больше выходов, либо максимально приблизить такую проверку. Алгоритм выбора следующего таргета для оценки представлен ниже.

1. Посчитать новый *TSM*.

2. Сформировать набор $S = \{T_{k_1}, T_{k_2}, \ldots, T_{k_n}\}$, в котором каждый набор T_{k_i} — имеет минимальное число таргетов.

3. Выбрать набор $T_{\max}$ из S, имеющий максимальное число зависимых выходов: $|TSM(T_{\max})| = \max(|TSM(T_{k_i})|)$; $i = 1...n$.

4. Выбрать случайный таргет из $T_{\max}$.

После оценки очередного таргета и исключения зависящих только от него выходов считается новый *TSM* относительно оставшихся таргетов и выходов. Для рассмотренного примера последовательность выбора таргетов выглядит следующим образом:

‘t_7’ -> ‘t_0’ -> ‘t_1’ -> ‘t_2’ -> ‘t_3’ -> ‘t_4’ -> ‘t_5’ -> ‘t_6’.

Откат таргетов. Стратегия выбора таргетов служит задаче минимизации ситуаций, когда после определенной итерации основного алгоритма оказывается, что патч работает некорректно. Чтобы не откладывать проверку на эквивалентность на самый конец, после каждого найденного таргета проводится формальная проверка тех выходов, которые уже определены всеми таргетами. Тем не менее может случиться, что после какого-то этапа формальная проверка на эквивалентность дает

отрицательный результат. Это может возникнуть за счет того, что базовая процедура поиска патча для отдельного таргета, как правило, опирается только на некоторое подмножество входных стимулов, и в некоторых случаях может генерировать некорректный патч. В этом случае необходимо откатить неправильно найденный таргет. Поиск такого таргета также происходит по *TSM*. Рассмотрим базовую стратегию такого поиска на том же примере. Пусть алгоритм остановился на этапе: ‘t_7’ -> ‘t_0’ -> ‘t_1’ -> ‘t_2’, после чего средство проверки формальной эквивалентности выявило ошибку на первичном выходе ‘g82’. Этот выход зависит от трех таргетов: {‘t_0’, ‘t_2’, ‘t_7’}. На выходе средства проверки эквивалентности мы получаем некоторый стимул, на котором происходит нарушение эквивалентности (стимул-контрпример). Далее, патчи для каждого таргета поочередно исключаются из "списка подозреваемых" и поиск запускается заново, но уже включая стимул-контрпример. Особенность базовой процедуры поиска патча состоит в том, что, имея стимул-конрпример, уже на первых этапах ее работы становится ясно — возможно ли построить корректный патч. Отрицательный ответ подразумевает, что исключенный таргет был на самом деле корректным. Противоположный случай говорит о том, что исключенный таргет и был источником ошибки. Такой патч исключается из списка найденных, а таргет откатывается.

Определение значащих входов. С помощью анализа структуры схем также можно определить список значащих входов схемы. Это позволит более эффективно использовать симуляционное моделирование, обрабатывая те стимулы, на которых с большей вероятностью могут возникнуть конфликты. Получение значащих входов становится возможным за счет наличия эталонной схемы *G*. Для нахождения таких входов достаточно построить входные конусы от всех зависимых выходов в эталонной схеме. Таким образом, мы исключаем из рассмотрения те входы, которые не могут повлиять на эквивалентность тестовой и эталонной схем. Это также снижает размерность задачи и для некоторых сравнительно больших схем возникает возможность использовать полный перебор только по значащим входам, если их число невелико. Это гарантирует нахождение корректного патча.

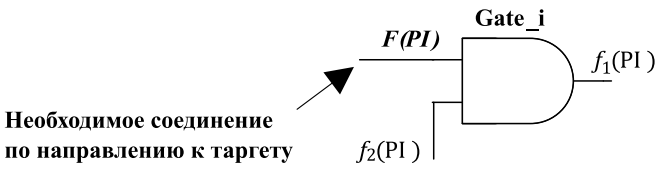

Рис. 2. Поиск функции для входа вентиля при известных функциях на другом входе и выходе

a	*b*	**OR(*a*,*b*)**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

a	OR(*a*,*b*)	***b***
0	0	**0**
0	1	**1**
1	0	**-**
1	1	**x**

Рис. 3. Нахождение обратной функции для вентиля OR2

3. Метод формального поиска патча

Еще один способ гарантированного нахождения корректного патча заключается в использовании формальных методов обратного распространения функции. Однако использование данного метода ограничено определенной структурой схемы. Анализ соответствующих ограничений для формального метода можно также провести на первом этапе работы нашего маршрута *ECO*.

Задача формального метода — найти точную функцию для таргета относительно первичных входов схемы *F*. Базисом для данного метода является процедура обратного распространения функции вентиля. Она формулируется следующим образом: пусть нам известны функции от входов схемы на одном из входов вентиля и функция на его выходе; необходимо найти функцию на другом его входе. Рассмотрим пример. Пусть дан вентиль *OR* и известны функции $f_1(PI)$ и $f_2(PI)$ на выходе и на одном из входов вентиля соответственно, где *PI* — некоторое подмножество основных входов схемы. Задача — найти функцию на втором его входе (рис. 2).

Для этого необходимо вначале определить, какая функция для вентиля *OR* будет обратной. Анализируя таблицу истинности, можно легко определить, что для этой роли подойдет как функция *OR*, так и функция *XOR*, функция второго операнда и инверсия прямой импликации (рис. 3).

Таким образом, функция для первого входа вентиля может быть найдена по формуле $F(PI)n = f_1(PI) + f_2(PI)$, или просто как

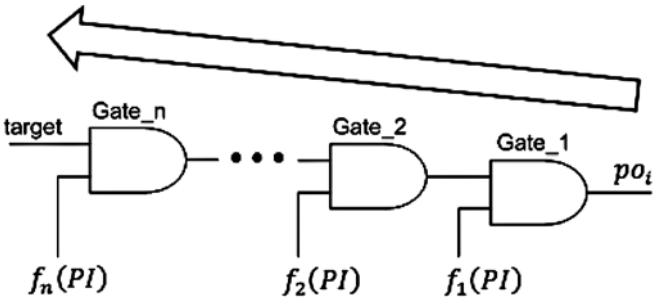

Рис. 4. Принцип обратного распространения для задачи формального поиска патча

$F(PI) = f_2(PI)$. Таким образом можно продвигаться в обратную сторону от выходов к входам схемы. Отправной точкой служит известная функция для основного выхода (*primary output*) эталонной схемы *G*. Конечной точкой служит вентиль схемы, к которому непосредственно подсоединен таргет.

Возможности данного метода существенно ограничены структурой схемы, показанной на рис. 4. В подсхеме от таргета до выхода не должно быть разветвлений, а все входные сигналы *fi* должны зависеть только от входов схемы, и не должны зависеть от других таргетов. Несмотря на довольно строгие структурные ограничения, подобные ситуации встречаются довольно часто — примерно в 20 % случаев.

Достоинствами метода можно считать скорость работы (поскольку требуется только простой структурный анализ и однократная процедура обратного распространения), а также гарантию корректности патча. К недостаткам, помимо структурных ограничений, можно отнести довольно большой размер получаемых патчей. Тем не менее второй недостаток может быть нивелирован дополнительным вторым этапом, на котором проводится минимизация размера патча.

4. Метод генерации патчей на базе моделирования схемы

Ядром разработанного маршрута *ECO* является метод генерации патчей основанный на моделировании. Как следует из названия, этот метод принципиально базируется на симуляции схемы. Вследствие бит-параллельного моделирования схемы и эталона получаем набор сигнатур для входов, выходов и узлов-кандидатов схемы *F*. Сигнатура *S* — это *n*-битный вектор на входах, выходах или внутренних узлах схемы, определяющий значения на этом узле, полученные в результате моделирования схемы, где *n* — это число моделирований схемы. Обращение к *n*-му биту сигнатуры, как и к любому битовому вектору, будем обозначать *S*[*k*]. Поскольку моделирование на вычислительных машинах удобно выполнять в бит-параллельном формате, то каждая сигнатура получается параллельно и параметр *n* характеризует длину входной последовательности при симуляции схемы. После первоначального моделирования получаем некоторый набор сигнатур. Пример представлен в табл. 1.

Следует отметить, что из списка узлов-кандидатов на первом этапе удаляются те узлы, которые зависят от таргетов. Это позволяет обрабатывать таргеты отдельно и независимо, а также уберегает от случайной генерации цепей обратной связи.

Следующий этап связан с построением некоторого массива, определяющего какие комбинации значений на таргетах должны быть для каждой симуляции, для того чтобы первичные выходы схем *F* и *G* совпадали. Назовем такой массив целевым (*Target array* (*Ta*)). Для получения этого массива требуется провести 2^k бит-параллельных симуляций схемы *F*, где *k* — число таргетов, построчно сравнивая получившиеся значения выходов *F* со значениями выходов *G* (табл. 2).

Этот массив дает исчерпывающую информацию о том, какие сигналы должны формироваться на выходах патча для того чтобы схема работала корректно на заданных стимулах. Дальнейшая работа алгоритма связана с по-

Таблица 1

Таблица сигнатур, полученная в результате бит-параллельного моделирования

	№	Схема *F*						Схема *G*		
		S_{pi_1}	...	S_{pi_I}	S_{g_1}	...	S_{g_m}	S_{po_1}	...	S_{po_k}
Число симуляций	0	0	...	0	0	...	1	0	...	1
	1	0	...	0	1	...	0	0	...	1
	2	0	...	0	0	...	0	0	...	0
	3	0	...	0	1	...	0	0	...	1
	...	...								
	$n-1$	1	...	1	0	...	0	0	...	1
		Сигнатуры на первичных входах			Сигнатуры на вентилях схемы *F*			Сигнатуры на первичных выходах схемы *G*		

следовательной обработкой таргетов в порядке, описанном в предыдущих разделах.

Имея *Ta* и нужный таргет для обработки, получаем целевой вектор (*Target vector* (*Tv*)), битовый вектор длиной *n*, определяющий, какой именно сигнал должен быть подан на этот таргет. Этот вектор похож на сигнатуру, за тем исключением, что кроме 0 и 1 в его разрядах могут быть безразличные состояния (*X*). При выделении *Tv* из *Ta* используется жадный алгоритм, иными словами максимизируется число *X* в *Tv*. Дальше задача сводится к нахождению базиса из узлов-кандидатов для формирования патча.

Эта задача решается с помощью метода поиска базиса на основе конфликтов. Метод подробно описан в нашей предыдущей работе [19].

5. Результаты экспериментов

В экспериментальной части мы использовали тестовые схемы, которые представляют собой реальные практические случаи с различными сценариями *ECO*. Наборы эталонных тестов были созданы из схем ISCAS, ITC99 в IWLS 2005, OpenCore, LGSynth'93 и некоторых комбинационных участков сложных промышленных проектов.

Были рассмотрены различные сценарии *ECO* с различным числом точек, разным расстоянием между точками до первичных входов/выходов, а также разными размерами схем. Стратегии распределения весов также различались. Эти примеры были заимствованы из ежегодного конкурса ICCAD Contest, проведенного в 2017 г. Все контрольные примеры доступны в открытом доступе на сайте [20]. Результаты экспериментов представлены в табл. 3.

Таблица 2

Построение целевого массива в рамках задачи автоматизированной функциональной коррекции схем

Sim. №	Схема *F*							Схема *G*			*Ta*
	$(t_0, t_1, ..., t_k)$ (0, 0, ..., 0)			...	$(t_0, t_1, ..., t_k)$ (1, 1, ..., 1)						
	S_{po_1}	...	S_{po_k}	...	S_{po_1}	...	S_{po_k}	S_{po_1}	...	S_{po_k}	
0	0	...	0	...	0	...	1	0	...	1	{(0, 0, ...1), (0, 1, ...1), ...}
1	0	...	0	...	1	...	0	0	...	1	{(1, 1, ...1), (1, 0, ...1), ...}
2	0	...	0	...	0	...	0	0	...	0	{(1, 0, ...0), (1, 0, ...0), ...}
3	0	...	0	...	1	...	0	0	...	1	{(0, 0, ...0), (1, 0, ...1), ...}
...	...										...
$n-1$	1	...	1	...	0	...	0	0	...	1	{(0, 0, ...0), (1, 0, ...1), ...}

Таблица 3

Исследование эффективности маршрута автоматизированной ресурсоориентированной функциональной коррекции схем

Схема	Число таргетов	Число входов	Число выходов	Число элементов	Число элементов в эталоне	Среднее время работы, с	Среднее число элементов в патче
1	1	3	2	5	6	0	3
2	1	157	64	1122	1219	30	9
3	1	411	128	2074	1929	14	8
4	1	11	6	74	77	5	4
5	2	450	282	24 355	21 056	302	123
6	2	99	128	13 832	11 818	451	9
7	1	207	24	2944	1721	264	10
8	1	179	64	2512	3337	41	18
9	4	256	245	5845	4657	104	46
10	2	32	129	1673	2052	322	251
11	8	48	50	2065	2177	400	850
12	1	46	27	13 803	821	76	3
13	1	25	39	377	435	30	47
14	12	17	15	1969	1006	140	274

Заключение

В работе представлен маршрут автоматизированной ресурсо-ориентированной функциональной коррекции схем. Предложены различные структурные подходы, которые позволяют уменьшить вычислительную нагрузку на симуляционный метод поиска базиса.

На языке Python 3.4 был реализован программный модуль для автоматизированной ресурсоориентированной функциональной коррекции схем. Экспериментальные результаты показывают высокую эффективность метода даже на очень сложных задачах с большим числом таргетов, элементов и входов.

Исходный код вместе с тестовыми примерами, а также подробное описание задачи опубликованы в открытом доступе на сайте [20]. Это открывает путь для разработчиков САПР в области *ECO* для сравнения их алгоритмов и использования разработок для собственного программного обеспечения.

Список литературы

1. **Veneris A. G., Hajj I. N.** A fast algorithm for locating and correcting simple design errors in VLSI digital circuits // Proc. of Great Lake Symposium on VLSI Design. 1997. P. 45—50.
2. **Abadir M., Ferguson J., Kirkland T.** Logic design verification via test generation // IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 1988. Vol. 7, N. 1. P. 138—148.
3. **Chung P.-Y., Hajj I. N.** Accord: Automatic catching and correction of logic design errors in combinational circuits // Proc. of Int. ITC, Sep. 1992. P. 742—751.
4. **Shao-Lun Huang, Wei-Hsun Lin, Po-Kai Huang, Chung-Yang Huang.** Match and Replace: A Functional ECO Engine for Multierror Circuit Rectification // Computer-Aided Design of Integrated Circuits and Systems IEEE Transactions. 2013. Vol. 32. P. 467—478.
5. **Brand D., Drumm A. D., Kundu S., Narain P.** Incremental synthesis // International Conference on Computer Aided Design. 1994. P. 14—18.
6. **Krishnaswamy S., Ren H., Modi N., Puri R.** DeltaSyn: An efficient logic difference optimizer for ECO synthesis // International Conference on Computer Aided Design. 2009. P. 789—796.
7. **Huang S.-Y., Chen K.-C., Cheng K.-T.** AutoFix: A hybrid tool for automatic logic rectification // Transactions on Computer-Aided Design of Integrated Circuits and Systems. 1999. Vol. 18, N. 9. P. 1376—1384.
8. **Lin C.-C., Chen K.-C., Marek-Sadowska M.** Logic synthesis for engineering change // IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 1999. Vol. 18, N. 3. P. 282—292.
9. **Yang Y.-S., Sinha S., Veneris A., Brayton R.** Automating logic rectification by approximate SPFDs // Proc. ASP-DAC, Jan. 2007. P. 402—407.
10. **Ling A., Brown S., Safarpour S., Zhu J.** Toward automated ECOs in FPGAs // IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., Jan. 2011. Vol. 30, N. 1. P. 18—30.
11. **Wu B.-H., Yang C.-J., Huang C.-Y., Jiang J.-H.** A robust functional ECO engine by SAT proof minimization and interpolation techniques // Proc. IEEE/ACM Int. Conf. ICCAD. Nov. 2010. P. 729—734.
12. **Chang K. H., Markov I., Bertacco V.** Fixing design errors with counterexamples and resynthesis // IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., Jan. 2008. Vol. 27, N. 1. P. 184—188.
13. **Cheng A.-C., Jiang I. H.-R., Jou J.-Y.** Resource-aware functional ECO patch generation // Design, Automation & Test in Europe Conference & Exhibition (DATE). 2016. P. 1036—1041.
14. **Cadence** Encounter Conformal ECO Designer. URL: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/functional-eco/conformal-eco-designer.html
15. **Tang K.-F., Wu C.-A., Huang P.-K., Huang C.-Y.** Interpolation-based incremental ECO synthesis for multi-error logic rectification // Design Automation Conference (DAC), 2011. P. 146—151.
16. **Tang K.-F., Huang P.-K., Chou C.-N., Huang C.-Y.** Multi-patch generation for multi-error logic rectification by interpolation with cofactor reduction // Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012. P. 1567—1572.
17. **Lin C.-H., Huang Y.-C., Chang S.-C., Jone W.-B.** Design and design automation of rectification logic for engineering change // Asia and South Pacific Design Automation Conference (ASP-DAC), 2005. P. 1006—1009.
18. **Berkeley** Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. URL: http: // www.eecs.berkeley.edu/~alanmi/abc/.
19. **Stempkovskiy A. L., Telpukhov D. V., Soloviev R. A.** Fast and accurate resource-aware functional ECO patch generation tool // Proc. of Moscow Workshop on Electronic and Networking Technologies (MWENT), 2018. P. 1—5.
20. **Исходный** код средств автоматизации ресурсоориентированной функциональной коррекции логических схем. URL: https://github.com/IDMIPPM/Functional_ECO

A. L. Stempkovskiy, D. Sci., Scientific Director,
D. V. Telpukhov, Ph. D., Head of Department,
R. A. Solovyev, Ph. D., Chief Researcher, **Y. V. Bitkov,** Researcher,
Institute for Design Problems in Microelectronics (IPPM RAS), Moscow, 124365, Russian Federation,
nofrost@inbox.ru,

Development of Methods for Automating Resource-aware Functional ECO Patch Generation

In the modern design process of VLSIs, situations often arise when an incorrect operation of the circuit is detected only at the stage of final verification. Worse, when errors occur even after the final stage of design or production of microcircuits. Correction of errors at such late stages requires enormous labor and financial costs. In order to minimize these costs, methods have been developed for making the changes in the advanced stages of design process (engeneering change order, ECO) for making corrections to the final design, instead of performing a complete redesign. At present, such corrections are mostly made manually, despite the fact that the main CAD vendors already have some automated tools for ECO.

In this paper, we propose methods for automating the functional correction of circuits based on conflicts, as well as methods for analyzing the structure of the circuit. Implemented software demonstrates high efficiency on a number of real-world tasks.

Keywords: *engeneering change order, ECO, functional correction, conflicts, ICCAD Contest.*

DOI: 10.17587/it.24.755-762

References

1. **Veneris A. G., Hajj I. N.** A fast algorithm for locating and correcting simple design errors in VLSI digital circuits, *Proc. of Great Lake Symposium on VLSI Design*, 1997, pp. 45—50.
2. **Abadir M., Ferguson J., Kirkland T.** Logic design verification via test generation, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 1988, vol. 7, no. 1, pp. 138—148.
3. **Chung P.-Y., Hajj I. N.** Accord: Automatic catching and correction of logic design errors in combinational circuits, *Proc. Int. ITC, Sep. 1992*, pp. 742—751.
4. **Huang S.-L., Lin W.-H., Huang P.-K., Huang C.-Y.** Match and Replace: A Functional ECO Engine for Multierror Circuit Rectification, *Computer-Aided Design of Integrated Circuits and Systems IEEE Transactions*, 2013, vol. 32, pp. 467—478.
5. **Brand D., Drumm A. D., Kundu S., Narain P.** Incremental synthesis, *International Conference on Computer Aided Design*, 1994, pp. 14—18.
6. **Krishnaswamy S., Ren H., Modi N., Puri R.** DeltaSyn: An efficient logic difference optimizer for ECO synthesis, *International Conference on Computer Aided Design*, 2009, pp. 789—796.
7. **Huang S.-Y., Chen K.-C., Cheng K.-T.** AutoFix: A hybrid tool for automatic logic rectification, *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1999, vol. 18, no. 9, pp. 1376—1384.
8. **Lin C.-C., Chen K.-C., Marek-Sadowska M.** Logic synthesis for engineering change, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 1999, vol. 18, no. 3, pp. 282—292.
9. **Yang Y.-S., Sinha S., Veneris A., Brayton R.** Automating logic rectification by approximate SPFDs, *Proc. ASP-DAC, Jan. 2007*, pp. 402—407.
10. **Ling A., Brown S., Safarpour S., Zhu J.** Toward automated ECOs in FPGAs, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, Jan. 2011, vol. 30, no. 1, pp. 18—30.
11. **Wu B.-H., Yang C.-J., Huang C.-Y., Jiang J.-H.** A robust functional ECO engine by SAT proof minimization and interpolation techniques, *Proc. IEEE/ACM Int. Conf. ICCAD, Nov. 2010,* pp. 729—734.
12. **Chang K. H., Markov I., Bertacco V.** Fixing design errors with counterexamples and resynthesis, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, Jan. 2008, vol. 27, no. 1, pp. 184—188.
13. **Cheng A.-C., Jiang I. H.-R., Jou J.-Y.** Resource-aware functional ECO patch generation, *Design, Automation & Test in Europe Conference & Exhibition (DATE),* 2016, pp. 1036—1041.
14. **Cadence** Encounter Conformal ECO Designer, available at: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/ digital-design-and-signoff/functional-eco/conformal-eco-designer.html
15. **Tang K.-F., Wu C.-A., Huang P.-K., Huang C.-Y.** Interpolation-based incremental ECO synthesis for multi-error logic rectification, *Design Automation Conference (DAC),* 2011, pp. 146—151.
16. **Tang K.-F., Huang P.-K., Chou C.-N., Huang C.-Y.** Multi-patch generation for multi-error logic rectification by interpolation with cofactor reduction, *Design, Automation & Test in Europe Conference & Exhibition (DATE),* 2012, pp. 1567—1572.
17. **Lin C.-H., Huang Y.-C., Chang S.-C., Jone W.-B.** Design and design automation of rectification logic for engineering change, *Asia and South Pacific Design Automation Conference (ASP-DAC), 2005,* pp. 1006—1009.
18. **Berkeley** Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, available at: http://www.eecs.berkeley.edu/~alanmi/abc/.
19. **Stempkovskiy A. L., Telpukhov D. V., Soloviev R. A.** Fast and accurate resource-aware functional ECO patch generation tool, *Proc. Moscow Workshop on Electronic and Networking Technologies (MWENT),* 2018, pp. 1—5.
20. **IDMIPPM** / Functional-ECO, available at: https://github.com/IDMIPPM/Functional_ECO