УДК 004.023 DOI 10.31114/2078-7707-2018-1-23-29

Алгоритмы решения задачи функциональной коррекции схем из функциональных элементов

М.А. Автайкина, М.С. Шуплецов

Московский государственный университет им. М.В. Ломоносова – факультет вычислительной математики и кибернетики, shupletsov@cs.msu.ru

***Аннотация* — Согласно статистике [1] более 60% ошибок проектирования интегральных схем (ИС) связано с функциональным несоответствием схем их первоначальным спецификациям. При обнаружении таких ошибок на финальных стадиях проектирования ИС их исправление влечет за собой большие затраты, связанные с восстановлением функциональности и перепроектированием схемы. Исправление схемы вручную является очень трудоемкой задачей, а повторный запуск маршрута проектирования требует существенного времени. Функциональная коррекция представляет собой автоматизированный подход к решению данной проблемы. Она заключается в формировании небольшой подсхемы-заплатки, внедрение которой в уже синтезированную схему позволяет исправить все обнаруженные логические несоответствия. При этом не требуется заново повторять все этапы логического синтеза, что существенно сокращает время проектирования схемы. В данной работе представлен алгоритм функциональной коррекции, в основе которого лежит идея разбиения схем на соответствующие друг другу подсхемы. Ввиду того, что описания современных ИС обладают большим размером и сложностью, анализ указанных описаний как единого целого требует значительных вычислительных ресурсов. Разбиение исходных схем на подсхемы меньшего размера позволяет существенно снизить сложность рассматриваемой задачи. Эксперименты показали, что представленный алгоритм успешно справляется с локализацией и исправлением неисправностей в схеме.**

***Ключевые слова* — схемы из функциональных элементов, логический синтез, разбиение схем, функциональная коррекция, поиск функциональных соответствий.**

I. Введение

Для современных маршрутов проектирования интегральных схем, когда время от начала проектирования устройства до его выхода на рынок является ключевым фактором, поздние модификации в структуре схемы, связанные с изменением спецификации или обнаружением логических ошибок в схеме, реализуются при помощи формирования небольшой подсхемы-заплатки (англ. Engineering Change Order, ECO) [2]. Внедрение такой подсхемы в уже синтезированную схему позволяет исправить все обнаруженные логические несоответствия. При этом не требуется заново повторять все этапы логического и физического синтеза ИС, что существенно сокращает время проектирования схемы. Необходимо учитывать, что для больших схем обнаружение небольших неэквивалентных фрагментов схем является очень трудоемкой задачей. Возможность локализовать и изолировать такие фрагменты за счет разбиения исходных схем позволяет не только снизить сложность решаемой задачи, но и улучшить качество получаемых подсхем-заплаток.

Таким образом, решение задачи функциональной коррекции может быть упрощено за счет разбиения схем на соответствующие друг другу подсхемы меньшего размера. Основная цель при этом заключается в выделении эквивалентных подсхем, которые покрывают максимальную часть исходных схем и тем самым позволяют минимизировать размер оставшейся неэквивалентной части.

Изучению рассматриваемой задачи посвящен целый ряд научных работ. Так, например, в [3,4] представлен метод поиска минимальной заплатки на основе структурного соответствия схем. В данном методе узлы схем просматриваются в топологическом порядке. При обнаружении несоответствия типов функциональных элементов (ФЭ), приписанных соответствующим узлам, ФЭ узла неисправной схемы заменяется на ФЭ соответствующего узла правильной схемы. Несмотря на высокую скорость работы данный алгоритм очень редко применяется в реальных задачах, так как на практике схемы, как правило, имеют различную структуру.

Принципиально иной подход описан в [2]. Авторы работы используют двоичные решающие диаграммы (англ. Binary Decision Diagram, BDD) [5] для нахождения функционально эквивалентных узлов схем. Для каждого выхода неисправной схемы на найденных узлах строится подсхема-заплатка с тем, чтобы на соответствующих выходах схем реализовывались равные функции алгебры логики (ФАЛ).

Нередко задачу функциональной коррекции сводят к задаче обнаружения и устранения неисправностей [6, 7]. Существующий набор алгоритмов для ее решения подразумевает некоторую заданную модель неисправностей (например, неверно заданный тип ФЭ в узле, неправильное соединение узлов и т.п.). Это сужает область поиска и позволяет найти решение за разумное время. Однако основной недостаток

подобных алгоритмов заключается в том, что довольно часто схемы слишком сильно отличаются друг от друга, и эта разность не может быть описана заранее заданной моделью неисправностей.

В свою очередь, в [8, 9] представлены системы для построения подсхемы-заплатки наименьшего размера с использованием интерполяции и специальных программ для решения задачи выполнимости конъюнктивных нормальных форм (КНФ), так называемых SAT-решателей (например, MiniSat [10]).

Наконец, в работе [11] представлен двухэтапный подход к решению рассматриваемой задачи. Минимизация заплатки при этом сводится к максимизации эквивалентной части схем. На первом этапе ищутся структурные и функциональные соответствия в верхней части схем (обход при этом осуществляется от входов к выходам). На втором этапе соответствия ищутся в обратном направлении (от выходов к входам) с использованием принципов динамического программирования и алгоритмов поиска булевых соответствий (англ. Boolean matching) [12]. Соответствия, найденные на первом этапе, образуют верхнюю, а на втором этапе - нижнюю «границы» искомой заплатки. Таким образом, в качестве заплатки используется подсхема правильной схемы, заключенная между найденными границами. Данный подход особенно эффективен для схем, различия между которыми незначительны и располагаются близко к выходам схем.

Стоит отметить, что задаче разбиения схем на эквивалентные подсхемы с целью упрощения решения задач проверки эквивалентности и функциональной коррекции сверхбольших ИС было посвящено международное соревнование по разработке алгоритмов автоматизации проектирования ИС «CAD Contest at ICCAD 2015» [13]. По результатам этого соревнования в работах [14, 15, 16] был представлен ряд эффективных алгоритмов разбиения пары ИС на подсхемы, которые позволяли находить множества эквивалентных подсхем, тем самым локализуя оставшиеся неэквивалентные подсхемы. Алгоритм, предложенный авторами в работе [16] и использующий идею динамического программирования, показал наилучшие результаты в локализации неэквивалентной части схем на тестовой выборке, предоставленной в рамках соревнования [13].

В данной работе представлен алгоритм решения задачи функциональной коррекции схем, в основе которого лежит идея разбиения схем большого размера на соответствующие друг другу эквивалентные подсхемы. Рассматриваемый алгоритм позволяет эффективно обнаруживать неэквивалентные фрагменты и заменять их подсхемами, восстанавливающими эквивалентность исходных схем. Данный подход лишен недостатков [3, 4], так как оперирует функциональной, а не структурной схожестью схем. Кроме того, в отличие от [11], разработанный алгоритм способен находить и исправлять неисправности вне зависимости от их количества, сложности или расположения в схеме.

Статья имеет следующую структуру. В части II приводятся формальное описание задачи функциональной коррекции схем и набор вспомогательных определений. Последующие части описывают алгоритмы и общую структуру системы, разработанной для решения рассматриваемой задачи. В заключительной части приводятся результаты тестирования предложенного решения задачи.

II. Основные понятия и постановка задачи

Даны две схемы из функциональных элементов (СФЭ) Σ_1 и Σ_2, реализующие системы ФАЛ $F_1 = (f_1^1, \dots, f_k^1)$ и $F_2 = (f_1^2, \dots, f_k^2)$, соответственно. При этом каждому входу Σ_1 взаимно однозначно сопоставлен вход Σ_2, а каждому выходу Σ_1 взаимно однозначно сопоставлен выход Σ_2.

Будем говорить, что две вершины эквивалентны, если в них реализуются равные ФАЛ. Тогда СФЭ Σ_1 и Σ_2 *эквивалентны* $(\Sigma_1 \sim \Sigma_2)$, если эквивалентны соответствующие друг другу выходы схем. Тогда под *задачей проверки эквивалентности* СФЭ будем понимать проверку того факта, что заданные схемы являются эквивалентными. А под *задачей поиска функциональных соответствий* - нахождение такой перестановки входов схем, если она существует, что результирующие схемы будут эквивалентны.

Пусть задана СФЭ Σ с множеством вершин V_Σ и множеством ребер E_Σ. СФЭ Σ' будем называть *подсхемой* схемы Σ $(\Sigma' \subseteq \Sigma)$, если $V_{\Sigma'} \subseteq V_\Sigma$ и $E_{\Sigma'} \subseteq E_\Sigma$. Будем говорить, что подсхемы Σ' и Σ'' *СФЭ* Σ не пересекаются $(\Sigma' \cap \Sigma'' = \emptyset)$, если $V_{\Sigma'} \cap V_{\Sigma''} = \emptyset$.

В данной работе рассматриваются подсхемы специального вида – конусы. Для их описания введем ряд вспомогательных определений. Пусть задана некоторая вершина $v, v \in V_\Sigma$. Будем говорить, что вершина v, $v \in V_\Sigma$, достижима из вершины u, $u \in V_\Sigma$, если в СФЭ Σ существует ориентированный (u, v)-путь. Для вершины v, $v \in V_\Sigma$, и заданного подмножества вершин $\bar{V}$, $\bar{V} \subseteq V_\Sigma$, из которых достижима вершина v, под *конусом* $C_v(\bar{V})$ будем понимать множество всех вершин СФЭ Σ (исключая вершины множества $\bar{V}$), которые лежат на ориентированных путях, соединяющих вершины множества $\bar{V}$ с вершиной v. При этом мощность конуса $C_v(\bar{V})$ будем называть его *весом*, вершину v - *выходом конуса*, множество вершин $\bar{V}$ - *входами конуса*, а мощность множества $\bar{V}$ - *шириной* данного конуса. Если множество $\bar{V}$ состоит из всех входов схемы Σ, из которых вершина v достижима, то соответствующий конус будем называть *максимальным* и обозначать через F_v.

Задача функциональной коррекции СФЭ Σ_1 и Σ_2 заключается в поиске минимальных по числу ФЭ пар схем $\{(\Sigma^i, \Sigma_2^i) \mid \Sigma_i^2 \subseteq \Sigma_2\}, i = \overline{1, l}$, таких, что:

- $\Sigma_2^i \cap \Sigma_2^j = \emptyset$ для любых $i, j = \overline{1, l}, i \neq j$.
- Каждому входу Σ^i взаимно-однозначно сопоставлен вход Σ_2^i, каждому выходу Σ^i взаимно-однозначно сопоставлен выход Σ_2^i.
- СФЭ Σ_2', полученная из СФЭ Σ_2 в результате замены всех подсхем Σ_2^i на соответствующие им схемы Σ^i, эквивалентна СФЭ Σ_1.

Набор схем $\{\Sigma^i\}$ будем называть *заплаткой* для СФЭ Σ_2.

III. Предварительный анализ схем

Прежде чем приступить к поиску неисправностей и генерации заплаток система производит анализ исходных схем Σ_1 и Σ_2 с целью выявить заведомо «корректную» часть схемы Σ_2.

На первом шаге проводится симуляция схем. При большом числе входов полная симуляция схемы не представляется возможной ввиду экспоненциального роста числа входных наборов. В таком случае симуляция проводится на ограниченном числе случайных наборов, а при необходимости установить эквивалентность вершин дополнительно используются средства программы ABC [17], предназначенные для проверки эквивалентности комбинационных схем.

На втором шаге для каждой пары соответствующих друг другу выходов $y_1 \in V_{\Sigma_1}$, $y_2 \in V_{\Sigma_2}$ проверяется равенство ФАЛ, реализуемых в них. Если равенство установлено, все вершины, вошедшие в максимальный конус F_{y_2}, фиксируются и заносятся в специальную структуру данных T, которая каждой паре <*PI*,*LUT*> ставит в соответствие список вершин, обладающих заданными свойствами (где PI – множество входов схемы, из которых достижима вершина v; LUT – столбец значений v, полученный в результате симуляции). Эти вершины не могут быть удалены – они войдут в итоговое решение. Если же в рассматриваемой паре выходов реализуются различные ФАЛ, соответствующие выходы запоминаются для дальнейшей обработки.

Обозначим через Σ_2^{free} подсхему схемы Σ_2, содержащую все свободные (нефиксированные) вершины СФЭ Σ_2.

IV. Разбиение схемы на подсхемы меньшего размера

Максимальным деревом для вершины $v, v \in V_\Sigma$ СФЭ Σ будем называть максимальный по весу конус, все вершины которого имеют исходящую степень, равную 1. Нетрудно видеть, что разбиение СФЭ Σ на максимальные деревья единственно.

Разбиение СФЭ Σ на подсхемы меньшего размера проводится в два этапа. Сначала выполняется разбиение схемы на максимальные деревья согласно [18]. Если среди найденных деревьев есть те, ширина которых превосходит заданное число K, производится дальнейшее подразбиение дерева на конусы ширины меньшей или равной K [18]. Подобное разбиение Σ на конусы ограниченной ширины выполняется за линейное от числа вершин время $O(|\Sigma|)$.

Выбор параметра K зависит от решаемой задачи. В данной работе K выбран равным 4 по ряду причин. Прежде всего, такой выбор параметра позволяет в дальнейшем упростить процесс поиска функциональных соответствий между подсхемами. При $K = 4$ достаточно рассмотреть лишь 24 возможные перестановки входов подсхем. Если бы K равнялось 5, таких перестановок было бы уже 120, что при большом числе подсхем значительно повлияло бы на общее время работы алгоритма. Кроме того, параметр K напрямую влияет на размер получаемых подсхем, что, в свою очередь, отражается на размере искомой заплатки. Подсхемы должны быть достаточно большими, чтобы эффективно выявлять эквивалентные фрагменты между исходными схемами, но в то же время достаточно маленькими, чтобы максимально локализовать неисправности и сгенерировать заплатку соответствующего размера. Эмпирическим путем было выявлено, что выбор $K = 4$ является оптимальным для рассматриваемой задачи функциональной коррекции схем.

Конусы, полученные в результате разбиения схемы, для удобства будем называть *блоками*.

V. Алгоритм функциональной коррекции схем

Рассмотрим псевдокод алгоритма функциональной коррекции схем, представленный на рис. 1.

```
FUNCTIONAL_CORRECTION(Σ1, Σ2, neq_output_pairs, T)
    B2 = GET_PARTITION(Σ2, Σ2^free)
    foreach ((y1, y2) in neq_output_pairs)
        upper_bound = GET_UPPER_BOUND(Σ1, y1, T)
        cone_to_check = GET_CONE(Σ1, y1, upper_bound)
        B1 = GET_PARTITION(Σ1, cone_to_check)
        foreach (block in B1)
            matched_block = BOOLEAN_MATCH(block, B2)
            if (matched_block)
                ERASE(B2, matched_block)
            else
                matched_block = NEW_BLOCK(Σ2, block)
            matches[block] = matched_block
            UPDATE(T, matched_block)
    UPDATE_WIRES(Σ2, matches, Σ1)
```

Рис. 1. Псевдокод алгоритма функциональной коррекции

На вход алгоритма подаются СФЭ Σ_1 и Σ_2. При этом каждому входу Σ_1 взаимно однозначно сопоставлен вход Σ_2, а каждому выходу Σ_1 взаимно однозначно сопоставлен выход Σ_2. Также алгоритму подаются список неэквивалентных пар выходов схем

neq_output_pairs и структура данных T, полученные в ходе предварительного анализа схем (раздел III).

На первом шаге при помощи процедуры GET_PARTITION алгоритм записывает в B_2 блоки, полученные в результате разбиения схемы $\Sigma_2^{free} \subseteq \Sigma_2$ согласно разделу IV.

Далее последовательно просматриваются пары выходов из списка *neq_output_pairs*. Для очередной пары $y_1 \in V_{\Sigma_1}, y_2 \in V_{\Sigma_2}$ проводится следующий набор действий:

1) Функция GET_UPPER_BOUND определяет верхнюю границу неисправностей ([11]) в максимальном конусе F_{y_1}. Верхняя граница представляет собой подмножество $\bar{V}$ вершин схемы F_{y_1} такое, что каждой вершине $v \in \bar{V}$ сопоставлена эквивалентная ей вершина из T, и $C_{y_1}(\bar{V})$ - минимальный по весу конус из всех возможных. Данный шаг помогает сократить пространство поиска неисправностей, так как выше верхней границы неисправных фрагментов быть не может.

2) При помощи процедуры GET_PARTITION в структуру B_1 заносятся блоки, полученные в результате разбиения конуса $C_{y_1}(\bar{V})$, $C_{y_1}(\bar{V}) \subseteq F_{y_1}$, на подсхемы меньшего размера. Данный шаг является ключевым в локализации неисправностей, потому как размер итоговой заплатки ограничен сверху суммарной площадью блоков, содержащих неисправные фрагменты. То есть, чем мельче разбиение, тем точнее и меньше генерируемая заплатка.

3) Функция BOOLEAN_MATCH для каждого блока $b_1 \in B_1$ ищет эквивалентный ему блок из B_2 с точностью до перестановки входов и инвертирования выхода b_1 (решается задача поиска функциональных соответствий). Найденный блок удаляется из B_2 при помощи процедуры ERASE. Если же подходящего блока в B_2 не нашлось, процедура NEW_BLOCK генерирует в схеме Σ_2 подсхему-заплатку Σ_{b_1}, являющуюся копией блока b_1. Так или иначе в Σ_2 определится подсхема, соответствующая блоку b_1. Информация о всех установленных соответствиях для блоков из B_1 сохраняется в структуре *matches*.

4) По окончании функция UPDATE_WIRES восстанавливает правильные соединения в схеме Σ_2 между блоками из *matches*. Таким образом, блок за блоком восстанавливается функциональность подсхемы, лежащей ниже верхней границы.

5) Структура T пополняется вершинами из подсхем СФЭ Σ_2, вошедших в *matches*. Это означает, что на следующей итерации цикла эти вершины смогут участвовать в определении верхней границы неисправностей.

На выходе получаем обновленную схему Σ_2' с восстановленной функциональностью и эквивалентную схеме Σ_1.

Представленный алгоритм гарантированно решает задачу функциональной коррекции схем, блок за блоком строя из схемы Σ_2 искомую схему Σ_2'. При структурной схожести схем алгоритм успешно локализует неисправности с точностью до размера блоков, в который они попадают. Самый худший случай предполагает, что в исходных схемах не было ни одной пары эквивалентных выходов и не нашлось ни одной пары соответствующих блоков. Тогда все блоки из Σ_1 будут скопированы в Σ_2, соединены соответствующим образом, и схема Σ_2' будет представлять собой копию схемы Σ_1. К счастью, на практике подобные случаи маловероятны. Ведь, как правило, схемы обладают большими размерами, что при правильно подобранной ширине разбиения порождает много небольших подсхем, среди которых почти наверняка найдутся эквивалентные.

VI. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Проверка корректности работы программной реализации алгоритма проводилась путем установления эквивалентности схем Σ_1 и Σ_2'. Данная проверка осуществлялась с использованием системы для синтеза и верификации СФЭ ABC [17]. Синтаксическая корректность Verilog-описания полученного решения также была установлена при помощи ABC.

Таблица 1

Результаты тестирования для пар схем с заданными видами неисправностей

№	Площадь неисправной части в схеме Σ_2	Площадь заплатки	Доля части схемы Σ_2, сохраненной в схеме Σ_2', %	Время, сек
1	0	0	100	26
2	0	12	99.8	27
3	1	1	99.9	26
4	1	8	99.8	27
5	34	46	99.7	32
6	4987	7431	56.1	84

Ввиду отсутствия шаблонных тестов к задаче функциональной коррекции разработанный алгоритм был протестирован на тестах, сгенерированных вручную. Для тестирования была использована одна из схем, предоставленных в рамках международного соревнования «CAD Contest at ICCAD 2015». Схема площади 13876 (249 входов, 914 выходов) выступала в качестве эталонной, в которую затем вносились неисправности особого вида. В данном наборе тестовых пар схем авторы постарались отразить всевозможные виды неисправностей с тем, чтобы установить область применимости алгоритма, выявить его сильные и слабые стороны. Результаты тестирования представлены в таблице 1.

Тесты 1 и 2 задают пары схем Σ_1 и Σ_2, неэквивалентность между которыми возникает за счет

единичного перебрасывания провода в схеме Σ_2. При этом в тесте 1 переброшено соединение, исходящее из многовыходного, а в тесте 2 - из одновыходного элемента. Отметим, что в первом случае данное соединение заведомо не попадет ни в один из блоков, тем самым не нарушив их эквивалентность, и неисправность будет устранена путем правильного соединения фрагментов схем. Во втором случае «некорректное» соединение, скорее всего, окажется внутри одного из блоков, и заплатка будет построена для неисправного блока целиком.

В тестах 3 и 4 неисправность представляет собой единичную смену типа элемента в схеме Σ_2. Если в тесте 3 неисправный элемент задает свой собственный блок с единственным элементом, то в тесте 4 он попадает в блок площади 8. Таким образом, в обоих случаях заплатка строится по размеру неисправного блока.

Тесты 5 и 6 демонстрируют работу алгоритма на парах схем при малых и значительных неисправностях схемы Σ_2, соответственно.

Тестирование программы также проводилось на ряде других тестов с тем, чтобы установить время ее выполнения на схемах разной площади. В частности, использовался набор из 11 пар неэквивалентных схем соревнования «CAD Contest at ICCAD 2015» [13]. Размер схем варьируется от 7323 до 213224 ФЭ. Как видно из таблицы 2 программа успешно справляется со своей задачей даже на больших (более 100 тысяч элементов) схемах со значительными различиями за разумное время (не более 0.5 часа).

Отличительной особенностью представленного алгоритма является его способность достаточно хорошо локализовать различия между схемами. Из описания [11] двухэтапного алгоритма DeltaSyn с построением верхней и нижней границ неисправностей следует, что, будучи не в силах выявить две последовательные неисправности в пределах одного максимального конуса, DeltaSyn заменяет весь фрагмент схемы между найденными границами. В то время как алгоритм авторов выявляет неисправности с точностью до размера блоков, в которые они попали, что позволяет сохранить «корректную» часть между найденными неэквивалентными фрагментами.

К сожалению, сопоставление результатов работы представленного алгоритма и существующих подходов к решению задачи функциональной коррекции схем не представляется возможным. Отсутствие в свободном доступе единого набора тестов, исчерпывающей информации о вносимых в схемы изменениях и программных реализаций опубликованных алгоритмов не позволяет провести такое сравнение в явном виде.

Исходя из представленных результатов тестирования, можно сделать вывод, что разработанный алгоритм по праву может быть использован наравне с другими алгоритмами функциональной коррекции СФЭ.

ПОДДЕРЖКА

Работа выполнена при финансовой поддержке РФФИ – грант № 18-01-00800-а.

Таблица 2

Результаты тестирования для пар схем соревнования «CAD Contest at ICCAD 2015»

Имя теста	**Число входов**	**Число выходов**	**Число неэквивалентных пар выходов**	**Число ФЭ в Σ_1**	**Число ФЭ в Σ_2**	**Площадь заплатки**	**Доля части схемы Σ_2, сохраненной в схеме Σ_2', %**	**Время, сек**
ut2	249	914	94	13876	10064	7264	53.6	86
ut4	1364	5397	507	78169	52219	19531	69.6	302
ut8	2282	2726	1297	51439	91236	21340	75.4	875
ut9	650	2474	123	59886	48061	27605	61.7	422
ut11	56	129	65	14409	14600	911	93.4	26
ut13	99	128	93	28993	28052	8573	76.5	360
ut15	99	128	117	7323	17572	1641	86.6	29
ut17	128	256	192	82408	46395	70163	39.7	1006
ut19	160	385	285	147784	74618	127222	35.6	1323
ut21	80	288	287	213224	200217	21621	90.2	382
ut23	80	192	146	94707	101754	26457	79.2	885
						Среднее значение по всем тестам:	69.2	

Литература

[1] Jaeger J. Virtually every ASIC ends up an FPGA // EE Times. December 7, 2007.
[2] Lin Chih-chang, Chen Kuang-Chien, Chang Shih-Chieh, Marek-Sadowska Malgorzata, Cheng Kwang-Ting. Logic Synthesis for Engineering Change // Proc. DAC'95. P. 647-652.
[3] Brand D., Drumm A., Kundu S., Narain P. Incremental Synthesis // Proc. ICCAD'94. P. 14-18.
[4] Shinsha T., Kubo T., Sakataya Y., Ishihara K. Incremental Logic Synthesis Through Gate Logic Structure Identification // Proc. DAC'86. P. 391-397.
[5] Sheldon B. Akers. Binary Decision Diagrams // IEEE Transactions on Computers. C-27(6). 1978. P. 509–516.
[6] Veneris A., Hajj I. A fast algorithm for locating and correcting simple design errors in VLSI digital circuits // Proc. Great Lake Symposium on VLSI Design. 1997. P. 45-50.
[7] Huang S.-Y., Chen K.-C., Cheng K.-T. Error correction based on verification techniques // Proc. DAC'96. P. 258-261.
[8] Wu B.-H., Yang C.-J., Huang C.-Y., Jiang J.-H. R. A robust functional ECO engine by SAT proof minimization and interpolation techniques // Proc. ICCAD'10. P. 729–734.
[9] Tang K.-F., Wu C.-A., Huang P.-K., Huang C.-Y. Interpolation-based incremental ECO synthesis for multi-error logic rectification // Proc. DAC'11. P. 146–151.
[10] MiniSat. URL: http://minisat.se/MiniSat.html (дата обращения: 30.03.2018)
[11] Krishnaswamy S., Ren H., Modi N., Puri R. DeltaSyn. An efficient logic-difference optimizer for ECO synthesis // Proc. ICCAD'09. P. 789–796.
[12] Katebi Hadi, Markov Igor L. Large-scale Boolean Matching // Proc. DAC'10. P. 771–776.
[13] C. J. Hsu, C. A. Wu, W. H. Lin, K. Y. Khoo. ICCAD-2015 CAD contest in large-scale equivalence checking and function correction and benchmark suite // Proc. ICCAD'15. Austin, TX. 2015. P. 916-920.
[14] Антюфеев Г.В., Жуков В.В., Зенин Е.Ю., Шуплецов М.С. Методы разбиения логических схем для оптимизации решения задач проверки эквивалентности и функциональной коррекции схем // Сб. трудов Всероссийских научно-технических конференций "Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС)". Том 1. 2016. С. 16-23.
[15] Grace Wu, Yi-Tin Sun, Jie-Hong R. Jiang. Design partitioning for large-scale equivalence checking and functional correction // Proc. DAC'16. ACM. New York, NY. Article 23. 6 pages.
[16] M. Shupletsov, M. Avtaikina. Dynamic programming algorithms for large-scale equivalence checking and functional correction // Proc. EIConRus'17. St. Petersburg. 2017. P. 1032-1035.
[17] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. URL: http://www.eecs.berkeley.edu/~alanmi/abc/ (дата обращения: 30.03.2018)
[18] Cong J., Ding Y. Combinational logic synthesis for LUT based field programmable gate arrays // ACM Trans. Des. Autom. Electron. Syst. 1, 2. 1996. P. 145–204.

Algorithms for Functional Correction of Boolean Circuits

M.A. Avtaikina, M.S. Shupletsov

Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics, shupletsov@cs.msu.ru

***Abstract* — During the IC design flow, functional specifications are often modified late in the design cycle, after placement and routing are completed. However, designers are left either to process such modifications manually or to restart the design flow from the scratch - a very costly option. Functional correction is an automated way of detecting logic difference between a modified high-level specification and an implemented design and generating patch-circuit to resolve the difference. But it is usually not feasible to analyze entire designs for large-scale circuits because of their size and complexity. Partitioning the design, based on two designs' correspondence, can significantly reduce the complexity of the analysis.**

In this paper, a new algorithm for functional correction is described. With partitioning large designs into corresponding smaller designs, our algorithm is able to identify multiple logic differences and modify the existing design minimally such that the new specification can be realized. The specification change is carried out within minimal time because of small sizes of analyzed designs.

Experiments show that this technique successfully implements ECO while preserving the most part of the existing logic. The size of the generated patch does not exceed the sum of sizes of small designs, which hold the logic differences. Unlike the most of previous methods, the developed system can process functional correction for a design of around 100K gates in less than half an hour.

***Keywords* — Boolean circuits, logic synthesis, partitioning, equivalence checking, functional correction, engineering change order, Boolean matching.**

References

[1] Jaeger J. Virtually every ASIC ends up an FPGA // EE Times. December 7, 2007.
[2] Lin Chih-chang, Chen Kuang-Chien, Chang Shih-Chieh, Marek-Sadowska Malgorzata, Cheng Kwang-Ting. Logic Synthesis for Engineering Change // Proc. DAC'95. P. 647-652.
[3] Brand D., Drumm A., Kundu S., Narain P. Incremental Synthesis // Proc. ICCAD'94. P. 14-18.
[4] Shinsha T., Kubo T., Sakataya Y., Ishihara K. Incremental Logic Synthesis Through Gate Logic Structure Identification // Proc. DAC'86. P. 391-397.
[5] Sheldon B. Akers. Binary Decision Diagrams // IEEE Transactions on Computers. C-27(6). 1978. P. 509–516.

[6] Veneris A., Hajj I. A fast algorithm for locating and correcting simple design errors in VLSI digital circuits // Proc. Great Lake Symposium on VLSI Design. 1997. P. 45-50.
[7] Huang S.-Y., Chen K.-C., Cheng K.-T. Error correction based on verification techniques // Proc. DAC'96. P. 258-261.
[8] Wu B.-H., Yang C.-J., Huang C.-Y., Jiang J.-H. R. A robust functional ECO engine by SAT proof minimization and interpolation techniques // Proc. ICCAD'10. P. 729–734.
[9] Tang K.-F., Wu C.-A., Huang P.-K., Huang C.-Y. Interpolation-based incremental ECO synthesis for multi-error logic rectification // Proc. DAC'11. P. 146–151.
[10] MiniSat. URL: http://minisat.se/MiniSat.html (accessed: 30.03.2018)
[11] Krishnaswamy S., Ren H., Modi N., Puri R. DeltaSyn. An efficient logic-difference optimizer for ECO synthesis // Proc. ICCAD'09. P. 789–796.
[12] Katebi Hadi, Markov Igor L. Large-scale Boolean Matching // Proc. DAC'10. P. 771–776.
[13] C. J. Hsu, C. A. Wu, W. H. Lin, K. Y. Khoo. ICCAD-2015 CAD contest in large-scale equivalence checking and function correction and benchmark suite // Proc. ICCAD'15. Austin, TX. 2015. P. 916-920.
[14] G.V.Antiufeev, V.V. Zhukov, E.Y. Zenin, M.S.Shupletsov. Metody razbiyeniya logicheskih skhem dlya optimizacii resheniya zadachi proverki ekvivalentnosti i funkcionalnoy korrekcii skhem (Partitioning methods for Large-Scale Equivalence Checking and Function Correction) // Problems of Perspective Micro- and Nanoelectronic Systems Development. 2016. Part 1, P. 16-23.
[15] Grace Wu, Yi-Tin Sun, Jie-Hong R. Jiang. Design partitioning for large-scale equivalence checking and functional correction // Proc. DAC'16. ACM. New York, NY. Article 23. 6 pages.
[16] M. Shupletsov, M. Avtaikina. Dynamic programming algorithms for large-scale equivalence checking and functional correction // Proc. EIConRus'17. St. Petersburg. 2017. P. 1032-1035.
[17] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. URL: http://www.eecs.berkeley.edu/~alanmi/abc/ (accessed: 30.03.2018)
[18] Cong J., Ding Y. Combinational logic synthesis for LUT based field programmable gate arrays // ACM Trans. Des. Autom. Electron. Syst. 1, 2. 1996. P. 145–204.