УДК 621.3.049.771.14 DOI: 10.31114/2078-7707-2022-4-9-14

Трехуровневая минимизация логических функций с использованием графических ускорителей

В.В. Надоленко

Институт проблем проектирования в микроэлектронике РАН, г. Москва, vl777nd@list.ru

***Аннотация* — В данной статье представлен метод трехуровневой логической минимизации. Основой для него послужил алгоритм двухуровневой минимизации Espresso. Ключевое различие состоит в том, что логическая функция представляется в виде суммы произведений литералов и XOR-группы. Такое представление позволяет адаптировать под себя алгоритмы Espresso и сделать их более эффективными. Масштабируемость метода гарантируется за счет сведения его реализации к алгоритмам Espresso, а также использования матричных операций, позволяющих проводить вычисления на графических ускорителях. На данный момент реализованы и протестированы трехуровневые аналоги expand и irredundant. Тестирование проводилось на схемах из набора LGSynth'91. Предметом оценки являлась эффективность алгоритма в сокращении входной логической функции, а также его быстродействие. Для сравнения использовалось программное средство Espresso в аналогичной конфигурации. Результаты экспериментов показывают, что при дальнейшем исследовании темы у трехуровневой оптимизации есть потенциал развития.**

***Ключевые слова* — логический синтез, двухуровневая минимизация, трехуровневая минимизация, espresso.**

I. Введение

Логический синтез – важнейшая задача в области проектирования микроэлектронных устройств. Он является первым этапом в длинной цепочке автоматизированных превращений от высокоуровневого описания проекта до его топологии и набора производственных процессов, необходимых для изготовления микросхемы [1]. На данном этапе используемое программное средство генерирует нетлист – список логических элементов и их соединений – на основе предоставленного RTL-описания. Задачей логического синтезатора при этом является оптимизация определенных характеристик разрабатываемой схемы: занимаемой площади, задержки распространения сигнала, потребляемой мощности [2]. Однако исходное RTL-описание, как правило, имеет достаточно сложный вид, а технологические библиотеки могут содержать сотни логических элементов с десятками уникальных функций. Поэтому генерация нетлиста не выполняется одним алгоритмом, а разбивается на подзадачи: собственно логический синтез и технологический мэппинг [3]. Для синтеза при этом используется небольшой набор элементов с примитивными функциями, составляющими логический базис – например, AND и INV [4]. Основная оптимизационная работа выполняется над структурой, состоящей из таких абстрактных элементов. Затем та же структура составляется из элементов конкретной технологической библиотеки, т.е. выполняется технологический мэппинг.

В классическом подходе к логическому синтезу присутствует разделение на двухуровневую и многоуровневую оптимизацию. На данном принципе основываются, например, алгоритмы Yosys/ABC [5][6]. Суть двухуровневой минимизации заключается в сокращении ДНФ или КНФ логической функции. Такое представление само по себе неэффективно, однако его примитивность позволяет использовать мощные оптимизационные инструменты и получить приемлемый промежуточный результат. Стандартом в области двухуровневой минимизации является алгоритм Espresso [3], допускающий масштабирование до десятков миллионов термов, что делает его применимым к отдельным модулям современных проектов СБИС. Для получения многоуровневой структуры над ДНФ/КНФ выполняют процедуру факторизации, т.е. выносят общие "множители", применяя дистрибутивный закон. Дальнейшая оптимизация, как правило, выполняется с помощью менее мощных локальных инструментов, эффективность которых зависит от начальной структуры [7].

В данной статье предлагается алгоритм трехуровневой минимизации Xpresso, основанный на идеях Espresso. Данный алгоритм позволяет провести более эффективную глобальную оптимизацию за счет дополнения структуры логическими XOR. Сопутствующие вычисления при этом усложняются, однако масштабируемость сохраняется, как будет показано далее. Кроме того, отдельные вычислительно затратные операции реализованы в виде матричных вычислений и могут выполняться с использованием графических ускорителей.

В следующей главе представлена используемая структура и принципы ее оптимизации. В главе 3 описаны трехуровневые аналоги основных шагов Espresso и сам алгоритм. В четвертой главе приведены результаты тестирования алгоритма на схемах из набора LGSynth'91. Пятая глава заключает статью и описывает направления дальнейшего развития данного исследования.

II. Принципы трехуровневой минимизации

Алгоритм Espresso в своей работе использует ДНФ функции, т.е. сумму произведений. В данном исследовании ДНФ заменяется суммой псевдопроизведений, т.е. произведений литералов и некоторой функции:

$$pp = \bigcap_i X_i * g(X). \quad (1)$$

В нашем случае функция $g(X)$ имеет вид операции XOR над некоторым набором литералов:

$$g(X) = \bigcap_i X_i * \bigoplus_j X_j. \quad (2)$$

Важно, что множества литералов, входящих в произведение и XOR, не пересекаются. Таким образом, произведение можно рассматривать как ограничение некоторой области булева пространства, а XOR – как логическую функцию в данной области. В частном случае, при отсутствии XOR-части функция в области тождественна логической единице.

Рассмотрим возможные оптимизационные действия над такой структурой. Espresso сокращает количество литералов за счет расширения кубов – тех самых ограничивающих областей. Однако при работе с ДНФ функция в каждой области тождественна единице независимо от ее расширений. XOR-функция при исключении литерала из ассоциированного произведения может либо остаться прежней – скопироваться в новую область, – либо пополниться данным литералом – скопироваться в инвертированном виде (рис. 1). Расширение "копией" предпочтительнее, поскольку оно немедленно избавляет функцию от одного литерала, но "инверсия" также потенциально увеличивает области пересечения кубов и помогает сократить их количество.

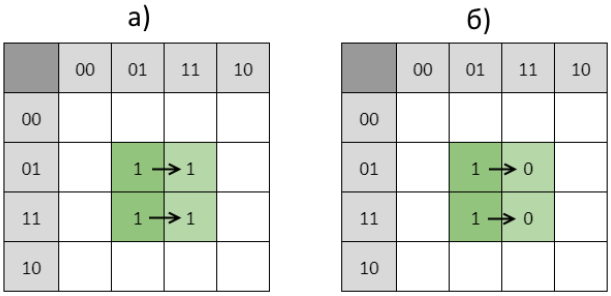

Рис. 1. Расширение куба на примере карты Карно: а) копией; б) инверсией

В классической двухуровневой минимизации для максимального расширения куба используется принцип ограничений. Исходная функция инвертируется, и таким образом составляется список блокирующих кубов. При расширении функционального куба устанавливается запрет на пересечение с блокирующими (иначе функция изменится). Условие непересечения кубов записывается как сумма разделяющих их литералов. Если хотя бы один из этих литералов останется в кубе при расширении, то функция будет корректной. Однако условие непересечения должно выполняться для каждого блокирующего куба, поэтому общее условие корректности расширения имеет вид произведения сумм, т.е. КНФ. Минимальный набор "единичных" значений, удовлетворяющих данной КНФ, задает максимальное расширение функционального куба. Таким образом, поиск максимального расширения сводится к задаче покрытия множеств [8].

Тот же принцип ограничений используется и в предложенном алгоритме трехуровневой минимизации. В первую очередь для каждого потенциально исключаемого литерала при рассмотрении соседних блокирующих кубов определяется тип расширения: прямое или инвертирующее. Таким образом, функция расширяемого куба фиксируется на всем булевом пространстве и более не зависит от набора исключенных литералов. Это позволяет сразу определить наличие конфликта в потенциальных пересечениях. Пусть C – расширенный куб, B – блокирующий куб, H – область их пересечения, в которой они имеют соответствующие функции:

$$C_H = \bigoplus_i X_{Ci}, \quad (3)$$

$$B_H = \bigoplus_i X_{Bi}. \quad (4)$$

Условие отсутствия конфликта остается прежним: непересечение "единичной" функциональной области с "нулевой" блокирующей:

$$\overline{C_H} | B_H \equiv 1. \quad (5)$$

Данное условие выполняется в трех случаях:

$$Q_{C0} = C_H \equiv 0 \ (X_C = \emptyset), \quad (6)$$

$$Q_{B1} = B_H \equiv 1 \ (X_B = \{1\}), \quad (7)$$

$$Q_{eq} = C_H \equiv B_H \ (X_C = X_B). \quad (8)$$

Несложно доказать, что по каждому из критериев Q_{C0}, Q_{B1}, Q_{eq} пересечение либо невозможно ($Q_{C0|B1|eq} = 0$), либо имеет единственный максимальный набор расширений внутри блокирующего куба ($Q_{C0|B1|eq} = \bigcap_i X_i$). Более того, все три критерия не могут выполняться одновременно, поэтому условие корректного расширения примет вид:

$$Q = Q_S + Q_{B1} + Q_{C0} + Q_{eq} = \bigcup_i X_i + \bigcap_j X_j + \bigcap_k X_k \quad , \quad (9)$$

где Q_S – условие непересечения.

Отметим, что Q можно привести в КНФ, используя дистрибутивный закон. Полученное количество сумм будет выражаться следующим образом:

$$N = |X_j| * |X_k|. \quad (10)$$

Очевидно, что N может получиться достаточно большим и серьезно замедлить решение задачи покрытия. Однако это число можно ограничить сверху, убрав из рассмотрения слишком громоздкие условия $Q_{C0|B1|eq}$. Q_S при этом гарантирует корректность расширения.

III. Алгоритм

На данный момент в предложенный алгоритм входят треуровневые аналоги функций Espresso expand и irredundant.

A. 3-EXPAND

Из предыдущей главы видно, что получение функции корректности расширения требует множества однотипных действий. На рис. 2 представлен маршрут матричных операций от входных кубов до задачи покрытия, совместимый с графическими ускорителями. В первую очередь определяются области минимального пересечения расширяемого куба с каждым блокирующим. Затем для минимальных пересечений определяются возможные дальнейшие расширения по каждому из критериев Q_{C0}, Q_{B1}, Q_{eq}. На данном этапе формируется матрица флагов расширения по каждой переменной для каждого блокирующего куба. Типы расширения еще не выбраны, поэтому функция в области минимального пересечения может быть как прямая, так и инвертированная, т.е. ее фаза не определена. Расширение минимальных пересечений имеет смысл только в том случае, когда в них изначально нет конфликта, поэтому к ним применяется дополнительный фильтр. Наконец, после определения "конфликтных" пересечений определяется тип расширения по каждой переменной, матрица сокращается. Далее из нее также убираются бесполезные условия по критериям Q_{C0}, Q_{B1}, Q_{eq}, а оставшиеся записываются в "дополнительную" часть матрицы покрытия. Задача покрытия может решаться как с этими дополнительными столбцами в матрице, имеющими веса в соответствии с количеством литералов, так и после преобразования в обычную матрицу покрытия по дистрибутивному закону.

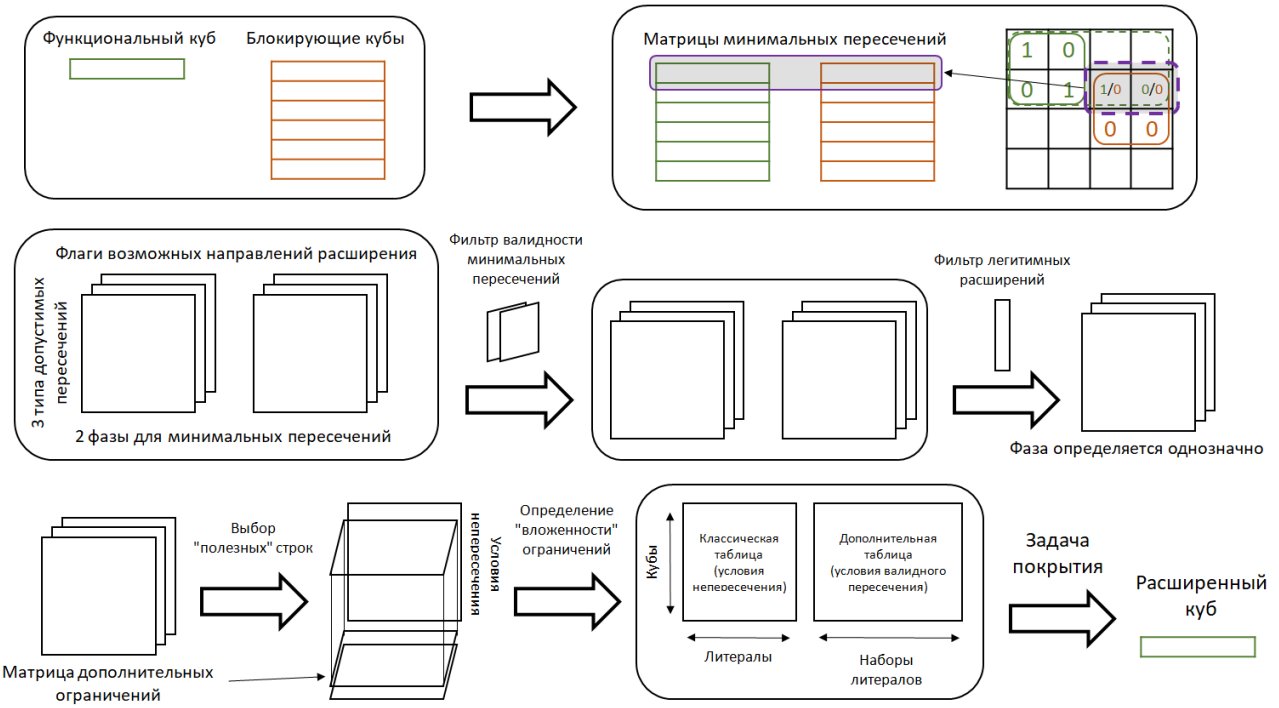

Рис. 2. Решение задачи расширения трехуровневого куба в матричном виде

B. 3-IRREDUNDANT

Алгоритм irredundant позволяет определить избыточные кубы в псевдопроизведении и потому вносит наибольший вклад в минимизацию функции. В данном исследовании применяется жадный алгоритм, выполняющий перебор всех кубов и исключающий те из них, которые полностью покрываются остальными. Более формально, куб считается избыточным в том случае, если функция остальных кубов в его области, сложенная с инвертированной функцией самого куба, тождественна единице.

Задача сравнения функции в виде суммы псевдопроизведений решается с помощью рекурсивного выполнения разложений Шеннона. Таким образом изначальная область разбивается на части, а функция в них упрощается. 3 важных наблюдения помогают решить задачу быстрее:

1. Монотонная функция в виде ДНФ (т.е. функция, не имеющая одного и того же литерала с инверсией и без) не может быть тождественна единице, если единица не входит в нее как терм.

2. Сумма XOR-функций может быть проверена на наличие нулей конвертированием в систему уравнений. Поскольку XOR - линейная операция, то и система будет линейной, решить ее не составит труда.

3. Смешанная функция может быть разделена, если множества литералов в ДНФ части и XOR-части не пересекаются.

Из вышеуказанного следует, что при выполнении разложений Шеннона эффективно исключать в первую очередь литералы, которые делают ДНФ часть не монотонной и препятствуют разделению функции.

IV. Экспериментальные результаты

Эксперименты проводились на наборе логических схем LGSynth'91. Сравнение производилось со связкой expand+irredundant алгоритма Espresso. В качестве метрик эффективности использовалось количество кубов, литералов и операций в итоговой функции, а также среднее количество расширений в кубах. Таблица 1 показывает результаты проведенных экспериментов для некоторых схем. Из представленных значений видно, что трехуровневая минимизация имеет более высокий потенциал, хотя в редких случаях дополнительная сложность отрицательно сказывается на эффективности первой итерации. На рис. 3 представлены экспериментальные данные по всему набору схем в виде графиков.

Таблица 1

Сравнение эффективности Espresso expand+irredundant и предложенного алгоритма на схемах из набора LGSynth'91

Схема	Кубы		Литералы		Среднее количество расширений		Количество операций	
	Espresso	Xpresso	Espresso	Xpresso	Espresso	Xpresso	Espresso	Xpresso
9sym	87	66	522	396	3.00	5.00	521	330
apex5	1088	1038	6089	5757	111.40	111.58	289435	276073
clip	137	104	695	528	3.93	4.94	2242	1656
con1	9	9	23	23	4.44	4.44	46	46
cordic	1180	626	18213	9883	7.57	9.22	21751	11396
cps	193	211	2232	2541	12.44	12.27	61777	67933
ex1010	735	584	7047	4816	0.41	1.80	25630	12933
rd53	31	13	140	53	0.48	2.38	301	120
rd73	127	68	756	339	1.05	2.85	1385	704
rd84	255	158	1774	998	1.04	2.13	3753	2236
sao2	58	47	423	318	2.71	3.94	869	678
xor5	16	1	80	5	0.00	5.00	79	4

Также проводилось сравнение быстродействия алгоритма expand, однако время на решение задачи покрытия при этом не учитывалось. В данном исследовании не предлагается новых методов ее решения, а существующие предсказуемо масштабируются. При этом увеличение размерности задачи при переходе к трехуровневой минимизации регулируется незначительными ограничениями, как было показано в главе 2. Целью сравнения было показать, что принципиально новые вычисления также занимают приемлемое время, даже с учетом реализации на интерпретируемом языке программирования Python.

Таблица 2

Сравнение быстродействия Espresso expand и предложенного алгоритма на схемах из набора LGSynth'91

Схема	5xp1	9sym	apex1	apex3	apex5	con1	spla	t481
Espresso	0.064 с	0.063 с	0.089 с	0.105 с	0.295 с	0.043 с	0.244 с	0.079 с
Xpresso	0.064 с	0.071 с	0.397 с	0.263 с	2.258 с	0.016 с	2.218 с	0.504 с

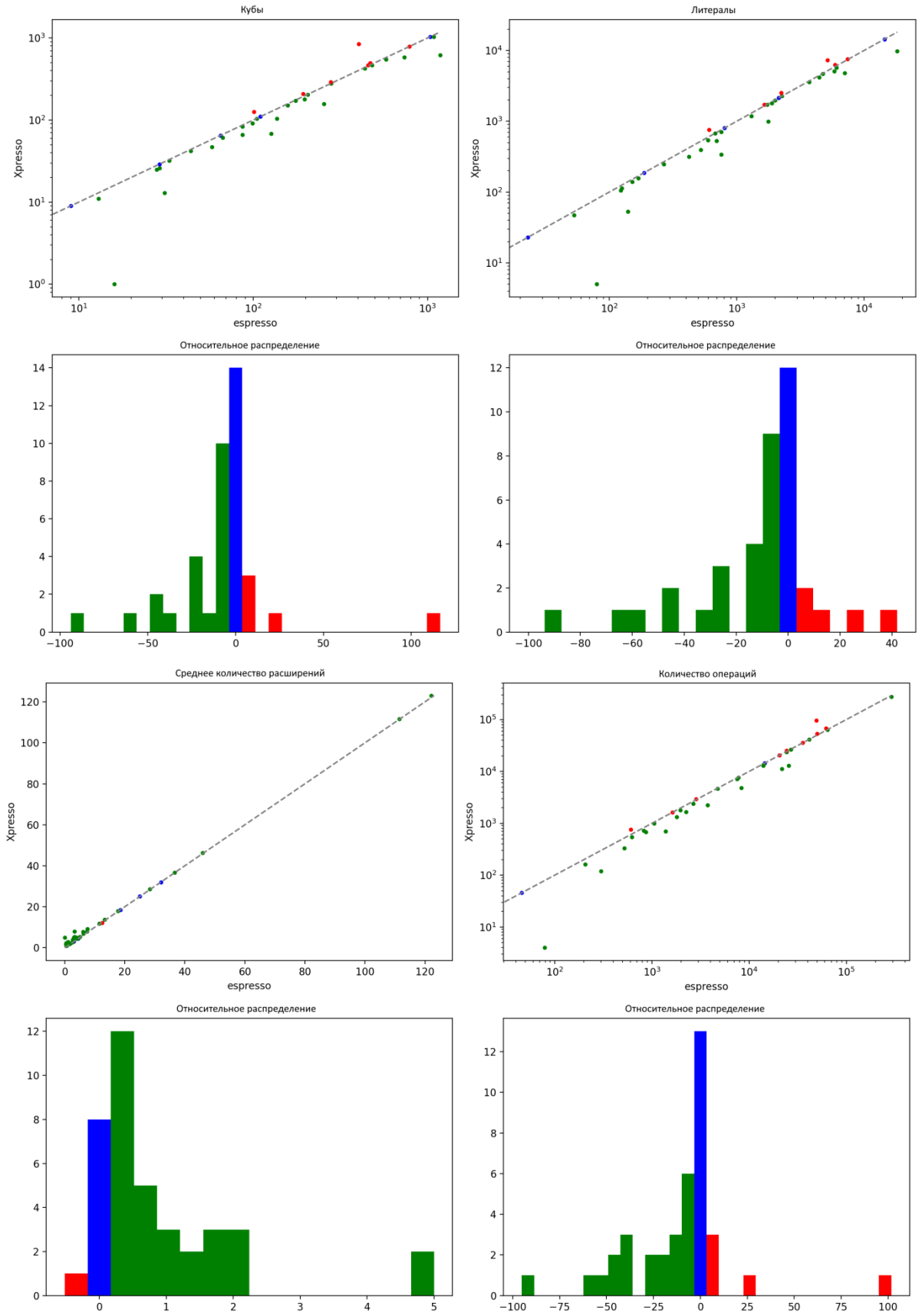

Рис. 3. Сравнение эффективности Espresso и предложенного алгоритма

V. Заключение

В данной работе представлен алгоритм трехуровневой минимизации логических функций. На текущий момент он содержит в себе аналоги шагов expand и irredundant известного двухуровневого минимизатора Espresso. Экспериментальные данные показывают, что у такого подхода есть потенциал развития как в сторону эффективности, так и в сторону увеличения быстродействия. Дальнейшие исследования могут вестись по двум основным направлениям: разработка трехуровневого аналога reduce для реализации полного итеративного алгоритма и представление большего количества вычислений в матричном виде для распараллеливания.

Литература

[1] С.Г. Бобков, А.С. Басаев. Методы и средства аппаратного обеспечения высокопроизводительных микропроцессорных систем. 2021.

[2] Keutzer, K., Ravindran, K. (2008). Technology Mapping. In: Kao, MY. (eds) Encyclopedia of Algorithms. Springer, Boston, MA.

[3] Rudell, R.: Logic Synthesis for VLSI Design. Ph. D. thesis, University of California at Berkeley, ERL Memo 89/49, April 1989

[4] Biere, A. The AIGER And-Inverter Graph (AIG) Format. Linz, Austria: [s.n.], 2007.

[5] http://yosyshq.net/yosys

[6] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. 2018.

[7] Mishchenko, Alan & Brayton, Robert. (2006). Scalable Logic Synthesis using a Simple Circuit Structure.

[8] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. European Journal of Operational Research, vol. 101, pp. 81-92, **1997.**

Three Level Logic Minimization using Graphics Processing Units

V.V. Nadolenko

Institute for Design Problems in Microelectronics of RAS, Moscow, vl777nd@list.ru

***Abstract* — This paper presents a method for three level logic minimization. It takes inspiration from two level minimization algorithm Espresso and attempts to enhance its capabilities by working with more complex Boolean formulae. Espresso only processes sums or products resulting in two level logic circuits at the output. Proposed algorithm called Xpresso works with sums of pseudo-products that consist of product of some literals and XOR of some other literals. This allows better global optimization before running any local multi-level minimization algorithms. Pseudo-product processing relies on Espresso principles of expanding cubes and removing redundant ones. Classic cube is a subspace of minimized function's domain in which the function takes value of 1. Three level cube also defines a subspace but it can be filled with 1's in an XOR pattern instead. Taking that into account allows introducing new expansion principles. Espresso cubes can only expand as long as they do not run into blocking areas of inverted function. Xpresso cubes, however, not necessarily conflict with function's off-set on overlap as they can also include off-set areas. This complicates expansion's validity function but it still can be converted to product of sums form under some restrictions. Hence maximal expansion can be found as a solution to set covering problem, similar to Espresso's approach. In the same fashion this paper suggests a way to irredundant algorithm on three level cubes. Furthermore, expand's computations up to set covering are represented in matrix form and can be performed with graphics processing unit. Both algorithms — expand and irredundant — were compared to those in Espresso by their effectiveness measured in number of cubes, literals, operations and average expansions in resulting functions. Results show that further research has promising perspectives.**

***Keywords* — logic synthesis, two level minimization, three level minimization, espresso.**

References

[1] S.G. Bobkov, A.S. Basaev. Metody i sredstva apparatnogo obespechenija vysokoproizvoditel'nyh mikroprocessornyh sistem (Methods of providing hardware for high performance microprocessor systems). 2021.

[2] Keutzer, K., Ravindran, K. (2008). Technology Mapping. In: Kao, MY. (eds) Encyclopedia of Algorithms. Springer, Boston, MA.

[3] Rudell, R.: Logic Synthesis for VLSI Design. Ph. D. thesis, University of California at Berkeley, ERL Memo 89/49, April 1989

[4] Biere, A. The AIGER And-Inverter Graph (AIG) Format. Linz, Austria: [s.n.], 2007.

[5] http://yosyshq.net/yosys

[6] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. 2018.

[7] Mishchenko, Alan & Brayton, Robert. (2006). Scalable Logic Synthesis using a Simple Circuit Structure.

[8] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. European Journal of Operational Research, vol. 101, pp. 81-92, 1997.