Расчет таблицы истинности при верификации логических схем путем векторизации вычисления логических функций

А. В. Акимов, email: akimov@vsu.ru

Воронежский государственный университет

Аннотация. *Представлен основанный на векторизации кода подход к увеличению скорости расчета таблицы истинности (ТИ) при верификации логических схем. Приведена схема и описан принцип работы векторизованного алгоритма, основанный на упаковке входных значений сразу нескольких строк ТИ в одну переменную (по одной для каждого входа исследуемой схемы) и расчете логической функции для всего их набора с последующей распаковкой результатов. Проведены и проанализированы результаты экспериментов по расчету ТИ, показывающие преимущество векторизованной версии алгоритма.*

Ключевые слова: *верификация логических схем, бинарная логика, векторизация кода, таблица истинности.*

Введение

Одной из задач, возникающих при проектировании логических схем, является их верификация. Верификация выступает как один из этапов создания программного или аппаратного обеспечения и используется в данном частном случае для того, чтобы проверить, что получившаяся логическая схема правильно функционирует и соответствует предъявляемым к ней требованиям [1-3]. По принципу работы логические элементы и схемы их них составленные можно разделить на комбинационные и последовательные [4, 5]. Для комбинационной логики характерно, что выходные значения зависят только от текущей комбинации значений входных величин. В случае последовательной логики в дело вступает также их внутреннее состояние, поэтому такие элементы в том числе называют элементами с памятью.

Одним из инструментов, используемых при верификации логических схем комбинационного типа, является таблица истинности (ТИ) [3, 6]. Ее построение требует перебора всех возможных комбинаций значений входов исследуемой схемы и расчета для каждой из них, что будет на ее выходах. Из-за таким образом получающейся экспоненциальной сложности вычислений при переборе всех строк ТИ

отмечается ее слабая пригодность для работы с большими схемами [7, 8], и потому является важной задача ускорения расчета ТИ всеми доступными методами.

Одним из возможных подходов здесь является векторизация кода [9]. Благодаря специфике задачи – расчет логических операций, которые можно выполнять побитово, – ускорение достигается за счет упаковки входных значений сразу нескольких строк ТИ в одну переменную (по одной для каждого входа исследуемой схемы), запуск расчета логической функции сразу для всего набора упакованных значений с последующей распаковкой результатов. Далее о предлагаемой технике будет рассказано более подробно.

1. Расчет таблицы истинности

Таблица истинности за пределами схемотехники используется в общем случае в исчислении высказываний для проверки и доказательства логических законов и выражений [10], и, как уже упоминалось ранее, ее расчет основывается на переборе всех возможных комбинаций значений аргументов исследуемого логического выражения.

На практике при создании программной реализации такой механизм достаточно легко реализовать, перебирая значения специальной переменной-счетчика с шагом 1, и, для известного порядка столбцов ТИ, используя значения ее битов в качестве текущей комбинации значений входов логического выражения. Самый младший бит в данном случае будет соответствовать самому правому и часто перебираемому входному столбцу ТИ.

Что касается вычисления логики, выполняемой исследуемой схемой для каждой конкретной строки ТИ, то здесь возможно два основополагающих подхода: либо интерпретация ее поведения на лету за счет ее анализа как дерева вычислений, либо предварительное формирование набора логических выражений, ею реализуемых, и последующий расчет уже их целиком. Сами логические выражения, сформированные таким образом, или исходно прописанные для каждого отдельного элемента схемы, по сути, представляют собой строки, подлежащие интерпретации. И в первом (для расчета логики, выполняемой конкретными элементами) и во втором (целиком) случае здесь помимо собственного их анализа и интерпретации возможно применение встроенных или сторонних интерпретаторов, доступных из основной среды разработки, например, передача их для расчета интерпретатору JavaScript QML внутри среды QT C++ [11], подход, использованный в данной работе.

Последнее замечание касается еще одной специфики данной частной задачи – исследование или верификация произвольной схемы комбинационной логики, в которой в общем случае возможно наличие нескольких выходов, – и таким образом ТИ может содержать более одного столбца результата. На практике это означает, что для каждой строки ТИ для одной и той же комбинации входов вызывается расчет нескольких логических выражений, по одному для каждого выхода исследуемой схемы. (Вопрос возможного ускорения вычислений за счет анализа повторяющихся ветвей логического выражения и переиспользования уже частично известных результатов при этом остается за рамками данной работы.)

2. Векторизация вычисления логических функций

Исследуемая логическая схема может содержать несколько входов и выходов. Каждый из них будет соответствовать одному столбцу ТИ. Для расчета значения каждого выходного столбца ТИ в общем случае используется своя собственная логическая функция, полученная на основе анализа поведения схемы для данного выхода. При этом ее аргументы соответствуют всем входам схемы или входным столбцам ТИ.

Ускорение вычислений достигается за счет упаковки нескольких следующих друг за другом значений строк ТИ для каждого входа в соответствующее значение аргумента логической функции. Так, если ее вычисления производится средствами JavaScript, то в одну переменную типа `number` можно упаковать до 32 бит одновременно. Таким образом за один вызов обычной логической функции, в которой вместо стандартных логических операций используются их побитовые аналоги, можно рассчитать сразу 32 строки ТИ. Значения на выходе данной функции далее распаковываются в нужное число строк для соответствующего этой функции столбца ТИ. Блок-схема алгоритма расчета ТИ с использованием описанной схемы ускорения вычислений за счет их векторизации представлена на рисунке.

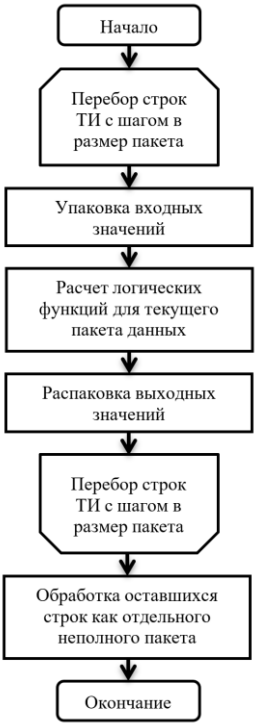

Рисунок. Блок-схема алгоритма векторизации вычисления логических функций

Увеличение производительности в данном случае достигается за счет возможной большей сложности исследуемой логической функции по отношению к сложности операций упаковки и распаковки входных и выходных ее аргументов, что обычно соответствует большому числу входов-строк рассчитываемой ТИ, по сути когда и возникает необходимость в ускорении вычислений. Специфичность данной задачи – работа с бинарной логикой, позволяет достаточно легко реализовать стандартную технику векторизации кода – замены самого внутреннего цикла с определенным числом шагов на одну операцию над большей структурой данных [9] и при этом на достаточно высоком уровне, не прибегая к обычно необходимому при векторизации вычислений написанию низкоуровневых библиотек или вызову специфичных функций запуска векторизованных вычислений напрямую [9, 12, 13].

3. Проведение и анализ экспериментов по сравнению скорости работы обычного и векторизованного алгоритмов расчета ТИ

Были проведены эксперименты по сравнению скорости работы обычного и векторизованного алгоритмов расчета ТИ для разного числа строк и логических схем разной степени сложности с большим числом входов. В таблице представлены типичные результаты для одной и той же схемы, содержащей 75 элементов, 32 входа и 20 выходов, глубиной от выхода ко входу от 1-го до в основном порядка 7-ми элементов при последовательном увеличении в два раза задаваемого при расчете числа строк от 2 до 1048576. Для получения каждого замера скорости работы эксперимент по расчету данного числа строк ТИ проводился 100 раз подряд и в таблицу записано среднее значение.

Как видно из таблицы выигрыш в скорости отсутствует только при расчете двух строк таблицы истинности. Далее векторизованный алгоритм начинает постепенно одерживать верх до достижения уже при работе с ТИ размером в 512 строк преимущества при его использовании порядка 20 раз, которое продолжает далее сохраняться. Таким образом при использовании векторизации ощутимая в пол секунды для пользователя задержка в получении результатов откладывается при работе с ТИ размером в 32768 строк, что соответствует схеме с 15-ю входами, до ТИ размером в 524288 строк (схема с 19 входами).

Отдельно следует отметить эффект, заключающийся в постоянной скорости работы векторизованной версии алгоритма при работе с ТИ от 2-х до 32-х строк. Он связан в первую очередь с размером пакета векторизации в 32 бита, и таким образом, тестируя все на одной и той же схеме, мы имеем один и тот же объем действий, необходимый для выполнения алгоритма. Тем не менее, если заглянуть в ту же верхнюю строчку таблицы, в сравнении с обычным вариантом алгоритма видно, что накладные расходы по упаковке и распаковке бит в пакет для расчета набора из 32-х строк ТИ за раз не играют никакой ощутимой роли на фоне самого расчета ТИ для схемы данного уровня сложности.

Таблица

Эксперименты по сравнению скорости работы обычного и векторизованного алгоритмов расчета ТИ

№ п/п	Число строк ТИ	Без векторизации, мс	С векторизацией, мс	Выигрыш в скорости, число раз
1	2	0,05	0,05	1,0
2	4	0,08	0,05	1,6
3	8	0,15	0,05	3,0
4	16	0,28	0,05	5,6
5	32	0,55	0,05	11,0
6	64	1,08	0,08	13,5
7	128	2,16	0,13	16,6
8	256	4,31	0,23	18,7
9	512	8,59	0,42	20,5
10	1024	17,25	0,81	21,3
11	2048	34,30	1,59	21,6
12	4096	68,65	3,14	21,8
13	8192	136,61	6,26	21,8
14	16384	272,29	12,57	21,7
15	32768	546,30	25,49	21,4
16	65536	1091,77	51,14	21,3
17	131072	2187,53	104,21	21,0
18	262144	4365,85	209,70	20,9
19	524288	8760,34	421,40	20,8
20	1048576	17492,22	848,77	20,6

Заключение

В данной работе был представлен основанный на векторизации кода подход к увеличению скорости расчета таблицы истинности при верификации логических схем. Описана техника ее построения с использованием для расчета логических функций встроенного в среду Qt C++ интерпретатора JavaScript QML и особенности, возникающие в частном случае задачи верификации комбинационной логики.

Приведена схема и описан принцип работы векторизованного алгоритма, основанный на упаковке входных значений сразу нескольких строк ТИ в одну переменную (по одной для каждого входа исследуемой схемы) и расчете логической функции для всего их набора с последующей распаковкой результатов. Проведены и проанализированы результаты экспериментов по построению таблицы истинности при постепенном двукратном увеличении числа рассчитываемых строк, показывающие преимущество векторизованной версии алгоритма порядка 20 раз при достаточно большом числе строк ТИ.

Список литературы

1. Gupta, A. Formal Hardware Verification Methods: A Survey / A. Gupta // Computer-Aided Verification. – Springer, Boston, MA, 1992. – P. 5-92.
2. Kern, C. Formal Verification in Hardware Design: A Survey / C. Kern, M. R. Greenstreet // ACM Transactions on Design Automation of Electronic Systems (TODAES). – 1999. – Vol. 4. – № 2. – P. 123-193.
3. Захарова, О. В. Верификация программной системы моделирования АЛУ непосредственного формирования в процессах проектирования / О. В. Захарова // Информационные системы и технологии. – 2014. – Т. 84. – № 4. – С. 51-61.
4. Угрюмов, Е. П. Цифровая схемотехника: Учеб. пособие для вузов / Е. П. Угрюмов. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 816 с.: ил.
5. Томас, Д. Логическое проектирование и верификация систем на SystemVerilog / Д. Томас; пер. с анг. А. А. Слинкина, А. С. Камкина, М. М. Чупилко; науч. ред. А. С. Камкин, М. М. Чупилко – М.: МДК Пресс, 2019. – 384 с.: ил.
6. Dias, F. J. O. Truth-table Verification of an Iterative Logic Array / F. J. O. Dias // IEEE Transactions on Computers. – 1976. – Vol. 25. – № 06. – P. 605-613.
7. d'Agostino, M. Are Tableaux an Improvement on Truth-Tables? / M. d'Agostino // Journal of Logic, Language and Information. – 1992. – Vol. 1. – № 3. – P. 235-252.
8. Oumarou, O. QUANTIFY: A Framework For Resource Analysis And Design Verification of Quantum Circuits / O. Oumarou, A. Paler, R. Basmadjian // 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). – IEEE, 2020. – P. 126-131.
9. Воеводин, В. В. Параллелизм в сложных программных комплексах (почему сложно создавать эффективные прикладные

пакеты) / В. В. Воеводин // Чебышевский сборник. – 2017. – Т. 18. – №. 3 (63). – С. 187-200.

10. Bi, H. H. Applying Propositional Logic to Workflow Verification / H. H. Bi, J. L. Zhao // Information Technology and Management. – 2004. – Vol. 5. – № 3 – P. 293-318.

11. Qt QML [Электронный ресурс]: JavaScript Environment for QML Applications. – Режим доступа: https://doc.qt.io/qt-6/qtqml-index.html#javascript-environment-for-qml-applications

12. Nie, J. Vectorization for Java / J. Nie, B. Cheng, S. Li, L. Wang, X.-F. Li // IFIP International Conference on Network and Parallel Computing. – Springer, Berlin, Heidelberg, 2010. – P. 3-17.

13. Jensen, P. SIMD in JavaScript via C++ and Emscripten / P. Jensen, I. Jibaja, N. Hu, D. Gohman, J. McCutchan // Workshop on Programming Models for SIMD/Vector Processing. – 2015. – P. 1-7.