ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2022, № 3, с. 69–80

ДИСКРЕТНЫЕ СИСТЕМЫ

УДК 004.312.4

СИНТЕЗ БЫСТРЫХ КОНЕЧНЫХ АВТОМАТОВ НА ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМАХ ПУТЕМ РАСЩЕПЛЕНИЯ ВНУТРЕННИХ СОСТОЯНИЙ[1]

© 2022 г. В. В. Соловьев

Белостокский технологический ун-т, Белосток, Польша

email: valsol@mail.ru

Поступила в редакцию 14.03.2020 г.
После доработки 06.12.2021 г.
Принята к публикации 31.01.2022 г.

Представлен метод синтеза быстрых конечных автоматов на программируемых логических интегральных схемах типа программируемых пользователем вентильных матриц. Метод основан на операции расщепления внутренних состояний, что позволяет снизить ранги функций переходов и уменьшить число уровней функциональных генераторов при реализации функций переходов. Приводятся оценки числа уровней функциональных генераторов при реализации функций переходов конечного автомата в случае последовательной и параллельной декомпозиции. Описывается алгоритм расщепления внутренних состояний для синтеза быстрых конечных автоматов. Результаты экспериментальных исследований показали, что предложенный метод позволяет увеличить быстродействие конечных автоматов в среднем от 1.08 до 1.19 раза, а максимально – от 1.52 до 1.73 раза. Выполнено также сравнение представленного метода с университетскими программами JEDI и NOVA.

DOI: 10.31857/S0002338822030131

Введение. Крупные функциональные блоки и узлы цифровой системы, а также сама цифровая система, как правило, включают устройство управления или контроллер. Быстродействие цифровой системы и составляющих ее функциональных блоков непосредственно зависит от быстродействия их устройств управления. Математической моделью большинства устройств управления и контроллеров является конечный автомат. Поэтому для построения высокопроизводительных цифровых систем необходимы методы синтеза быстрых конечных автоматов. При синтезе быстрых конечных автоматов можно пренебречь стоимостью реализации, поскольку площадь на кристалле, занимаемая устройством управления, составляет небольшую часть по сравнению с другими компонентами системы.

В настоящее время в качестве элементной базы для построения цифровых систем широко используются программируемые логические интегральные схемы (ПЛИС). Среди множества архитектур ПЛИС широкое распространение получили два типа архитектур [1]: на основе двух программируемых матриц И и ИЛИ, а также на базе функциональных генераторов типа LUT (look up table). ПЛИС первого типа получили название сложные программируемые устройства (complex programmable logic devices – CPLD), а второго – программируемые пользователем вентильные матрицы (field programmable gate arrays – FPGA). Системы на кристалле (system on chip – SoC) также имеют в своей архитектуре конфигурируемую часть со структурой FPGA.

Архитектуру FPGA можно представить как совокупность большого количества функциональных генераторов LUT, объединяемых межсоединениями. Каждый элемент LUT позволяет реализовать произвольную булеву функцию от небольшого числа аргументов (4–6). Методы синтеза быстрых конечных автоматов на CPLD были рассмотрены в [1]. В данной работе приведен метод синтеза быстрых конечных автоматов на FPGA.

Проблемам проектирования быстрых конечных автоматов на ПЛИС посвящено достаточно много работ. В [2] представлен метод повышения производительности синхронных схем при их

[1] Работа выполнена при финансовой поддержке Белостокского технологического университета (Польша) (грант № WZ/WI-IIT/4/2020).

реализации на FPGA путем изменения расположения регистров. В [3] описан метод синтеза и оптимизации последовательностных схем на FPGA. Метод основан на концепции синтеза, управляемого информацией, общей декомпозиции и теории отношений информационных мер. В [4] предлагается метод оптимизации тактирования сложных конечных автоматов путем добавления в схему избыточного блока. В [5, 6] исследуются стили описания конечных автоматов на языке VHDL, а также известные способы кодирования внутренних состояний для реализации быстрых конечных автоматов. В [7] применяются эволюционные методы для минимизации площади кристалла и задержки выходных сигналов конечного автомата. В [8] рассматривается задача кодирования внутренних состояний и оптимизация комбинационной схемы при реализации на CPLD быстрых конечных автоматов. В [9] представлена новая архитектура программируемой логики, специально предназначенная для реализации конечных автоматов, которая позволяет сократить площадь кристалла, задержки сигналов и потребляемую мощность. В [10] предложена новая модель автомата, названная виртуальным конечным автоматом. Для реализации виртуального конечного автомата применяется архитектура, основанная на блоках памяти. В [11] рассматривается реализация конечных автоматов на FPGA с использованием встроенных блоков памяти, которая позволяет уменьшить площадь кристалла и увеличить быстродействие конечного автомата. В [12, 13] представлены методы минимизации внутренних состояний полностью и неполностью определенных конечных автоматов, которые также способствуют увеличению быстродействия конечного автомата. В [14] описывается архитектура конечного автомата с одним входом на основе бинарного дерева, для реализации которого используется модель виртуального конечного автомата [10]. В [15] рассматривается метод увеличения быстродействия микропрограммных автоматов типа Мура путем введения дополнительных внутренних состояний.

Анализ известных подходов для проектирования быстрых последовательностных схем и конечных автоматов показал следующее. Отдельные методы базируются на внесении изменений в схему устройства путем перестановки регистров [2] или добавления избыточных блоков [4], что не всегда допустимо и может привести к усложнению анализа и отладки схемы конечного автомата. Метод [3], основанный на теории отношений информационных мер, видится перспективным, однако нет сообщений о его практическом применении. Работы [5, 6] ничего нового в решение проблемы не вносят, это традиционный подход: выбор из имеющихся возможностей наиболее подходящего стиля описания конечного автомата и способа кодирования внутренних состояний. Генетические и эволюционные методы [7] не годятся для практического использования из-за своей большой вычислительной сложности. Методы синтеза быстрых конечных автоматов на CPLD [1, 8] не применимы при реализации конечного автомата на FPGA. Специальные архитектуры для реализации быстрых конечных автоматов [9, 10] существуют только в теории. Методы [11, 14] имеют существенные ограничения на условия применения. Методы [12, 13] незначительно увеличивают быстродействие конечных автоматов. Метод [15] годится для синтеза только микропрограммных автоматов, поведение которых описывается на языке граф-схем алгоритмов.

Таким образом, проведенный анализ показал, что отсутствуют эффективные методы логического синтеза быстрых конечных автоматов на FPGA. Поэтому открытым остается вопрос: как синтезировать и практически реализовать быстрый конечный автомат на реальных FPGA?

В [16] рассматривалась задача снижения числа аргументов функций переходов путем расщепления внутренних состояний конечного автомата, что способствует как снижению стоимости реализации, так и повышению быстродействия конечных автоматов при их реализации на FPGA. Однако в [16] не исследовалось, на сколько увеличивается быстродействие конечных автоматов.

В работе также используется операция расщепления внутренних состояний, но целью такого расщепления является повышение быстродействия конечных автоматов на FPGA, основанных на функциональных генераторах LUT. Операция расщепления внутренних состояний относится к операциям эквивалентных преобразований конечного автомата, которая не изменяет алгоритм его функционирования. При расщеплении внутренних состояний также сохраняется тип автомата: Мили или Мура.

Суть рассматриваемого метода заключается в уменьшении числа уровней элементов LUT комбинационной схемы, которая реализует функции переходов конечного автомата. В результате снижается задержка сигналов, питающих элементы памяти, и быстродействие конечного автомата увеличивается. Для уменьшения числа уровней элементов LUT используется операция расщепления внутренних состояний конечного автомата. Однако бесконтрольное расщепление состояний может привести к обратному эффекту: увеличению числа уровней LUT. Поэтому в

методе предусмотрено условие, когда процесс расщепления внутренних состояний следует остановить.

Статья организована следующим образом. В разд. 1 определяются ранги функций переходов и число уровней функциональных генераторов LUT, необходимых для их реализации. В разд. 2 описывается метод синтеза быстрых конечных автоматов. В разд. 3 рассматривается расщепление внутренних состояний для уменьшения числа уровней элементов LUT. В разд. 4 представлены результаты экспериментальных исследований. В заключение обсуждаются результаты экспериментальных исследований и указываются перспективные направления проектирования быстрых конечных автоматов.

1. Определение числа аргументов (рангов) функций переходов и числа уровней элементов LUT для их реализации. Конечный автомат будем характеризовать числом M внутренних состояний множества $A = \{a_1, ..., a_M\}$, числом L входных переменных множества $X = \{x_1, ..., x_L\}$, числом N выходных переменных множества $Y = \{y_1, ..., y_N\}$, а также множеством D функций переходов (функций возбуждения элементов памяти).

Для синтеза быстрых конечных автоматов на FPGA традиционно используется унарное кодирование (one-hot) внутренних состояний, при этом каждому внутреннему состоянию соответствует отдельный триггер памяти автомата, установка которого в значение 1 указывает, что автомат находится в данном состоянии. Вход каждого триггера управляется функцией переходов d_i, $d_i \in D$, $i = \overline{1, M}$, т.е. каждому внутреннему состоянию автомата соответствует своя функция переходов d_i.

Пусть $A(a_i)$ – множество состояний, в которых оканчиваются переходы из состояния a_i; $B(a_i)$ – множество состояний, переходы из которых оканчиваются в состоянии a_i; $X(a_m, a_i)$ – множество входных переменных, значения которых инициируют переход из состояния a_m в состояние a_i; $X(a_i)$ – множество входных переменных, значения которых инициируют переходы в состояние a_i:

$$X(a_i) = \bigcup_{a_m \in B(a_i)} X(a_m, a_i)a_m, \quad a_i \in A.$$

Чтобы реализовать некоторый переход из состояния a_m в состояние a_i необходимо проверить значение выхода триггера активного состояния a_m (один бит) и значения переменных множества $X(a_m, a_i)$, которые инициируют данный переход. Чтобы реализовать функцию перехода d_i в состояние a_i, необходимо проверить значения выходов триггеров всех состояний, переходы из которых оканчиваются в состоянии a_i, т.е. $|B(a_i)|$ значений, где $|A|$ – мощность множества A. Кроме того, необходимо проверить значения всех входных переменных, которые инициируют переходы в состояние a_i, т.е. $|X(a_i)|$ значений.

Определим ранг функции переходов d_i в состояние a_i как число аргументов этой функции следующим образом:

$$r_i = |B(a_i)| + |X(a_i)|. \tag{1.1}$$

Пусть n – число входов функциональных генераторов LUT. Если ранг r_i некоторой функции переходов d_i, $I = \overline{1, M}$, превысит n входов функционального генератора LUT, то возникает необходимость в декомпозиции функции переходов d_i и ее реализации на нескольких элементах LUT. Отметим, что путем расщепления внутренних состояний нельзя снизить ранг функций переходов ниже величины

$$r^* = \max_{m=\overline{1,M}, s=\overline{1,M}} (|X(a_m, a_s)|) + 1. \tag{1.2}$$

Значение величины r^* в предлагаемом методе используется в качестве верхней границы рангов функций переходов при расщеплении внутренних состояний.

Известно два основных подхода при декомпозиции булевых функций: последовательная (линейная) и параллельная [1]. При последовательной декомпозиции все функциональные генераторы LUT последовательно соединяются в цепочку. На вход первого генератора LUT поступает n аргументов реализуемой функции, а на входы всех остальных элементов LUT – на единицу

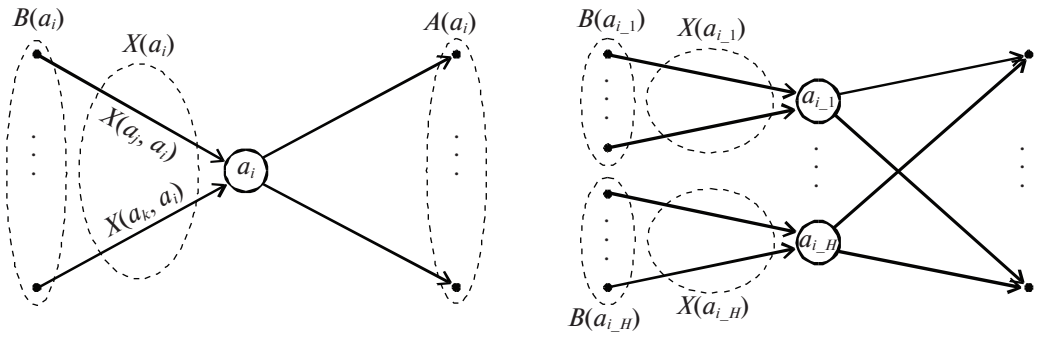

Рис. 1. Расщепление состояния a_i для уменьшения ранга функции переходов d_i

меньше. Число l_i^s уровней функциональных генераторов LUT при реализации функции переходов ранга r_i в случае последовательной декомпозиции определяется из выражения

$$l_i^s = \text{int}\left(\frac{r_i - n}{n-1}\right) + 1, \tag{1.3}$$

где $\text{int}(A)$ – наименьшее целое, большее или равное A.

В случае параллельной декомпозиции функциональные генераторы LUT соединяются в виде иерархической древовидной структуры. Значения аргументов функции поступают на входы элементов LUT первого уровня. На входы всех последующих уровней элементов LUT поступают значения промежуточных функций. Число l_i^p уровней функциональных генераторов LUT при параллельной декомпозиции функции переходов ранга r_i определяется следующим выражением:

$$l_i^p = \text{int}(\log_n r_i). \tag{1.4}$$

Какую декомпозицию, последовательную или параллельную, использует конкретный синтезатор, предсказать сложно. Предварительные исследования показали, что, например, в пакете Quartus Prime одновременно применяется как последовательная, так и параллельная декомпозиция. Число l_i уровней функциональных генераторов LUT при реализации на FPGA функции переходов d_i с рангом r_i может находиться между значениями l_i^s и l_i^p, $i = \overline{1, M}$.

Введем целочисленный коэффициент k, $k \in [0, 10]$, который позволяет адаптировать предлагаемый алгоритм при определении числа уровней функциональных генераторов LUT к конкретному синтезатору. В этом случае число l_i уровней элементов LUT, необходимых для реализации функции переходов d_i ранга r_i, будет определяться следующим выражением:

$$l_i = \left(\frac{l_i^s - l_i^p}{10} k + l_i^p\right). \tag{1.5}$$

Из (1.5) следует, что при $k = 0$ имеем $l_i = l_i^p$, т.е. число уровней соответствует самой быстрой параллельной декомпозиции, а при $k = 10$ имеем $l_i = l_i^s$, т.е. число уровней соответствует самой медленной последовательной декомпозиции. Конкретное значение коэффициента k зависит от архитектуры семейства ПЛИС и используемого синтезатора.

2. Метод синтеза быстрых конечных автоматов. Пусть для некоторой функции переходов d_i ранг r_i превышает значение r^*, т.е. $r_i > r^*$. В этом случае предлагается расщепить состояние a_i на H состояний $a_{i\_1}, \ldots, a_{i\_H}$ (рис. 1) таким образом, чтобы выполнялось $r_{i\_h} \le r^*$ для всех $h = \overline{1, H}$.

Однако имеется опасность расхождения (выполнения до бесконечности) процесса расщепления внутренних состояний. Дело в том, что при расщеплении некоторого состояния a_i, $i = \overline{1, M}$, кроме увеличения числа состояний M, также увеличивается число переходов в состояния множества $A(a_i)$. В результате расщепления состояния a_i для состояний множества $A(a_i)$ увеличива-

ются мощности множеств $B(a_m)$, $a_m \in A(a_i)$. Поэтому для состояний множества $A(a_i)$, согласно (1.1), возрастают ранги функций переходов, что может привести к увеличению значений l_i^s, l_i^p и l_i.

В предлагаемом алгоритме процесс расщепления внутренних состояний прекращается при выполнении условия

$$l_{\max} \le \text{int}(l_{\text{mid}}), \tag{2.1}$$

где $l_{\max}$ – число уровней функциональных генераторов LUT, необходимых для реализации самой "плохой" функции, имеющей максимальный ранг; l_{mid} – среднеарифметическое значение числа уровней функциональных генераторов LUT для всех функций переходов; int(A) – наименьшее целое, большее или равное A. Процесс расщепления внутренних состояний также прекращается, если $l_{\max} > l^*$, где l^* – значение $l_{\max}$ на предыдущей итерации выполнения алгоритма.

С учетом вышеизложенного алгоритм расщепления внутренних состояний для синтеза быстрых конечных автоматов выглядит следующим образом.

Алгоритм 1.

Ш а г 1. Определяется значение коэффициента k, $k \in [0, 10]$, который отражает используемый средством проектирования способ декомпозиции булевых функций.

Ш а г 2. Согласно (1.1) определяются ранги r_i, $i = \overline{1, M}$, функций переходов конечного автомата.

Ш а г 3. На основании (1.3)–(1.5) для каждой функции переходов d_i определяется число l_i уровней функциональных генераторов LUT, необходимых для ее реализации.

Ш а г 4. Определяются значения $l_{\max}$ и l_{mid}. Если выполняются условия (2.1), то идти к шагу 9, иначе – к шагу 5.

Ш а г 5. Выбирается состояние a_i, для которого $r_i = \max$, если таких состояний несколько, среди них выбирается состояние, для которого $|A(a_i)| = \min$ (минимизируется увеличение рангов других состояний в результате расщепления состояния a_i).

Ш а г 6. Полагается $l^* = l_{\max}$.

Ш а г 7. С помощью алгоритма 2 (приведенного в разд. 3) выполняется пробное расщепление состояния a_i, выбранного на шаге 5.

Ш а г 8. Вновь вычисляется $l_{\max}$. Если $l_{\max} > l^*$, то идти к шагу 9; иначе принимается расщепление состояния a_i, тогда идти к шагу 2.

Ш а г 9. Конец.

Дальнейший синтез конечного автомата выполняется традиционными методами, например, автоматически с помощью синтезатора используемого средства проектирования. Для этого достаточно описать конечный автомат, полученный после расщепления внутренних состояний, на одном из языков проектирования (Verilog или VHDL).

Значение коэффициента k, вводимого в шаге 1 алгоритма 1, определяется эмпирически путем синтеза тестовых примеров с помощью применяемого средства проектирования.

3. Расщепление внутренних состояний. Для расщепления некоторого состояния a_i, $i = \overline{1, M}$, выполняемого в шаге 6 алгоритма 1, строится булева матрица W следующим образом. Строки матрицы W соответствуют множеству переходов $C(a_i)$, которые оканчиваются в состоянии a_i. Столбцы матрицы W делятся на две части в соответствии с типами аргументов функции переходов d_i. Первая часть столбцов матрицы W соответствует множеству $B(a_i)$ состояний, переходы из которых оканчиваются в состоянии a_i, а вторая часть – множеству $X(a_i)$ входных переменных, значения которых инициируют переходы в состояние a_i. На пересечении строки t, $t = \overline{1, T}$, $T = |C(a_i)|$, и столбца j первой части матрицы W ставится единица, если переход c_t, $c_t \in C(a_i)$, выполняется из состояния a_j, $a_j \in B(a_i)$. На пересечении строки t и столбца j второй части матрицы W ставится единица, если входная переменная x_j, $x_j \in X(a_i)$, принимает значащее значение (0 или 1) на переходе c_t, $c_t \in C(a_i)$.

Теперь задача сводится к разбиению матрицы W на минимальное число H строчных миноров $W_1, \ldots, W_H$ таким образом, чтобы число столбцов, содержащих единицы, в каждом миноре W_h, $h = \overline{1, H}$, не превышало величину r^*, вычисляемую, согласно (1.2). Состояние a_i разбивается на H состояний $a_{i\_1}, \ldots, a_{i\_H}$, причем строки каждого минора W_h определяют переходы в состояние a_{i_h}, $h = \overline{1, H}$.

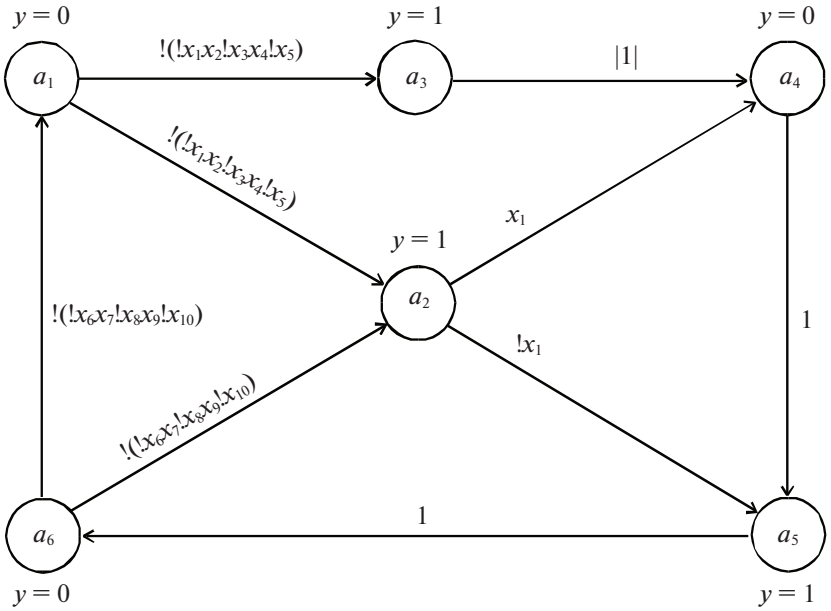

Рис. 2. Граф конечного автомата из примера

Пусть w_t – некоторая строка матрицы W. Для расщепления некоторого состояния a_i, $a_i \in$ A, может быть использован следующий алгоритм.

Алгоритм 2.

Ш а г 1. Строится булева матрица W для расщепления состояния a_i. Полагается $H := 0$.

Ш а г 2. Полагается $H := H + 1$. Начинается формирование минора W_H. В качестве опорной строки в минор W_H выбирается строка w_t с максимальным числом единиц. Строка w_t включается в минор W_H и исключается из дальнейшего рассмотрения, полагается $W_H := \{w_t\}$, $W := W \backslash \{w_t\}$.

Ш а г 3. Добавляются строки в минор W_H. Для этого среди строк матрицы W выбирается строка w_t, для которой выполняется неравенство

$$|W_H \cup \{w_t\}| \le r^*,$$

где $|W_H \cup \{w_t\}|$ – суммарное число единиц в столбцах минора W_H и строки w_t после объединения их по ИЛИ. Если таких строк может быть выбрано несколько, то среди них выбирается строка w_t, имеющая максимальное число общих единиц с минором W_H, т.е.

$$|W_H \cap \{w_t\}| \to \max_t,$$

где $|W_H \cap \{w_t\}|$ – суммарное число единиц в столбцах минора W_H и строки w_t после объединения их по AND. Строка w_t включается в минор W_H и исключается из дальнейшего рассмотрения, полагается $W_H := W_H \cup \{w_t\}$, $W := W \backslash \{w_t\}$.

Шаг 3 повторяется до тех пор, пока в минор W_H может быть включена хотя бы одна строка.

Ш а г 4. Если в матрице W все строки распределены между минорами, то идти к шагу 5, иначе – к шагу 2.

Ш а г 5. Состояние a_i разбивается на H состояний $a_{i\_1}, \ldots, a_{i\_H}$ таким образом, что строки каждого минора W_h определяют переходы в состояние a_{i_h}, $h = \overline{1, H}$.

Ш а г 6. Конец.

Работу предлагаемого метода синтеза продемонстрируем на примере. Пусть необходимо синтезировать быстрый конечный автомат, граф которого показан на рис. 2 (здесь ! обозначает логическое отрицание). Данный автомат представляет собой автомат типа Мура, имеет шесть состояний $a_1, \ldots, a_6$, десять входных переменных $x_1, \ldots, x_{10}$ и одну выходную переменную y, значение которой на рис. 2 показано возле каждого состояния. Переходы из состояний a_3, a_4 и a_5 являются

Таблица 1. Начальные значения величин $B(a_i)$, $X(a_i)$, r_i, l_i^s и l_i^p

Состояние	$B(a_i)$	$X(a_i)$	r_i	l_i^s	l_i^p
a_1	$\{a_6\}$	$\{x_6, x_7, x_8, x_9, x_{10}\}$	6	1	1
a_2	$\{a_1, a_6\}$	$\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$	12	3	2
a_3	$\{a_1\}$	$\{x_1, x_2, x_3, x_4, x_5\}$	6	1	1
a_4	$\{a_2, a_3\}$	$\{x_1\}$	3	1	1
a_5	$\{a_2, a_4\}$	$\{x_1\}$	3	1	1
a_6	$\{a_5\}$	Ø	1	1	1

безусловными, поэтому на этих переходах в качестве условия перехода записано логическое значение 1. Значения множеств $B(a_i)$ и $X(a_i)$, а также ранги r_i функций переходов для данного конечного автомата приведены в табл. 1. Поскольку для нашего примера имеем $\max(|X(a_m, a_s)|) = 5$, то, согласно (1.2), величина $r^* = 6$. Пусть необходимо построить конечный автомат на FPGA, для которой максимальное число входов функциональных генераторов LUT равно 6, т.е. $n = 6$.

Согласно (1.3) и (1.4), определяем значения l_i^s и l_i^p для каждого состояния (приведены в соответствующих столбцах табл. 1). Предположим, что мы не знаем, по какому алгоритму компилятор выполняет декомпозицию булевых функций, поэтому принимаем наихудший вариант: последовательную декомпозицию. Поэтому значение коэффициента k в выражении (1.5) полагается равным 10. В результате число уровней элементов LUT, необходимых для реализации каждой

	a_1	a_6	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
w_1	1	0	1	1	1	1	1	0	0	0	0	0
w_2	0	1	0	0	0	0	0	1	1	1	1	1

Рис. 3. Матрица W для расщепления состояния a_2

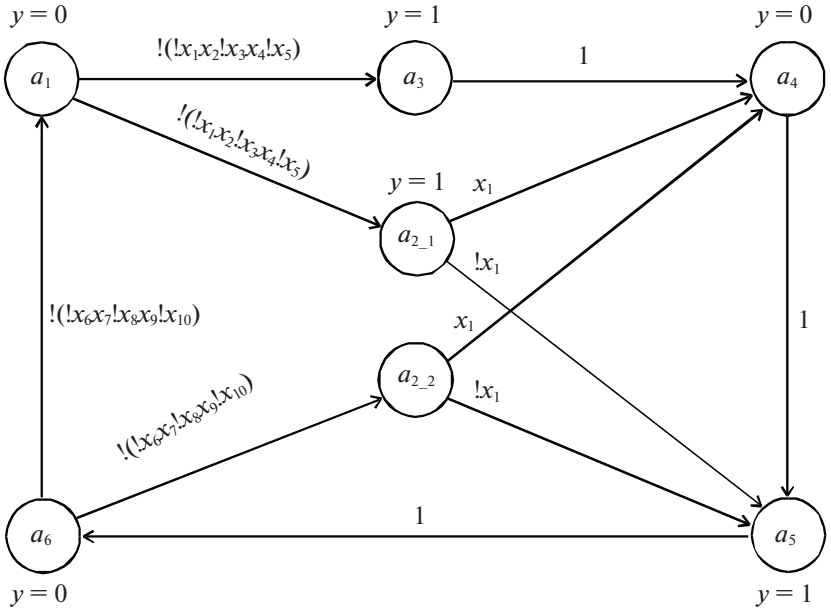

Рис. 4. Граф автомата после расщепления состояния a_2

Таблица 2. Значения величин $B(a_i)$, $X(a_i)$, r_i, l_i^s и l_i^p после расщепления состояния a_2

Состояние	$B(a_i)$	$X(a_i)$	r_i	l_i^s	l_i^p
a_1	$\{a_6\}$	$\{x_6, x_7, x_8, x_9, x_{10}\}$	6	1	1
a_{2_1}	$\{a_1\}$	$\{x_1, x_2, x_3, x_4, x_5\}$	6	1	1
a_{2_2}	$\{a_6\}$	$\{x_6, x_7, x_8, x_9, x_{10}\}$	6	1	1
a_3	$\{a_1\}$	$\{x_1, x_2, x_3, x_4, x_5\}$	6	1	1
a_4	$\{a_{2\_1}, a_3\}$	$\{x_1\}$	3	1	1
a_5	$\{a_{2\_2}, a_4\}$	$\{x_1\}$	3	1	1
a_6	$\{a_5\}$	Ø	1	1	1

Таблица 3. Результаты экспериментальных исследований метода синтеза быстрых конечных автоматов на эталонных примерах

FSM	*Ns*	*M*	*M**	*M*/M*	*P*	*P**	*P*/P*	$l_{\max}$	$l^*_{\max}$	$l_{\max}/l^*_{\max}$
BBSSE	11	16	48	3.00	56	188	3.36	7	3	2.33
CSE	1	16	23	1.44	91	154	1.69	8	5	1.60
EX1	1	18	22	1.22	233	379	1.63	8	5	1.60
EX2	1	19	34	1.79	72	72	1.00	6	2	3.00
EX3	1	10	17	1.70	36	36	1.00	3	2	1.50
EX5	1	9	16	1.78	32	32	1.00	3	2	1.50
EX7	1	10	18	1.80	36	36	1.00	4	2	2.00
KEYB	1	19	26	1.37	170	261	1.54	8	4	2.00
PLANET	2	48	51	1.06	115	120	1.04	3	2	1.50
PMA	8	24	39	1.63	73	106	1.45	5	2	2.50
S208	2	18	43	2.39	153	332	2.17	9	5	1.80
S298	1	218	246	1.13	1096	1236	1.13	27	19	1.42
S386	2	13	18	1.38	64	99	1.55	7	3	2.33
S420	2	18	43	2.39	137	296	2.16	9	5	1.80
S820	3	25	36	1.44	232	324	1.40	14	5	2.80
S832	3	25	38	1.52	245	412	1.68	14	5	2.80
S1488	3	48	63	1.31	251	341	1.36	19	3	6.33
S1494	3	48	63	1.31	250	364	1.46	19	3	6.33
SAND	3	32	39	1.22	184	325	1.77	6	5	1.20
SSE	11	16	48	3.00	56	188	3.36	7	3	2.33
STYR	2	30	41	1.37	166	259	1.56	9	5	1.80
mid	3.00			1.68			1.63			2.40
max	11			3			3.36			6.33

функции переходов, определяется величиной $l_i = l_i^s$. Для нашего примера $\text{int}(l_{\text{mid}}) = \text{int}(8/6) = 2$. Другими словами, процесс расщепления внутренних состояний прекращается, как только каждая функция переходов может быть реализована на двух уровнях элементов LUT.

Для рассматриваемого примера, согласно табл. 1, имеем $l_{\max} = l_2^s = 3$ для состояния a_2, поэтому выполняется расщепление состояния a_2. Строится матрица W для расщепления состояния a_2 (рис. 3). Матрица W содержит две строки. Строка w_1 соответствует переходу из состояния a_1 в состояние a_2, а строка w_2 соответствует переходу из состояния a_6 в состояние a_2. Выполнение

Таблица 4. Результаты реализации эталонных примеров конечных автоматов на FPGA семейств Arria II и Cyclone V

FSM	Arria II						Cyclone V					
	L	*L**	*L*/L*	*F*	*F**	*F*/F*	*L*	*L**	*L*/L*	*F*	*F**	*F*/F*
BBSSE	28	33	1.18	845	937	1.11	21	45	2.14	420	458	1.09
CSE	67	71	1.06	240	370	1.54	43	45	1.05	300	289	0.96
EX1	89	97	1.09	397	352	0.89	54	64	1.19	213	369	1.73
EX2	27	43	1.59	627	787	1.26	19	25	1.32	290	335	1.16
EX3	16	23	1.44	649	629	0.97	12	14	1.17	280	347	1.24
EX5	14	21	1.50	921	587	0.64	10	13	1.30	464	298	0.64
EX7	18	23	1.28	791	587	0.74	12	13	1.08	410	333	0.81
KEYB	50	72	1.44	760	876	1.15	37	50	1.35	360	410	1.14
PLANET	88	93	1.06	1011	1030	1.02	58	63	1.09	475	428	0.90
PMA	86	93	1.08	496	575	1.16	55	59	1.07	209	309	1.48
S208	30	62	2.07	812	806	0.99	19	36	1.89	383	415	1.08
S298	341	360	1.06	556	585	1.05	263	273	1.04	265	258	0.97
S386	33	40	1.21	781	928	1.19	22	29	1.32	414	463	1.12
S420	28	57	2.04	750	810	1.08	18	33	1.83	385	419	1.09
S820	64	80	1.25	636	796	1.25	43	58	1.35	324	371	1.15
S832	65	92	1.42	632	794	1.26	43	61	1.42	306	372	1.22
S1488	153	164	1.07	595	823	1.38	97	111	1.14	296	379	1.28
S1494	145	179	1.23	682	883	1.29	93	119	1.28	309	390	1.26
SAND	127	131	1.03	381	353	0.93	85	90	1.06	255	298	1.17
SSE	28	33	1.18	845	937	1.11	21	45	2.14	420	458	1.09
STYR	117	135	1.15	412	421	1.02	76	87	1.14	288	315	1.09
mid			1.31			1.10			1.35			1.13
max			2.07			1.54			2.14			1.73

алгоритма 2 приводит к разбиению строк матрицы W на два подмножества: $W_1 = \{w_1\}$ и $W_2 = \{w_2\}$. Поэтому состояние a_2 расщепляется на два состояния, как показано на рис. 4. Новые значения величин $B(a_i)$, $X(a_i)$, r_i, l_i^s и l_i^p приведены в табл. 2. Теперь имеем $l_{\max} = l_{\text{mid}} = 1$, поэтому выполнение алгоритма 1 завершается.

Таким образом, для данного примера путем расщепления состояния a_2 удалось снизить максимальное число уровней элементов LUT, необходимых для реализации функций переходов конечного автомата, с 3 до 1 в случае последовательной декомпозиции и с 2 до 1 в случае параллельной декомпозиции.

4. Результаты экспериментальных исследований. Рассмотренный метод синтеза быстрых конечных автоматов исследовался при реализации на FPGA эталонных примеров конечных автоматов, разработанных в центре MCNC [17]. Для этого к каждому эталонному примеру конечного автомата применялся представленный метод синтеза. Оба конечных автомата, исходный и синтезированный, описывались на языке Verilog. Затем выполнялась стандартная реализация на FPGA конечных автоматов с помощью системы Quartus Prime версии 17.1. При этом использовались параметры синтеза системы Quartus, устанавливаемые по умолчанию. В качестве числа входов функциональных генераторов LUT было принято $n = 4$. Предлагаемый метод удалось применить к 21 эталонному примеру из 48, т.е. в 43.75% случаев. Для остальных примеров условия (2.1) не были нарушены и расщепление состояний не выполнялось.

Результаты применения рассмотренного метода синтеза быстрых конечных автоматов приведены в табл. 3, где FSM – имя эталонного примера; *Ns* – число расщеплений внутренних состояний; M и P – число состояний и число переходов конечного автомата до применения метода; M^* и P^* – число состояний и число переходов конечного автомата после применения метода; $l_{\max}$

Таблица 5. Результаты реализации эталонных примеров конечных автоматов на FPGA семейств MAX II и Stratix V

FSM	MAX II						Stratix V					
	L	L^*	L^*/L	F	F^*	F^*/F	L	L^*	L^*/L	F	F^*	F^*/F
BBSSE	41	74	1.80	312	338	1.08	21	26	1.24	1151	1364	1.19
CSE	89	101	1.13	212	240	1.13	43	45	1.05	346	396	1.14
EX1	124	142	1.15	199	241	1.21	54	66	1.22	352	388	1.10
EX2	56	68	1.21	289	338	1.17	19	25	1.32	539	635	1.18
EX3	25	36	1.44	309	359	1.16	12	14	1.17	618	603	0.98
EX5	22	31	1.41	350	414	1.18	10	13	1.30	1294	525	0.41
EX7	25	33	1.32	304	402	1.32	12	13	1.08	1176	582	0.49
KEYB	76	95	1.25	279	324	1.16	37	52	1.41	983	1021	1.04
PLANET	142	144	1.01	402	427	1.06	59	62	1.05	1271	1289	1.01
PMA	124	157	1.27	257	305	1.19	56	67	1.20	220	360	1.64
S208	71	99	1.39	248	313	1.26	19	36	1.89	1131	1215	1.07
S298	903	905	1.00	203	207	1.02	261	270	1.03	656	678	1.03
S386	47	69	1.47	320	339	1.06	22	29	1.32	1139	1172	1.03
S420	31	63	2.03	265	310	1.17	18	33	1.83	1024	1238	1.21
S820	95	126	1.33	255	296	1.16	43	55	1.28	912	1034	1.13
S832	96	132	1.38	234	279	1.19	43	61	1.42	836	1025	1.23
S1488	218	247	1.13	217	323	1.49	98	112	1.14	740	965	1.30
S1494	213	245	1.15	214	326	1.52	93	113	1.22	759	959	1.26
SAND	178	196	1.10	219	253	1.16	85	90	1.06	310	317	1.02
SSE	41	74	1.80	312	338	1.08	21	26	1.24	1151	1364	1.19
STYR	155	188	1.21	214	252	1.18	78	87	1.12	353	347	0.98
mid			1.33			1.19			1.27			1.08
max			2.03			1.52			1.89			1.64

и $l^*_{\max}$ – максимальный ранг функций переходов до и после применения метода; M^*/M, P^*/P *и* $l_{\max}/l^*_{\max}$ – отношения соответствующих параметров; mid – среднее значение параметра; max – максимальное значение параметра.

Анализ табл. 3 показывает, что в синтезированных примерах конечных автоматов среднее число расщеплений составляет 3.00, а максимальное – 11. В результате применения метода число состояний конечных автоматов увеличилось в среднем в 1.68 раза, а максимально – в 3 раза; аналогично число переходов увеличилось в среднем в 1.63 раза, а максимально – в 3.36 раза. Применение данного метода позволило уменьшить максимальный ранг функций переходов в среднем в 2.40 раза, а максимально – в 6.33 раза. Таким образом, несмотря на увеличение числа состояний и числа переходов конечных автоматов, представленный метод позволяет значительно уменьшить максимальный ранг функций переходов. Однако открытым остается вопрос: на сколько увеличилось быстродействие конечных автоматов?

Для ответа на поставленный вопрос была выполнена реализация эталонных примеров на следующих семействах FPGA: Arria II, Cyclone V, MAX II и Stratix V. Результаты исследований приведены в табл. 4 и 5, где L и L^* – число логических элементов FPGA (стоимость реализации), необходимых для реализации исходного и синтезированного конечных автоматов; F и F^* – максимальная частота функционирования исходного и синтезированного конечных автоматов; L^*/L и F^*/F – отношения соответствующих параметров.

Анализ табл. 4 и 5 показывает, что предложенный метод позволяет для различных семейств FPGA увеличить быстродействие конечных автоматов в среднем от 1.08 до 1.19 раза, а максимально – от 1.52 до 1.73 раза (пример EX1 при реализации на FPGA семейства Cyclone V).

Таблица 6. Сравнение предложенного метода с программами JEDI и NOVA при реализации конечных автоматов на FPGA семейств Cyclone V и MAX II

FSM	Cyclone V					MAX II				
	FJ	*FN*	*F**	*F*/FJ*	*F*/FN*	*FJ*	*FN*	*F**	*F*/FJ*	*F*/FN*
BBSSE	402	119	937	2.33	7.87	311	163	937	3.01	5.75
CSE	300	221	370	1.23	1.67	222	161	370	1.67	2.30
EX1	267	156	352	1.32	2.26	217	129	352	1.62	2.73
EX2	333	188	787	2.36	4.19	302	175	787	2.61	4.50
EX3	280	294	629	2.25	2.14	348	219	629	1.81	2.87
EX5	422	298	587	1.39	1.97	340	206	587	1.73	2.85
EX7	398	308	587	1.47	1.91	320	193	587	1.83	3.04
KEYB	360	264	876	2.43	3.32	282	152	876	3.11	5.76
PLANET	435	215	1030	2.37	4.79	391	124	1030	2.63	8.31
PMA	224	198	575	2.57	2.90	267	116	575	2.15	4.96
S386	408	285	928	2.27	3.26	336	192	928	2.76	4.83
SSE	402	119	937	2.33	7.87	311	163	937	3.01	5.75
mid				2.03	3.68				2.33	4.47
max				2.57	7.87				3.11	8.31

Применение метода также увеличивает стоимость реализации конечных автоматов в среднем от 1.27 до 1.35 раза.

В табл. 6 приведено сравнение предложенного метода с известными университетскими программами кодирования внутренних состояний конечных автоматов JEDI [18] и NOVA [19], где *FJ*, *FN* и *F** – максимальная частота функционирования конечного автомата при использовании программы JEDI, NOVA и данного метода соответственно.

Анализ табл. 6 показывает, что с помощью предложенного метода быстродействие конечных автоматов увеличивается в среднем в 2.03–2.33 раза по сравнению с программой JEDI и в 3.68–4.47 раза по сравнению с программой NOVA. При этом максимальное увеличение быстродействия составляет 3.11 раза по сравнению с программой JEDI и 8.31 раза по сравнению с программой NOVA. Такое большое преимущество рассмотренного метода с методами, реализованными в программах JEDI и NOVA, объясняется тем, что данный метод нацелен на увеличение быстродействия, а упомянутые методы – на уменьшение стоимости реализации конечных автоматов.

Заключение. Приведенные результаты экспериментальных исследований показали, что использование представленного метода позволяет значительно снизить максимальный ранг функций переходов, однако увеличение быстродействия конечных автоматов наблюдается не во всех случаях. Это объясняется сложностью задачи синтеза быстрых конечных автоматов. Дело в том, что на быстродействие конечного автомата влияют не только результаты логического синтеза, но также результаты размещения и трассировки. Кроме того, на быстродействие конечных автоматов, кроме функций переходов, также влияет сложность выходных функций.

Предлагаемый метод может также применяться и при построении быстрых конечных автоматов на заказных микросхемах (application specific integrated circuit – ASIC). Для этого достаточно определить оценки числа уровней схемы (1.3)–(1.5) для конкретной архитектуры ASIC.

Дальнейшее развитие методов синтеза быстрых конечных автоматов может идти по пути учета сложности выходных функций, использования специальных структурных моделей конечных автоматов, архитектурных свойств FPGA, специального управления синхронизацией конечного автомата и др.

СПИСОК ЛИТЕРАТУРЫ

1. *Соловьев В.В., Климович А.* Логическое проектирование цифровых систем на основе программируемых логических интегральных схем. М.: Горячая линия – Телеком, 2008.

2. *Miyazaki N., Nakada H., Tsutsui A., Yamada K., Ohta N.* Performance Improvement Technique for Synchronous Circuits Realized as LUT-Based FPGA's // IEEE Trans. on Very Large Scale Integration (VLSI) Systems. 1995. V. 3. № 3. P. 455–459.
3. *Jozwiak L., Slusarczyk A., Chojnacki A.* Fast and Compact Sequential Circuits Through the Information-driven Circuit Synthesis // Proc. Euromicro Symp. on Digital Systems Design. Warsaw. Poland: IEEE, 2001. P. 46–53.
4. *Huang S.-Y.* On Speeding up Extended Finite State Machines Using Catalyst Circuitry // Proc. of the Asia and South Pacific Design Automation Conf. (ASP-DAC). Yokohama. Japan: IEEE, 2001. P. 583–588.
5. *Kuusilinna K., Lahtinen V., Hamalainen T., Saarinen J.* Finite State Machine Encoding for VHDL Synthesis // IEE Proc. Computers and Digital Techniques. 2001. V. 148. № 1. P. 23–30.
6. *Rafla N.I., Davis B.* A Study of Finite State Machine Coding Styles for Implementation in FPGAs // Proc. 49th IEEE Int. Midwest Sympos. on Circuits and Systems. San Juan: IEEE, USA, 2006. V. 1. P. 337–341.
7. *Nedjah N., Mourelle L.* Evolutionary Synthesis of Synchronous Finite State Machines // Proc. Intern. Conf. on Computer Engineering and Systems (ICCES). Cairo. Egypt: IEEE, 2006. P. 19–24.
8. *Czerwiński R., Kania D.* Synthesis Method of High Speed Finite State Machines // Bulletin of the Polish Academy of Sciences: Technical Sciences. 2010. V. 58. № 4. P. 635–644.
9. *Glaser J., Damm M., Haase J., and Grimm C.* TR-FSM: Transition-based Reconfigurable Finite State Machine // ACM Trans. on Reconfigurable Technology and Systems (TRETS). 2011. V. 4. № 3. P. 23:1–23:14.
10. *Senhadji-Navarro R., Garcia-Vargas I.* Finite Virtual State Machines // IEICE Trans. on Information and Systems. 2012. V. E95D. № 10. P. 2544–2547.
11. *Garcia-Vargas I., Senhadji-Navarro R.* Finite State Machines with Input Multiplexing: a Performance Study // IEEE Trans. on CAD. 2015. V. 34. № 5. P. 867–871.
12. *Klimowicz A.* On Using Speed as the Criteria of State Selection for Minimization of Finite State Machines // Eds K. Saeed, W. Homenda. Computer Information Systems and Industrial Management. CISIM 2016. Lecture Notes in Computer Science, Cham: Springer, 2016. V. 9842. P. 493–503.
13. *Klimowicz A.* Performance Targeted Minimization of Incompletely Specified Finite State Machines for Implementation in FPGA Devices // Proc. Euromicro Sympos. on Digital Systems Design. Vienna. Austria: IEEE, 2017. P. 145–150.
14. *Senhadji-Navarro R., Garcia-Vargas I.* High-performance Architecture for Binary-tree-based Finite State Machines // IEEE Trans. on CAD. 2018. V. 37. № 4. P. 796–805.
15. *Salauyou V., Bulatowa I.* Synthesis of High-speed ASM Controllers with Moore Outputs by Introducing Additional States // Eds K. Saeed, W. Homenda. Computer Information Systems and Industrial Management. CISIM 2018. Lecture Notes in Computer Science. Cham: Springer, 2018. V. 11127. P. 405–416.
16. *Solov'ev V.V.* Splitting the Internal States in Order to Reduce the Number of Arguments in Functions of Finite Automata // J. Computer and Systems Sciences International. 2005. V. 44. № 5. P. 777–783.
17. *Yang S.* Logic Synthesis and Optimization Benchmarks user Guide. Version 3.0 // Microelectronics Center of North Carolina (MCNC). North Carolina. USA. MCNC, 1991.
18. *Lin B., Newton A.R.* Synthesis of Multiple Level Logic from Symbolic High-level Description Languages // Proc. IFIP Intern. Conf. on Very Large Scale Integration. Munich. Germany. North-Holland, 1989. P. 187–196.
19. *Villa T., Sangiovanni-Vincentelli A.* Nova: State Assignment of Finite State Machines for Optimal Two-level Logic Implementation // IEEE Trans. on CAD. 1990. V. 9. № 9. P. 905–924.