УДК 004.312.4

Проектирование на программируемых логических интегральных схемах быстрых конечных автоматов

В.В. Соловьев

Белорусская государственная академия связи (Минск, Республика Беларусь), valsol@mail.ru

***Аннотация* — Представлен метод проектирования быстрых конечных автоматов на программируемых логических интегральных схемах (ПЛИС) типа программируемых пользователем вентильных матриц (Field Programmable Gate Arrays – FPGAs). Метод основан на операции расщепления внутренних состояний, что позволяет снизить ранги функций переходов и уменьшить число уровней функциональных генераторов LUT (Look Up Table) при реализации функций переходов. Метод ориентирован на практическое использование, не требует введения дополнительных блоков или синхросигналов и легко может быть включен в общий маршрут проектирования цифровых систем на FPGA. Приводятся оценки числа уровней LUT при реализации функций переходов конечного автомата в случае последовательной и параллельной декомпозиции. Описываются алгоритмы расщепления внутренних состояний. Результаты экспериментальных исследований показали, что метод позволяет увеличить быстродействие конечного автомата на 52%.**

***Ключевые слова* — конечный автомат, высокое быстродействие, расщепление внутренних состояний, программируемые логические интегральные схемы, ПЛИС, FPGA, LUT.**

I. Введение

Производительность проектируемой цифровой системы во многом определяется быстродействием входящих в нее конечных автоматов. 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 (например, памятью или приемопередатчиками).

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 основе функциональных генераторов типа LUT (Look Up Table). ПЛИС первого типа получили название сложные программируемые устройства (Complex Programmable Logic Devices – CPLDs), а второго – программируемые пользователем вентильные матрицы (Field Programmable Gate Arrays – FPGAs). Структуру FPGA можно представить как совокупность большого количества функциональных генераторов LUT, объединяемых межсоединениями. Каждый LUT позволяет реализовать произвольную булеву функцию от небольшого числа аргументов. Методы синтеза быстрых конечных автоматов на CPLD были рассмотрены в [1]. В данной работе рассматривается синтез 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 реализации на FPGA. Метод не вносит изменений в схему устройства, изменяется только расположение регистров. Предложенный подход позволяет увеличить частоту синхронизации и пропускную способность схемы за счет уменьшения задержки сигналов. В [3] представлен метод синтеза последовательностных схем на FPGA. Метод основан на концепции схемного синтеза, управляемого информацией, общей декомпозиции и ранее разработанной теории отношений информационных мер. В [4] предлагается метод оптимизации синхронизации сложных конечных автоматов. Метод основан на концепции катализаторов (catalyst) и добавляет в схему избыточный блок, который включает часть комбинационной схемы и несколько других регистров. В результате критические пути разделяются на стадии. 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] для синтеза конечных автоматов применяются эволюционные методы. На первом этапе решается задача кодирования внутренних состояний с помощью генетических алгоритмов. Затем применяются эволюционные алгоритмы 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, основанная на переходах между состояниями (Transitionbased Reconfigurable FSM - TR-FSM). Новая архитектура позволяет сократить площадь кристалла, задержки сигналов и потребляемую мощность по сравнению с реализацией конечных автоматов на FPGA. В [10] пред-

ложена новая модель автомата, названная виртуальным конечным автоматом (Finite Virtual State Machine - FVSM). Виртуальные конечные автоматы, реализованные на новой архитектуре, имеют преимущество по быстродействию по сравнению с традиционной реализацией конечных автоматов на памяти RAM. В [11] рассматривается реализация конечных автоматов с использованием встроенных блоков памяти FPGA. Предлагаются две архитектуры конечных автоматов с мультиплексорами на входах блоков памяти, которые позволяют уменьшить площадь кристалла и увеличить быстродействие конечного автомата. В [12] 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.

В настоящей 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 и не изменяет алгоритм его функционирования. При расщеплении внутренних состояний сохраняется тип автомата (Мили или Мура), не изменяется общая структура конечного автомата и не используются встроенные блоки памяти FPGA. Поэтому предлагаемый метод синтеза быстрых конечных автоматов на FPGA ориентирован на практическое использование и может быть легко включен в общий маршрут проектирования цифровых систем на FPGA.

II. Суть предлагаемого подхода

Конечный автомат будем характеризовать числом M внутренних состояний множества $A = \{a_1,\ldots,a_M\}$, числом L входных переменных множества $X = \{x_1,\ldots,x_L\}$, числом N выходных переменных множества $Y = \{y_1,\ldots,y_N\}$, 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 1 означает, 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 $X(a_m, a_i)$ – множество входных переменных, значения которых инициируют переход из состояния a_m в состояние a_i, $a_m, a_i \in A$. 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 $B(a_i)$ – множество состояний, переходы из которых оканчиваются в состоянии a_i, $|A|$ – мощность множества A. Кроме того, необходимо проверить значения всех входных переменных, которые инициируют переходы в состояние a_i, т.е. $|X(a_i)|$ значений, где $X(a_i)$ – множество входных переменных, значения которых инициируют переходы в состояние a_i, $X(a_i) = \bigcup_{a_m \in B(a_i)} X(a_m, a_i)$.

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

$$r_i = |B(a_i)| + |X(a_i)|. \quad (1)$$

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 генераторах LUT.

Отметим, что путем расщепления внутренних состояний нельзя снизить ранг функций переходов ниже величины

$$r^* = \max(|X(a_m, a_s)|) + 1, m, s = \overline{1,M}. \quad (2)$$

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

Известны два основных подхода при декомпозиции булевых функций: последовательная (линейная) и параллельная [1]. При последовательной декомпозиции все функциональные генераторы LUT последовательно соединяются в цепочку (рис. 1).

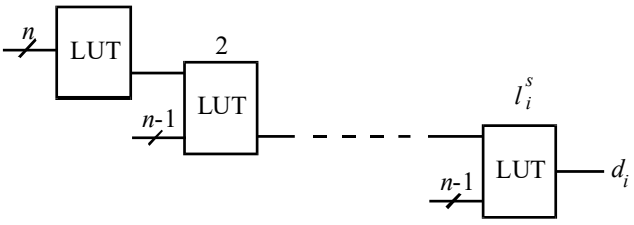

Рис. 1. Последовательная декомпозиция булевых функций

На вход первого генератора LUT поступает n аргументов реализуемой функции, а на входы всех остальных генераторов LUT – на единицу меньше. Число l_i^s уровней функциональных генераторов LUT при реализации функции переходов ранга r_i в случае последовательной декомпозиции определяется из выражения:

$$l_i^s = \mathrm{int}\left(\frac{r_i - n}{n-1}\right) + 1, \qquad (3)$$

где int(A) – наименьшее целое, большее или равное A.

В случае параллельной декомпозиции функциональные генераторы LUT соединяются в виде иерархической древовидной структуры (рис. 2).

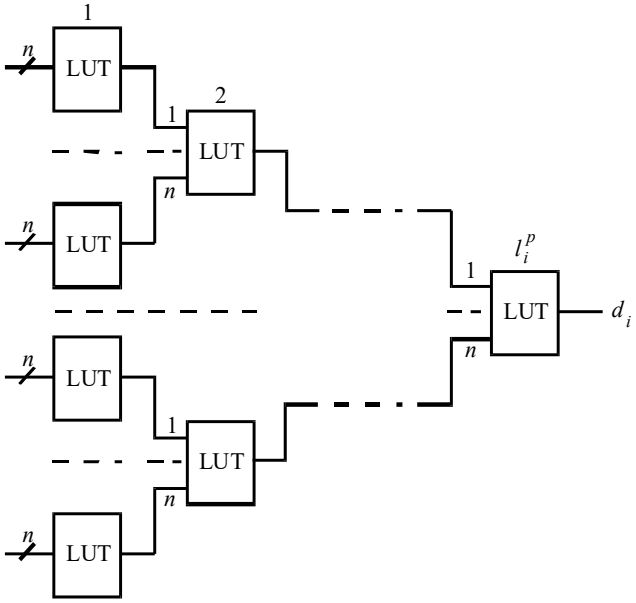

Рис. 2. Параллельная декомпозиция булевых функций

Значения аргументов функции поступают на входы генераторов LUT первого уровня. На входы всех последующих уровней генераторов LUT поступают значения промежуточных функций. Число l_i^p уровней функциональных генераторов LUT при параллельной декомпозиции функции переходов ранга r_i определяется следующим выражением:

$$l_i^p = \mathrm{int}\left(\log_n r_i\right). \qquad (4)$$

Какую декомпозицию, последовательную или параллельную, использует конкретный синтезатор, предсказать сложно. Предварительные исследования показали, что, например, в пакете Quartus II фирмы Altera одновременно используется 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 генераторов LUT, необходимых для реализации функции переходов d_i ранга r_i, будет определяться следующим выражением:

$$l_i = \mathrm{int}\left(\frac{10-k}{10} l_i^p + \frac{k}{10} l_i^s\right). \qquad (5)$$

Из выражения (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.

Следующей проблемой в предлагаемом подходе является ответ на вопрос: когда следует остановить процесс расщепления внутренних состояний синтезируемого конечного автомата? 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 $A(a_i)$, где $A(a_i)$ – множество состояний, в которых оканчиваются переходы из состояния a_i. В результате расщепления состояния a_i для состояний множества $A(a_i)$ увеличиваются мощности множеств $B(a_m)$, $a_m \in A(a_i)$. Поэтому для состояний множества $A(a_i)$, согласно (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 следующего условия:

$$l_{\max} \le \mathrm{int}\left(l_{mid}\right), \qquad (6)$$

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. Отметим, что в процессе расщепления внутренних состояний l_{mid} будет увеличиваться, а l_{max} – уменьшаться, поэтому выполнение алгоритма всегда заканчивается.

III. Метод синтеза быстрых конечных автоматов

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

Алгоритм 1.

1. Определяется значение коэффициента k, $k \in [0, 10]$, который отражает используемый синтезатором пакета проектирования способ декомпозиции булевых функций.
2. Согласно (1) определяются ранги r_i функций переходов конечного автомата, $i=\overline{1,M}$.
3. На основании (3), (4) и (5) для каждой функции переходов d_i определяется число l_i уровней функцио-

нальных генераторов LUT, необходимых для ее реализации.

4. Определяются значения l_{max} и l_{mid}. Если выполняются условия (6), то идти к пункту 7, иначе – к пункту 5.
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).
6. С помощью алгоритма 2 выполняется расщепление состояния a_i, выбранного в пункте 5, на минимальное число H состояний $a_{i\_1},\ldots,a_{i\_H}$ таким образом, чтобы для каждого состояния a_{i_h}, $h=\overline{1,H}$, выполнялось $r_{i\_h} \le r^*$, где r^* определяется согласно (2). Идти к пункту 2.
7. Конец.

Дальнейший синтез конечного автомата выполняется традиционными методами, например, автоматически с помощью синтезатора используемого пакета 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 пункте 1 алгоритма 1, определяется эмпирически путем синтеза тестовых примеров с помощью используемого пакета проектирования.

Для расщепления некоторого состояния a_i, $i=\overline{1,M}$, выполняемого в пункте 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^*, определяемую согласно (2). Строки каждого минора W_h будут определять переходы в вводимое состояние a_{i_h}, $h=\overline{1,H}$.

Пусть w_t – некоторая строка матрицы W. Для нахождения строчного разбиения матрицы W на минимальное число H строчных миноров $W_1,\ldots,W_H$ может быть использован следующий алгоритм.

Алгоритм 2.

1. Полагается $h := 0$.
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 $W_h := \{w_t\}$, $W := W\backslash\{w_t\}$.
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 строке 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. $|W_h \cap \{w_t\}| = \max$, где $|W_h \cap \{w_t\}|$ – суммарное число единиц в столбцах минора W_h и строке w_t после объединения их по И. Строка w_t включается в минор W_h и исключается из дальнейшего рассмотрения, полагается $W_h := W_h \cup \{w_t\}$, $W := W\backslash\{w_t\}$. Пункт 3 повторяется до тех пор, пока в минор W_h может быть включена хотя бы одна строка.
4. Если в матрице W все строки распределены между минорами, то идти к пункту 5, иначе – к пункту 2.
5. Конец.

Работу предлагаемого метода синтеза продемонстрируем на примере. Пусть необходимо синтезировать быстрый конечный автомат, граф которого показан на рис. 3.

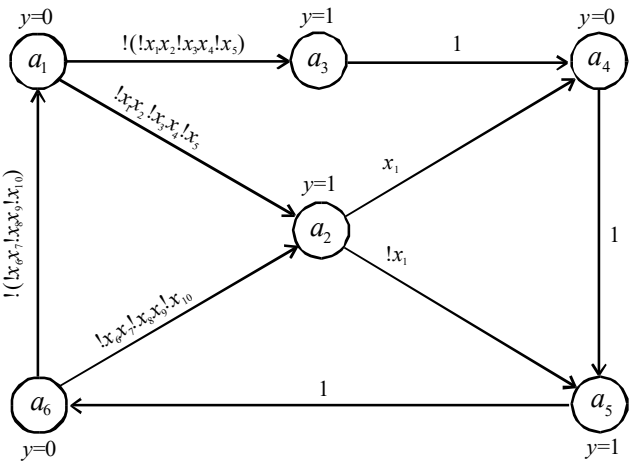

Рис. 3. Граф конечного автомата из примера

Данный автомат представляет собой автомат типа Мура, имеет 6 состояний $a_1,\ldots,a_6$, 10 входных переменных $x_1,\ldots,x_{10}$ и одну выходную переменную y, значение которой на рис. 3 показано возле каждого со-

стояния. Переходы из состояний a_3, a_4 и a_5 являются 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 данного примера имеем $\max(|X(a_m,a_s)|) = 5$, то согласно (2), величина $r^* = 6$. Пусть необходимо построить конечный автомат на FPGA, для которого максимальное число входов функциональных генераторов LUT равно 6, т.е. имеем $n = 6$.

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

Согласно (3) и (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 (5) полагается равным 10, т.е. имеем $k = 10$. В результате число уровней генераторов LUT, необходимых для реализации каждой функции переходов, определяется величиной $l_i = l_i^s$. Для нашего примера имеем $\text{int}(l_{mid}) = \text{int}(8/6) = 2$. Другими словами, процесс расщепления внутренних состояний прекращается, как только каждая функция переходов может быть реализована на двух уровнях генераторов LUT.

Для рассматриваемого примера имеем $l_{max} = l_2^s = 3$, т.е. условие (9) нарушено для состояния a_2, поскольку $l_{max} = l_2^s = 3 > \text{int}(l_{mid}) = 2$. Поэтому выполняется расщепление состояния a_2 с помощью алгоритма 2. Строится матрица W для расщепления состояния a_2 (рис. 4).

	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. 4. Матрица W для расщепления состояния a_2

Матрица W на рис. 4 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 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. 5. Новые значения величин $B(a_i)$, $X(a_i)$, r_i, l_i^s и l_i^p приведены в табл. 2. Теперь имеем $l_{max} = l_{mid} = 1$, поэтому выполнение алгоритма 1 завершается.

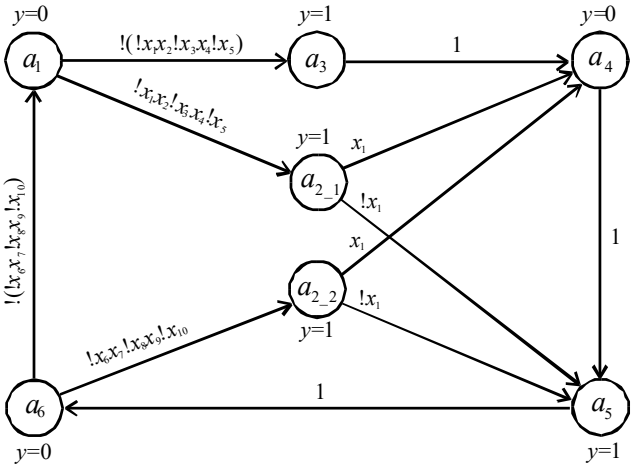

Рис. 5. Граф автомата после расщепления состояния a_2

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

Таким образом, для данного примера путем расщепления состояния a_2 удалось снизить максимальное число уровней генераторов 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.

IV. Результаты экспериментальных исследований

Эффективность предлагаемого метода синтеза быстрых конечных автоматов проверялась при реализации конечного автомата, представленного на рис. 3 и рис. 5 на ПЛИС фирмы Altera с помощью пакета автоматизированного проектирования Quartus II версии 15.0. В качестве основного критерия оптимизации был выбран параметр «speed» (быстродействие), в качестве способа кодирования внутренних состояний исходного конечного автомата был выбран «one-hot» (унарное кодирование), а после синтеза – «user» (коды состояний определяются из описания конечного автомата).

В табл. 3 представлены результаты экспериментальных исследований рассматриваемого метода синтеза быстрых конечных автоматов для различных семейств ПЛИС, где $nLUT_1$ и $nLUT_2$ – число используемых функциональных генераторов LUT при реализации исходного конечного автомата и синтезированного конечного автомата соответственно; F_1 и F_2 – максимальная частота функционирования (в MHz) для исходного и синтезированного конечного автомата соответственно; F_1/F_2 – отношение соответствующих параметров.

Таблица 3

Результаты экспериментальных исследований

Семейство	$nLUT_1$	F_1	$nLUT_2$	F_2	F_2/F_1
Arria II GX	8	1307	7	1269	0.97
Cyclone IV E	9	778	10	793	1.02
Cyclone IV GX	9	729	10	802	1.10
Cyclone V	6	686	6	925	1.35
MAX 10	10	800	11	816	1.02
MAX V	10	343	9	314	0.92
MAX II	10	389	9	593	1.52

Анализ табл. 3 показывает, что быстродействие конечного автомата в результате применения предлагаемого метода увеличилась для 5 семейств из 7. При этом для семейства MAX II быстродействие увеличилось в 1.52 раза (или на 52%), а для семейства Cyclone V – в 1.35 раза (или на 35%). Кроме того, для семейств Arria II GX, MAX V и MAX II в результате применения рассматриваемого метода синтеза наблюдается снижение стоимости реализации.

Заключение

Приведенные результаты экспериментальных исследований показали следующее. Несмотря на то, что в рассматриваемом примере был уменьшен максимальный ранг функций переходов с 12 до 6, что позволило снизить число уровней LUT с 3 до 1 в случае последовательной декомпозиции и с 2 до 1 в случае параллельной декомпозиции, быстродействие конечного автомата увеличилось не для всех семейств FPGA. Это объясняется сложностью задачи синтеза быстрых конечных автоматов, например, по сравнению с задачей уменьшения стоимости реализации. Дело в том, что на быстродействие конечного автомата влияют не только результаты логического синтеза, но также результаты размещения и трассировки.

Снижение стоимости реализации для некоторых семейств FPGA в результате применения данного метода объясняется просто: в результате уменьшения числа уровней функциональных генераторов LUT, необходимых для реализации функций переходов, также уменьшается и количество генераторов LUT.

Отметим, что предлагаемый метод может также применяться и при построении быстрых конечных автоматов на микросхемах ASIC. Для этого достаточно определить оценки числа уровней схемы (3) и (4) для конкретной архитектуры ASIC.

Дальнейшее развитие методов синтеза быстрых конечных автоматов может идти по пути использования специальных структурных моделей конечных автоматов, архитектурных свойств FPGA, специального управления сигналом синхронизации памяти конечного автомата и др.

Работа выполнена при частичной финансовой поддержке Белостокского технического университета (Польша), грант S/WI/1/2013.

Литература

[1] Соловьев В.В., Климович А. Логическое проектирование цифровых систем на основе программируемых логических интегральных схем. М.: Горячая линия – Телеком. 2008. 376 с.

[2] Miyazaki N., Nakada H., Tsutsui A., Yamada K., Ohta N. Performance Improvement Technique for Synchronous Circuits Realized as LUT-Based FPGA's // IEEE Transactions on Very Large Scale Integration (VLSI) systems, V. 3. № 3. 1995. P. 455-459.

[3] Jozwiak L, Slusarczyk A, Chojnacki A. Fast and compact sequential circuits through the information-driven circuit synthesis // Proc. of the Euromicro Symposium on Digital Systems Design. Warsaw. Poland. 4-6 September 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. (ASAP-DAC). Yokohama. Jan.-Feb. 2001. P. 583-588.

[5] Kuusilinna K., Lahtinen V., Hamalainen T., Saarinen J. Finite state machine encoding for VHDL synthesis // Computers and Digital Techniques. IEE Proceedings. 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. of the 49th IEEE International Midwest Symposium on Circuits and Systems. San Juan. USA. 6-9 Aug. 2006. V. 1. P. 337-341.

[7] Nedjah N., Mourelle L. Evolutionary synthesis of synchronous finite state machines // Proc. of the Int. Conf. on Computer Engineering and Systems. Cairo. Egypt. 5-7 Nov. 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 Transactions on Reconfigurable Technology and Systems (TRETS). Aug. 2011. V. 4, № 3, P. 23:1-23:14.

[10] Senhadji-Navarro R., Garcia-Vargas I. Finite virtual state machines // IEICE Transactions 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 Transaction on computer-aided design of integrated circuits and systems. 2015. V. 34. № 5. P. 867-871.

[12] Соловьев В.В. Расщепление внутренних состояний для снижения числа аргументов функций конечных автоматов // Изв. Российской академии наук. Теория и системы управления. 2005. № 5. С. 113-119.

Designing on FPGA of high-speed finite state machines

V.V. Salauyou

Belarusian State Academy of Telecommunications (Minsk, Republic of Belarus), valsol@mail.ru

***Keywords* — finite state machine, high-speed, state splitting, field programmable gate array, FPGA, SoC, look up table, LUT.**

ABSTRACT

A design method of high-speed finite state machines (FSMs) on LUT-based field programmable gate array (FPGA) by internal state splitting is offered. The method is aimed for the practical usage. The method does not change FSM's type (Mealy or Moore) [1], the method does not demand introduction of additional blocks or clock signals, and it can be easily included in a designing flow of digital systems on FPGA.

Many articles are devoted to problems of high-speed FSM design on FPGA. The paper [2] presents a technique for improving the performance of a synchronous circuit configured as a look-up table based FPGA without changing the initial circuit configuration; only the register location is altered. It improves clock speed and data throughput at the expense of latency. The article [3] present new methods and tools for state encoding and combinational synthesis of sequential circuits based on new criteria of information flow optimization. Together they form a unified and complete pre-placement synthesis chain. In [4] the method of optimization of synchronization of difficult finite state machines is offered. The method is based on the concept of catalytic agents (catalyst) and adds the excess unit which includes part of the combinative circuit and several other registers to the circuit. As result, critical paths are divided on stages. The paper [4] proposes a timing optimization technique for a complex FSM that consists of not only random logic but data operators also. The proposed technique, based on the concept of catalyst, adds a functionally redundant block (which includes a piece of combinational logic and several other registers) to the circuits under consideration so that the timing critical paths are divided on stages. In [5,6] styles of the FSM description in VHDL language and also the known methods of encoding of internal states for implementation of the fast FSMs are researched. The work [7] proposes usage of the evolutionary methodology to yield optimal evolvable hardware that implements the state machine control component. The evolved hardware requires a minimal hardware area and introduces a minimal propagation delay of the machine output signals. In [8], the task of FSM state assignment and optimization of the combinational circuit in case of implementation on CPLD of fast FSMs is considered. In [9], the new architecture of programmed logic which is specially intended for the implementation of FSMs based on transitions between statuses is provided (Transition-based Reconfigurable FSM - TR-FSM). The new architecture allows reduction of the chip area, time signal delay and power consumption in comparison with the implementation of FSMs on FPGA. The paper [10] presents a study of performance of RAM-based implementations in FPGAs of FSMs. The influence of the FSM characteristics on speed and area has been studied, taking into account the particular features of different FPGA families, like the size of LUTs, the size of memory blocks, the number of embedded multiplexer levels and the specific decoding logic for distributed RAM. In [11], the implementation of FSMs with the use of internal FPGA memory blocks is considered. Two architectures of the finite state machines with multiplexers on inputs of memory blocks, allowing reduction of the chip area and increasing high-speed performance of the FSM are offered. In [12], the way to decrease the number of arguments of FSM transition functions by decomposition of internal states of the FSM was considered. It is promotes both implementation reduction in cost, and increase of high-speed performance of FSMs in case of their implementation on FPGA.

Let $A = \{a_1, \ldots, a_M\}$ be the set of internal states, $X = \{x_1, \ldots, x_L\}$ be the set of input variables, $Y = \{y_1, \ldots, y_N\}$ be the set of output variables, and $D = \{d_1, \ldots, d_R\}$ be the set of transition functions of FSM.

A one-hot state assignment is traditionally used for synthesis of the high-speed FSMs on FPGAs. Thus, to each internal state corresponds the separate flip-flop of FSM's memory, which setting in 1 means that the FSM is in the given state. The input of each flip-flop is controlled by transition function d_i, $d_i \in D$, i.e. to each internal state of the FSM corresponds own transition function d_i.

Let $X(a_m,a_i)$ be the set of the FSM input variables which values initiate the transition from the state a_m to the state a_i (a_m, $a_i \in A$). To implement some transition from the state a_m to the state a_i, it is necessary to check the value of the flip-flop output for the active state a_m (one bit) and the input variable values of the set $X(a_m,a_i)$, which initiates the given transition. To implement the transition function d_i, it is necessary to check the values of flip-flop outputs for all states from which transitions lead to the state a_i, i.e. $|B(a_i)|$ values, where $B(a_i)$ is the set of states from which transitions terminate in the state a_i, where $|A|$ is the capacity of the set A. Besides, it is necessary to check the values of all input variables which initiate transitions to the state a_i, i.e. $|X(a_i)|$ values, where $X(a_i)$ is the set of the input variables which values initiate transitions to the state a_i.

Let r_i (1) be a rank of the transition function d_i. Let n be a number of inputs of the functional generators LUT. If the rank r_i for the some transition function d_i exceeds n inputs there is a necessity in a decomposition of the transi-

tion function d_i and its implementations on the several LUTs.

Note, that by splitting of internal states, it is impossible to lower the rank of the transition functions below the value r^* (2). In the proposed method, the value r^* is used as an upper bound of the ranks of the transition functions at splitting of FSM internal states.

Two basic approaches at the decomposition of Boolean functions are well-known: sequential and parallel [1]. At the sequential decomposition, all functional generator LUTs sequentially are connected in a chain. The n arguments of the function d_i arrive on inputs of the first LUT, and the (n-*1*) arguments arrive on inputs of all remaining LUTs. The number l_i^s of the LUT's levels in case the sequential decomposition of the transition function d_i having the rank r_i is defined from expression (3).

In case of parallel decomposition, the functional generator LUTs incorporate in the form of the hierarchical tree structure. The values of function arguments arrive on inputs of the first level LUTs, and the values of the intermediate functions arrive on inputs of all next levels of the LUTs. Because the number of the LUT's levels in case the parallel decomposition the transition function d_i having the rank r_i is defined by expression (4)

It is difficult to predict what decomposition (sequential or parallel) is used by the specific synthesizer. The preliminary researches have showed that, for example, in packet Quartus II from Altera, both sequential and parallel decomposition are used simultaneously. The number l_i levels of the functional generators LUT at implementation on FPGA transition function d_i with the rank r_i can is between values l_i^s and l_i^p .

Let us enter integer coefficient k, $k \in [0,10]$, which allows adaptation of the proposed algorithm at determination of the number of the LUT's levels for the specific synthesizer. In this case, the number l_i of the LUT's levels for implementation of the transition function d_i having the rank r_i will be defined by expression (5).

The next problem of the offered approach is to find an answer to the question: when it is necessary to stop splitting of the FSM states? The matter is that at splitting of the some state a_i except the increase of the number M of the FSM states, the number of the transitions in the states of a set $A(a_i)$ is also grown, where $A(a_i)$ is the set of the states in which transitions from the state a_i terminates. At splitting of the state a_i, the capacities of the sets $B(a_m)$, $a_m \in A(a_i)$, are increased for the states of set $A(a_i)$. Therefore, according to (1), for states of set $A(a_i)$, the ranks of the transition functions grow what can lead to the increase of the values and l_i^s , l_i^p , and l_i.

In the proposed algorithm, the process of state splitting is stopped when the following condition is performed (6), where l_{max} is the number of the LUT levels which necessary for implementation of the most "bad" function having the maximum rank; l_{mid} is the arithmetic mean value of number of the LUT levels for all transition functions. Note that the value l_{mid} will increase in the splitting process of internal states, and the value l_{max} will decrease, therefore the algorithm execution always comes to the end.

The analysis of experimental researches shows that high-speed performance of the FSM, as the result of application of the offered method, is increased for 5 families from 7. Thus, for the family MAX II, the high-speed performance is increased at 1.52 time (or by 52 %), and for the family Cyclone V, the high-speed performance is increased at 1.35 time (or on 35 %).

REFERENCES

[1] Salauyou V.V., Klimowicz A. Logic design of digital systems on programmable logic devices – Moscow, Hot Line – Telecom Publ., 2008. 376 p. (in Russian)

[2] Miyazaki N., Nakada H., Tsutsui A., Yamada K., Ohta N. Performance Improvement Technique for Synchronous Circuits Realized as LUT-Based FPGA's. IEEE Transactions on Very Large Scale Integration (VLSI) systems, vol. 3. no. 3. 1995. pp. 455-459.

[3] Jozwiak L, Slusarczyk A, Chojnacki A. Fast and compact sequential circuits through the information-driven circuit synthesis - Proc. of the Euromicro Symposium on Digital Systems Design. Warsaw. Poland. 4-6 September 2001. pp. 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. (ASAP-DAC). Yokohama. Jan.-Feb. 2001. pp. 583-588.

[5] Kuusilinna K., Lahtinen VOL., Hamalainen T., Saarinen J. Finite state machine encoding for VHDL synthesis. Computers and Digital Techniques. IEE Proceedings. 2001. vol. 148. no. 1. pp. 23-30.

[6] Rafla N. I., Davis B. A Study of finite state machine coding styles for implementation in FPGAs - Proc. of the 49th IEEE International Midwest Symposium on Circuits and Systems. San Juan. USA. 6-9 Aug. 2006. vol. 1. pp. 337-341.

[7] Nedjah N., Mourelle L. Evolutionary synthesis of synchronous finite state machines - Proc. of the Int. Conf. on Computer Engineering and Systems. Cairo. Egypt. 5-7 Nov. 2006. pp. 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. vol. 58, no. 4, pp. 635–644.

[9] Glaser J., Damm M., Haase J., and Grimm C. TR-FSM: Transition-based Reconfigurable Finite State Machine. ACM Transactions on Reconfigurable Technology and Systems (TRETS). 2011. vol. 4, no. 3, P. 23:1-23:14.

[10] Senhadji-Navarro R., Garcia-Vargas I. Finite virtual state machines. IEICE Transactions on information and systems. 2012. vol. E95D. no. 10. pp. 2544-2547.

[11] Garcia-Vargas I., Senhadji-Navarro R. Finite state machines with input multiplexing: a performance study. IEEE Transaction on computer-aided design of integrated circuits and systems. 2015. vol. 34. no. 5. pp. 867-871.

[12] Solov'ev V.V. Splitting the Internal States in Order to Reduce the Number of Arguments in Functions of Finite Automata. Journal of Computer and Systems Sciences International, vol. 44, no. 5, 2005, pp. 777-783.