ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

УДК 681.5

В.В. Кондратьев[1], К.М. Михайлов[2]

РЕКОНФИГУРИРОВАНИЕ КОНЕЧНЫХ АВТОМАТОВ

Нижегородский государственный технический университет им. Р.Е. Алексеева[1],
Мера НН[2]

Вводится понятие реконфигурируемых конечных автоматов как формальной модели для описания автоматов, конфигурация которых может быть изменена во время работы. С появлением устройств реконфигурируемой логики эта модель становится важной для описания и создания реконфигурирующегося оборудования. Предлагаются алгоритмические решения и эффективная аппаратная реализация процесса преобразования существующего конечного автомата в другой конечный автомат.

Ключевые слова: реконфигурируемый конечный автомат, реконфигурирующийся конечный автомат, события, состояния, реконфигуратор.

Постановка задачи

С появлением реконфигурируемого и частично реконфигурируемого оборудования, такого как FPGA (field programmable gate arrays) [1, 2], стало возможным оперативно изменять структуру оборудования. Большинство из этих микросхем основаны на статическом ОЗУ, и реконфигурация производится путем изменения данных в строках или столбцах конфигурируемых логических блоков (КЛБ).

Время реконфигурации этих микросхем порядка нескольких миллисекунд, что значительно быстрее, чем алгоритмы размещения конечных автоматов в реконфигурационые данные.

Для сокращения времени реконфигурации предварительно синтезированные конфигурационные данные перезаписываются или просто подменяют друг друга в процессе выполнения с использованием таких технологий, как мультиконтекстные FPGA.

Известно, что первые поколения микропроцессоров и операционных систем использовали идею самореконфигурируемого программного кода для того, чтобы выполнять более сложные или большие по размеру программы, которые не уместились бы в оперативной памяти или работали бы слишком медленно при классическом подходе. До сих пор подобный подход используется в системах динамического сжатия программного кода, и некоторых системах защиты от несанкционированного использования программного обеспечения.

Представляет значительный интерес использование этого принципа в контексте оборудования – создание схем, которые изменяют свою структуру во время работы.

Для этого необходимо создание:

- высокоуровневых моделей для описания самореконфигурируемого оборудования, таких как реконфигурируемые конечные автоматы;
- способа использования расширенных возможностей предоставляемых реконфигурацией, вместо простой замены одной цельной конфигурации другой.

Концепция реализации реконфигурируемого автомата

Назовем конечный автомат, в котором возможно изменять функцию выходов (G) и функцию переходов (F) *реконфигурируемым конечным автоматом*.

Автомат, реконфигурация которого инициируется самой системой, будем называть *реконфигурирующимся конечным автоматом*.

Внешние события, инициирующие изменение F и G, называются *реконфигурационными событиями*.

Реконфигурируемый конечный автомат описывается десятью элементами.
$(S, I, O, F, G, S_0, R, H_f\ H_g, H_i)$, где

- S – конечный набор внутренних состояний автомата;
- I – конечный набор входных сигналов, представленный либо символически, либо как бинарный вектор;
- O – конечный набор выходных сигналов, представленный либо символически, либо как бинарный вектор;
- $F(i,s)$ – функция переходов (зависимость следующего состояния от текущего и от входных сигналов);
- $G(i,s)$ – функция выходов (зависимость выходного сигнала от текущего и от входных сигналов);
- S_0 – набор начальных состояний;
- R – конечный набор состояний реконфигурации конечного автомата, представленный либо символически, либо как бинарный вектор входых сигналов реконфигурации;
- $H_f(r)$ – реконфигуратор функции переходов, где $r \in R$ – отображение реконфигурационного состояния во внутреннее состояние конечного автомата $F(i',s)$, где $i' \in I$, $s \in S$, т.е. $F(i',s)$ может быть реконфигурирована следующим образом: $F(i',s) := H_f(r)$;
- $H_g(r)$ – реконфигуратор функции переходов, где $r \in R$ – отображение реконфигурационного состояния в выходное состояние конечного автомата $G(i',s)$, где $i' \in I$, $s \in S$, т.е. $G(i',s)$ может быть реконфигурирована следующим образом: $G(i',s) := H_g(r)$;
- $H_i(i,r)$, где $r \in R,\ i \in I$ – отображение реконфигурационного состояния во входные сигналы.

Описанная модель реконфигурируемого конечного автомата представлена на рис. 1. Функция переходов F может быть изменена при помощи H_f, а функция выходов G – при помощи H_g. Функции H_g и H_f зависят от реконфигурационного состояния, в то время как H_i зависит от входных сигналов и реконфигурационного состояния. Функция H_i определена таким образом, что во время работы конечного автомата $I' = I$, а во время реконфигурации I' зависит только от *r*. Физические аспекты – когда и как могут изменяться функции выходов и переходов, будут рассмотрены далее.

Для реализации предложенного подхода выбираем вариант конечного автомата, в котором блоки, выполняющие функции *F* и *G* реализованы на основе блоков ОЗУ.

Таким образом, конфигурация автомата задается содержимым ОЗУ. В эту схему добавляется блок реконфигуратора, который может изменять содержимое ОЗУ (*F* и *G*). Кроме того, этот блок может генерировать входные сигналы, необходимые для перевода автомата в конкретное состояние в соответствии с программой реконфигурации.

При наступлении реконфигурационного события автомат переводится в режим реконфигурации. В режиме реконфигурации блок реконфигуратора в соответствии с програм-

мой реконфигурации генерирует входные сигналы, необходимые для формирования адреса в блоках ОЗУ, и подает на вход ОЗУ значения, необходимые для реконфигурации.

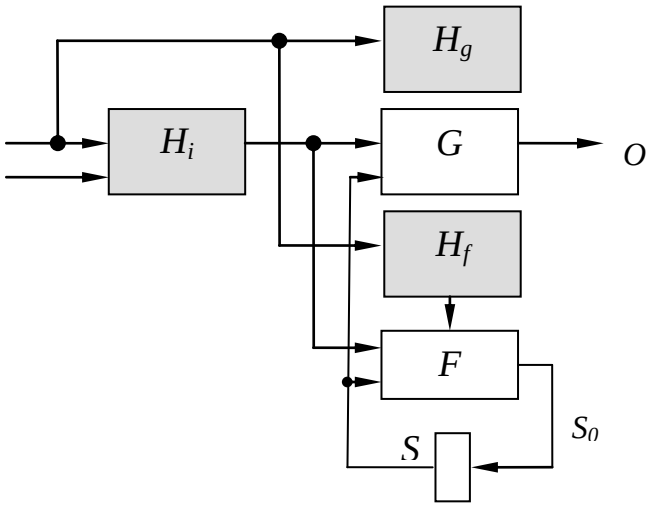

Рис. 1

На рис. 2 изображена блок-схема аппаратной реализации реконфигурируемого конечного автомата. В данной реализации блок реконфигуратора реализует функции H_g, H_f, H_i и генерирует два дополнительных сигнала – rst и rst-st, назначение которых объяснено далее. ***F*-ОЗУ** и ***G*-ОЗУ** – это память, которая содержит реконфигурируемые функцию переходов и функцию выходов $F(i',s), G(i',s), i' \in I, s \in S$.

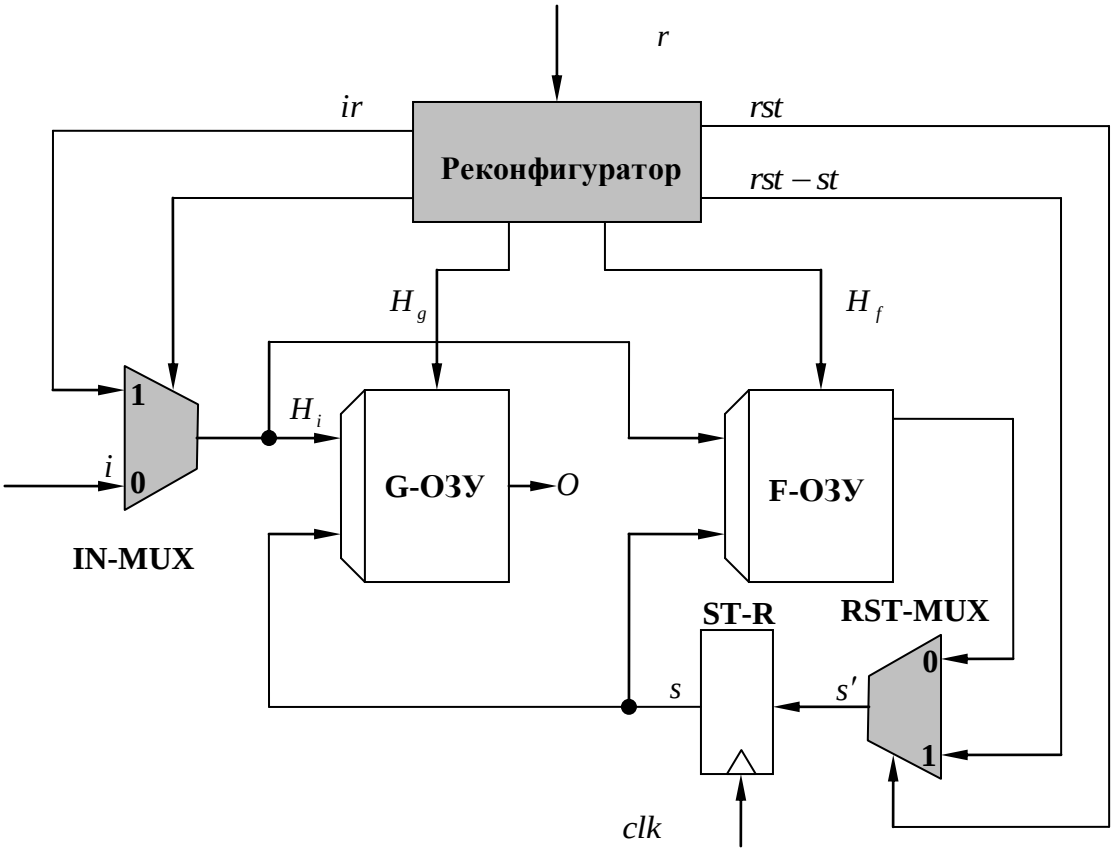

Рис. 2

Регистр состояния **ST-R** содержит текущее состояние *s*. Этот регистр заполняется по фронту каждого сигнала синхронизации значением следующего состояния s'.

Мультиплексор сброса **RST-MUX** используется для сброса автомата в нулевое состояние ($s' = S_0$) в зависимости от сигнала сброса rst. Таким образом, не имеет значения, в каком текущем состоянии *s* находится автомат, всегда есть возможность войти в нулевое состояние S_0 по следующему фронту сигнала синхронизации.

Когда автомат работает в нормальном режиме, мультиплексор **IN-MUX** выбирает входной сигнал $i(H_i(i,r)) = i$. Адрес в блоках ОЗУ ***F*** и ***G*** зависит от внешнего сигнала *i* и текущего состояния *s*. ***G*-ОЗУ** генерирует выход *o*, а ***F*-ОЗУ** генерирует следующее состояние s'.

В режиме реконфигурации мультиплексор **IN-MUX** выбирает сигнал *ir*, генерируемый блоком реконфигуратора $(H_i(i,r) = ir)$. Адрес в блоках ОЗУ ***F*** и ***G*** зависит от внешнего сигнала *ir* и текущего состояния *s*, т.е. входной сигнал *i* не оказывает влияния на автомат. В зависимости от состояния реконфигурации *r*, реконфигуратор генерирует новые значения ir, H_f, H_g для занесения в ***F*-ОЗУ** и ***G*-ОЗУ**.

Основными преимуществами данной реализации реконфигурируемых конечных автоматов является возможность работать на высокой тактовой частоте по причине простоты, а также независимость от конкретного оборудования и от размещения внутри кристалла или FPGA, в отличие от подходов, где схема должна генерировать конфигурационные данные в виде потоков данных, зависимых от конкретного производителя FPGA.

Алгоритмы реконфигурации

Для реализации реконфигурации исходного конечного автомата M в M' необходимо дать ответы на вопросы:

- При каких условиях M может превратиться в M' путем реконфигурации?
- Если M может превратиться в M', тогда каково минимальное число тактов реконфигурации?

Допустим, мы имеем конечные автоматы $M = (I, O, S, S_0, F, G)$ *и* $M' = (I', O', S', S'_0, F', G')$. S_c - объединение S *и* S', т.е. $S \subseteq S_c$ *и* $S' \subseteq S_c$. Аналогично определим I_c как объединение I и I', т.е. $I \subseteq I_c$ и $I' \subseteq I_c$, а также $O \subseteq O_c$ и $O' \subseteq O_c$.

Назовем *реконфигурируемостью автомата* M возможность его преобразования в конечный автомат M' при помощи конечной последовательности шагов, начинающейся в состоянии $S_0 \in S$ и заканчивающейся в $S'_0 \in S'$.

Оптимальной реконфигурацией будем называть преобразование конечного автомата M в M' за минимальное число шагов из состояния $S_0 \in S$ *в* $S'_0 \in S'$.

Допустим, мы имеем конечные автоматы $M = (I, O, S, S_0, F, G)$ *и* $M' = (I', O', S', S'_0, F', G')$. Пусть $T' = \{(i, s_x, s_y, o) : i \in I', s_x \in S', s_y = F'(i, s_x), o = G'(i, s_x)\}$ определяет все множество переходов автомата M'. Тогда переход $t_d = (i, s_x, s_y, 0) \in T'$ называется дельта-переходом и должен быть реконфигурирован для того, чтобы преобразовать M в M', если хотя бы одно из этих условий соблюдается:

- $i \in I \wedge s_x \notin S \wedge s_y \notin S \wedge o \notin O$;
- $s_y \neq F(i, s_x) \wedge i \in (I \cap I') \wedge s_x \in (S \cap S')$;
- $o \neq G(i, s_x) \wedge i \in (I \cap I') \wedge s_x \in (S \cap S')$.

Введенное понятие иллюстрирует следующий пример.

Пример 1. Рассмотрим конечный автомат $M = (I, O, S, S_0, F, G)$

$I = \{i\}, i \in \{0,1\}, O = \{o\}, o \in \{0,1\}, S = \{S_0, S_1, S_2\}$ и конечный автомат $M' = (I', O', S', S_0', F', G')$ $I' = I, O' = O, S' = \{S_0, S_1, S_2, S_3\}$ (рис. 3).

Для того, чтобы реконфигурировать M в M', нужны следующие дельта-переходы: $T_d = \{(0, S_1, S_0, 0), (1, S_2, S_3, 0), (1, S_3, S_3, 1), (0, S_3, S_0, 0)\}$(выделены жирными линиями).

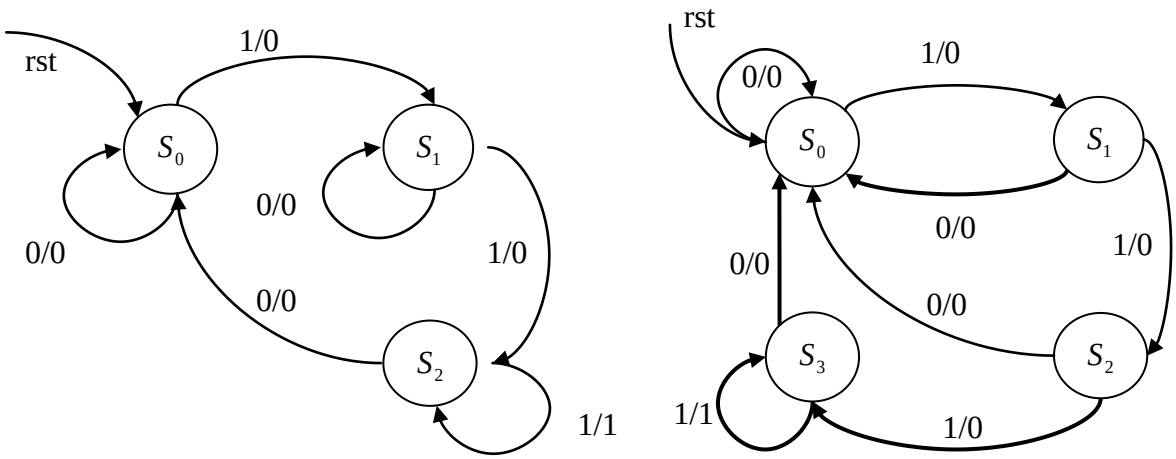

Рис. 3

Пусть $T_d \subseteq T'$ – множество всех дельта-переходов. Поскольку мы не можем реконфигурировать более одного перехода за такт, необходимо создать последовательность шагов, чтобы реконфигурировать каждый дельта-переход автомата M'. Назовем эту последовательность *программой реконфигурации* $Z = (z_0, z_1, ..., z_n), z_k \in T', 0 \le k \le n.$ Программа реконфигурации имеет начальное состояние, представляющее собой последнее состояние автомата M перед началом реконфигурации, и конечное состояние, которым является начальное состояние автомата M' после окончания реконфигурации. При помощи программы реконфигурации можно рассчитать необходимые действия для реконфигурации.

В процессе реконфигурации иногда возникает необходимость произвести несколько переходов, которые не являются дельта-переходами или уже реконфигурированы для того, чтобы достигнуть состояния с исходящим дельта-переходом. Для того, чтобы укоротить путь до этого дельта-перехода, мы можем реконфигурировать любой переход из текущего состояния в переход в необходимое нам состояние. Назовем такой переход *временным переходом*. Реконфигурирование уже конфигурированного перехода приводит к тому, что текущий переход превращается в дельта-переход. Далее предлагается алгоритм вычисления программы реконфигурации с использованием принципа временных переходов.

Эвристический алгоритм переходов

Пусть $M = (I, O, S, S_0, F, G)$ определяет конечный автомат в текущем состоянии $s \in S$, а $M' = (I', O', S', S_0', F', G')$ определяет целевой конечный автомат. В дальнейшем будем считать T_d множеством дельта переходов. Пусть $t_d \in T_d$, $t_d = (i, s_x, s_y, o)$, $H_{in} : T_d \to S'$, $H_{out} : T_d \to S'$, $H_{in}(t_d) = s_y$, $H_{out}(t_d) = s_x$. Функция $H_{in}(t_d)$ возвращает конечное состояние дельта перехода t_d, а функция $H_{out}(t_d)$ возвращает исходное состояние дельта перехода t_d. Пусть $S_0' \in S'$ будет конечным состоянием автомата после выполнения программы реконфигурации.

Вход: конечный автомат $M' = (I', O', S', S_0', F', G')$ и множество дельта переходов T_d.

Выход: программа реконфигурации Z.

(1) $n := 1$;
(2) $i_0 :=$ любой возможный входной сигнал $i \in I'$ конечного автомата M'
(3) $z_0 :=$ переход сброса
(4) для всех $t_d \in T_d$ {
(5) $z_n :=$ временный переход
(6) $z_{n+1} :=$ дельта-переход
(7) $z_{n+2} :=$ переход сброса
(8) $n := n + 3$;
(9) }
(10) $z_n := (i_0, S'_0, F'(i_0, S'_0), G'(i_0, S'_0))$;
(11) $z_{n+1} :=$ переход сброса

Сначала мы устанавливаем индекс n в единицу (1). Переменная i_0 представляет собой постоянный входной сигнал, используемый для всех временных переходов, которые мы используем в цикле. Переменная i_0 может быть любым входным сигналом $i \in I'$ автомата M' (2). Вне зависимости от текущего состояния мы переходим в состояние сброса S'_0 автомата M' (3). Внутри цикла (4) мы выбираем временный переход в зависимости от входного условия i_0 для прямого перехода в состояние с исходящим дельта-переходом (5). Таким образом, производится дельта переход. Мы конфигурируем дельта-переход (6) и возвращаемся в начальное состояние S'_0 (7). Шаги внутри цикла (5–8) повторяются для каждого оставшегося дельта-перехода, кроме того шага, который вызван временным переходом. Поскольку в каждой итерации цикла (4) для каждого временного перехода используется одинаковое входное условие i_0, дельта-переходы больше не создаются. Наконец, дельта-переход, вызванный временным переходом, реконфигурируется (10), и при помощи сброса автомата мы переходим в состояние S'_0 и заканчиваем реконфигурацию.

Пример 2

Рассмотрим конечный автомат $M = (I, O, S, S_0, F, G)$, $I = \{i\}, i \in \{0,1\}, O = \{o\}$, $o \in \{0,1\}$, $S = \{S_0, S_1, S_2\}$ и конечный автомат $M' = (I', O', S', S'_0, F', G')$, $I' = I$, $O' = O$, $S' = \{S_0, S_1, S_2, S_3\}$, графы которых изображены на рис.3.
$T_d = \{(0, S_1, S_0, 0), (1, S_2, S_3, 0), (1, S_3, S_3, 1), (0, S_3, S_0, 0)\}$ – набор дельта-переходов и S'_0 – конечное состояние.

Предположим, что мы находимся в состоянии S_1 конечного автомата M. Сначала переходим в начальное состояние S'_0 автомата M'. После этого переходим в любое состояние с исходящим дельта переходом, например, в состояние S_2 при помощи временного перехода $(1, S_0, S_2, 0)$, превращая существующий переход $(1, S_0, S_1, 0)$ в дельта-переход (рис. 4, *1*). После этого мы реконфигурируем переход $(1, S_2, S_3, 0)$ (рис. 4, *2*) и снова возвращаемся в начальное состояние S_0 (рис. 4, *3*).

Теперь, когда переход $(1, S_2, S_3, 0)$ уже реконфигурирован, мы продолжаем таким же образом обрабатывать каждый оставшийся дельта-переход, кроме того, который был вызван временным переходом. После того, как все остальные дельта-переходы реконфигурированы,

мы окончательно реконфигурируем дельта-переход $(1, S_0', S_1, 0)$. При сбросе мы попадаем в начальное состояние S_0' и заканчиваем реконфигурацию. Таким образом, в приведенном примере программа реконфигурации, полученная при помощи эвристического алгоритма переходов, имеет вид

$Z = (rst, (1, S_0, S_2, 0), (1, S_2, S_3, 0), rst, (1, S_0, S_3, 0), (1, S_3, S_3, 1), rst, (1, S_0, S_1, 0), (0, S_1, S_0, 0), rst, (1, S_0, S_3, 0), (0, S_3, S_0, 0), rst, (1, S_0, S_1, 0), rst)$.

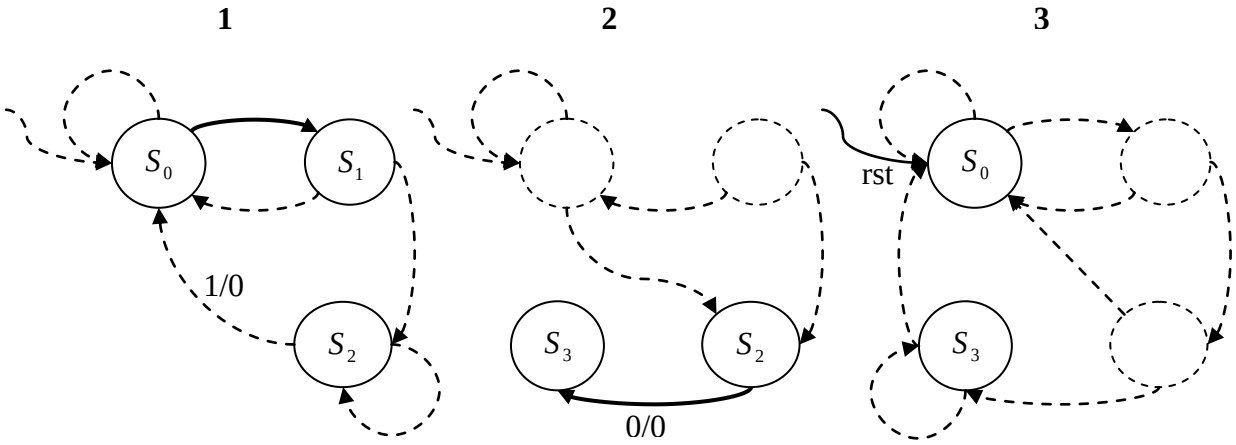

Рис. 4

Несомненная польза изложенной процедуры состоит в том, что длина получаемой программы реконфигурации легко предсказуема, поскольку она зависит только от количества дельта-переходов. На основании изложенного можно сформулировать следующие утверждения.

Утверждение 1. Всегда возможно определить конечную последовательность шагов реконфигурации для преобразования M в M'.

Утверждение 2 . Предположим что $|T_d|$ – количество дельта-переходов, и конечный автомат M' возможно перевести в состояние S_0' из любого текущего состояния. Тогда верхняя граница длины программы реконфигурации Z, сформированной при помощи эвристического алгоритма переходов, равна $3(|T_d|+1)$.

Утверждение 3 Нижняя граница длины программы реконфигурации – количество-дельта переходов $|T_d|$.

Выводы

Рассмотрен высокоуровневый, независимый от конкретной технологии подход к реализации концепции реконфигурируемых конечных автоматов. Кроме примитивной подмены контекста, предложенный вариант реализации позволяет реконфигурировать конечный автомат при помощи последовательности переходов из состояния в состояние.

Предложен вариант реализации подобных автоматов в оборудовании, а также алгоритмы реконфигурации и пределы их реализации.

Библиографический список

1. **Sidhu, R.** String matching on multicontext FPGAs using self-reconfiguration / R. Sidhu, A. Mei, and V. K. Prasanna // Proc. 7th Int. Symposium on Field Programmable Gate Arrays (FPGA).

2. **Fekete, S.P.** Optimal FPGA module placement with temporal precedence constraints/ S. P. Fekete, E. Köhler, J. Teich // Proc. DATE 2001, Design, Automation and Test in Europe. Munich, Germany, March 13-16 2001. IEEE Computer Society Press. P. 658–665.

Дата поступления
в редакцию 06.07.2010

V.V. Kondratev, K.M. Michailov

RECONFIGURING FINITE AUTOMATIONS

The concept of reconfiguring finite automations as formal model for the description of finite automations, which configuration can be changed during its operating time. With appearance of reconfigured logic this model becomes important for the description and making of the reconfigured equipment. It is offered algorithmic solutions and effective hardware representation of transformation process of an existing finite automation in other finite automation.

Key words: reconfigured finite automation, reconfiguring finite automation, events, states.