708

ISSN 0868-4081
0868-4154

TALLINNA TEHNIKAÜLIKOOLI

TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

МАШИННОЕ ПРОЕКТИРОВАНИЕ ЭЛЕКТРОННЫХ УСТРОЙСТВ И СИСТЕМ

TALLINN 1990

708

ALUSTATUD 1937

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 681.511.09 + 681.32 + 621.382

МАШИННОЕ ПРОЕКТИРОВАНИЕ ЭЛЕКТРОННЫХ УСТРОЙСТВ И СИСТЕМ

Электротехника и автоматика XXXIX

TALLINN 1990

Содержание

ТАЛЛИННСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Труды ТТУ № 708
МАШИННОЕ ПРОЕКТИРОВАНИЕ ЭЛЕКТРОННЫХ УСТРОЙСТВ И СИСТЕМ
Электротехника и автоматика XXXIX
На русском и английском языках
Ответственный редактор У. Тамм
Технический редактор Е. Зорина
Сборник утвержден коллегией Трудов ТТУ 22.05.90
Подписано к печати 5.12.90
Формат 60/90x16
Печ. л. 7,25 + 0,25 приложение. Уч.-изд. л. 6,07
Тираж 300. Заказ № 879. Цена 3 руб.
Таллиннский технический университет
200108 Таллинн, Эхитаяте тее, 5
Ротапринт ТТУ
200006 Таллинн, ул. Коскла, 2/9

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 519:713

А. Кеэваллик, Л. Каширова,
М. Круус

ПОСТРОЕНИЕ ПРОВЕРЯЮЩЕГО ТЕСТА ДЛЯ СЕТИ АВТОМАТОВ

Введение. С усовершенствованием элементного базиса и усложнением проектируемых дискретных управляющих устройств (ДУУ) все большую важность приобретают проблемы их функциональной диагностики. Как и многие другие задачи синтеза и анализа ДУУ, задача функциональной диагностики чрезвычайно трудоемка ввиду ее комбинаторной сложности. Одним из наиболее перспективных путей преодоления "барьеров сложности" является развитие декомпозиционных методов синтеза и анализа ДУУ, позволяющих свести сложные задачи синтеза и анализа к совокупности менее сложных. Однако не менее важно при этом и то, что декомпозиционный подход позволяет решить задачи синтеза и анализа в едином комплексе, имея в виду одновременно цели обеих задач. При этом особо следует подчеркнуть то, что таким образом откроется возможность сочетать цели синтеза и анализа, начиная с самых ранних этапов проектирования ДУУ, т.е. с абстрактного уровня его представления. Тем самым через декомпозиционный подход откроется оригинальный путь для синтеза контролепригодных ДУУ.

В данной работе рассматриваются проблемы построения проверяющих тестов для сетевых реализаций ДУУ, полученных в результате декомпозиции. Показано, что контрольная последовательность, найденная на уровне исходной автоматной модели ДУУ, позволяет обнаружить одиночные неисправности на связях между компонентами сети, являющейся ее декомпозицией. Также показано, что на основе найденной последовательности можно построить проверяющий тест для декомпозиции, реализованной в виде сети из программируемых логических матриц (ПЛМ).

Основные понятия. В качестве модели ДУУ рассмотрим микропрограммный автомат (МПА) Мили, который определяется как система

$$A = (S, \{0,1\}^L, \{0,1\}^M, \delta, \lambda, s_0),$$

где S - множество внутренних состояний;

$\{0,1\}^L$ и $\{0,1\}^M$ - входной и выходной алфавит соответственно;

L и M - число входных и выходных двоичных переменных;

$\delta : S \times \{0,1\}^L \to S$ - функция переходов;

$\lambda : S \times \{0,1\}^L \to \{0,1\}^M$ - функция выходов;

s_0 - начальное состояние.

Координаты векторов $\{0,1\}^L$ и $\{0,1\}^M$ обозначим соответственно $x_1, x_2, \ldots, x_L$ и $y_1, y_2, \ldots, y_M$.

В дальнейшем предположим, что А минимальный, полностью определенный, сильносвязанный и имеет диагностическую последовательность z [I].

МПА компактно описывается в виде списка переходов, одной из форм задания которого является прямая таблица переходов и выходов (см. табл. I).

Таблица I

№ п/п	Начальное состояние $s_н$	Конечное состояние $s_к$	Обобщенный входной сигнал $X(s_н, s_к)$	Выходная реакция $Y(s_н, s_к)$
I		2	x_1	$y_1 y_2$
2	I	2	$\bar{x}_1 x_2$	$y_1 y_3$
3		5	$\bar{x}_1 \bar{x}_2$	$y_1 y_2$
4		I	$\bar{x}_1 \bar{x}_3$	$y_3 y_4$
5	2	2	$x_1 \bar{x}_3$	y_4
6		3	x_3	$y_1 y_2 y_3$
7		I	$\bar{x}_1 \bar{x}_4$	$y_3 y_4$
8	3	2	$x_1 \bar{x}_4$	y_4
9		4	x_4	$y_2 y_3 y_4$
I0	4	I	x_3	y_1
II		2	$\bar{x}_3$	y_4
I2	5	4	(I)	$y_3 y_4$

Обозначим через $(\{0,1\}^L)^*$ множество всех слов в алфавите $\{0,1\}^L$. Пусть $\alpha \in (\{0,1\}^L)^*$. Расширим понятия функций переходов и выходов МПА A.

$\bar{\lambda}(\delta,\alpha)$ - выходная последовательность, соответствующая начальному состоянию δ и входной последовательности α. Значением функции $\bar{\delta}(\delta,\alpha)$ считаем конечное состояние, в которое переходит автомат из состояния δ под воздействием α.

Обозначим через α_δ обобщенный входной сигнал (см. табл. I), соответствующий некоторому переходу из состояния δ. Фактически α_δ некоторая конъюнкция из входных двоичных переменных автомата A.

Если T_δ входная последовательность, переводящая A в состояние δ, то последовательность $T_\delta \alpha_\delta z$ идентифицирует переход из состояния δ в состояние $\delta(\delta,\alpha_\delta)$, а последовательность $T_\delta z$ - состояние δ. На основе $T_\delta \alpha_\delta z$ и $T_\delta z$ можно для A составить последовательность α_A, идентифицирующую все его состояния и переходы [2]. α_A называется контрольной последовательностью A. Ее длина пропорциональна произведению $2^L \cdot |S|^2 \cdot d(z)$ [2], где $d(z)$ - длина диагностической последовательности z. Отметим, что метод нахождения контрольной последовательности с учетом специфики МПА предложен в источнике [I].

Далее приводим необходимые нам понятия, относящиеся к декомпозиционному синтезу МПА [3].

Сетью МПА называется система

$$N = (\{A_i\}, \{0,1\}^L, \{0,1\}^M, \{f_i\}, g, t_0),$$

где $\{A_i = (S_i, \{0,1\}^{l_i + k_i}, \delta_i)\}$, $1 \le i \le k$ - множество компонентов сети, в котором $\{0,1\}^{l_i}$ и $\{0,1\}^{k_i}$ - внешний и внутренний входной алфавит компонентного автомата соответственно;

$\{0,1\}^L$ и $\{0,1\}^M$ - входной и выходной алфавит сети;

$\{f_i : (\times S_j) \to \{0,1\}^{k_i}\}$, $1 \le i, j \le k$ - множество функций соединения;

$g : (\times S_j) \times \{0,1\}^L \to \{0,1\}^M$ - выходная функция сети;

t_0 - начальное состояние сети.

Через $K(S_i)$ обозначим некоторую кодировку состояний компонентного автомата A_i.

По сети N можно всегда найти функционально эквивалентный ей автомат A_N, который называется результирующим автоматом сети N. Не приводя здесь определения и конструктивного метода построения A_N, отметим лишь, что входной и выходной алфавиты равняются соответственно на $\{0,1\}^L$ и $\{0,1\}^M$ и что множество его внутренних состояний $S_N = \times S_i$. В дальнейшем через $pr_i\sigma$ обозначим состояние компонентного автомата A_i, соответствующее состоянию сети $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k)$, $\sigma \in S_N$.

Сеть N называется декомпозицией заданного МПА A, если она реализует A, т.е. если и только если для ее результирующего автомата A_N существует подавтомат A'_N, который изоморфен A [3]. Конструктивный метод построения декомпозиции МПА приведен в источнике [3].

В дальнейшем предположим, что у нас имеется сеть N, которая является декомпозицией для МПА A. Автомат A'_N определяет так называемую рабочую область сети N. Образно выражаясь, сеть N имеет множество состояний $S'_N \subset S_N$, необходимое для реализации A, которое в сочетании с входным алфавитом $\{0,1\}^L$ и образует рабочую область $S'_N \times \{0,1\}^L$. Отметим, что исправная сеть N, реализующая A, никогда не выходит из рабочей области.

Тестирование связей в сетевых реализациях МПА. Переходим теперь к проблемам, связанным с контролем сетевых реализаций МПА. Если в источнике [4] рассматривались задачи нахождения диагностических и установочных последовательностей для компонентных автоматов, то в данной работе внимание акцентируется на проблеме обнаружения неисправностей на входах, выходах и связях между компонентами сети.

Приводим классификацию рассматриваемых нами неисправностей F в сети N:

F_x - константные неисправности на входах сети N;
F_y - константные неисправности на выходах сети N;
F_A - константные неисправности на входах и выходах компонентных автоматов сети N.

При рассмотрении выделенных неисправностей предположим, что они существенные, т.е. искажают поведение сети N в ее рабочей области. Отметим, что неисправности, выводящие сеть N из рабочей области, легко обнаружить. Для этого достаточно ввести дополнительную выходную переменную (в нашей работе y_ε), которая принимает значение "I" при тех состояниях сети N, которые не входят в ее рабочую область.

Пусть α_A, как было сказано выше, последовательность, идентифицирующая все состояния и переходы МПА A [I]. Назовем последовательность α_A доопределенной и обозначим через $\bar{\alpha}_A$, если в ней во всех обобщенных сигналах α_s свободные переменные доопределены.

Дальше покажем, что при помощи последовательности $\bar{\alpha}_A$ можно обнаруживать неисправности из множества F в сети N, которая является декомпозицией A.

Утверждение I. Произвольная доопределенная последовательность $\bar{\alpha}_A$ является проверяющим тестом относительно любой неисправности из F при условии, что сеть N имеет исправные компоненты $\{A_i\}$, $\{f_i\}$ и g.

Поскольку в множестве F три класса неисправностей, то докажем утверждение относительно каждого из них.

I. Класс F_x.

Вследствие неисправности из класса F_x любая доопределенная последовательность $\bar{\alpha}_A$ преобразуется в последовательность $\bar{\alpha}'_A$, в которой одна из входных переменных $x_i = const$. Если переменная x_i существенно повлияет на поведение сети, а последовательность $\bar{\alpha}_A$ идентифицирует все переходы исходного МПА, то $\bar{g}(t_o, \bar{\alpha}_A) \neq \bar{g}(t_o \bar{\alpha}'_A)$. Следовательно, неисправность из класса F_x обнаружима последовательностью $\bar{\alpha}_A$ (см. [2]).

2. Класс F_y

Произвольная неисправность из класса F_y искажает в реакции сети N значение некоторой выходной переменной y_i на константную величину. Обозначим искаженную реализацию через $\bar{g}'(t_o, \bar{\alpha}_A)$. Очевидно, что для произвольной $\bar{\alpha}_A$ справедливо следующее: $g(t_o, \bar{\alpha}_A) \neq \bar{g}'(t_o, \bar{\alpha}_A)$.

Следовательно, неисправность класса F_y всегда обнаружима по выходу сети N.

3. Класс F_A.

Неисправности класса F_A искажают функцию переходов некоторого компонентного автомата A_i, сужая либо входной алфавит $\{0,1\}^{l_i+k_i}$, либо множество состояний S_i. Вследствие такого сужения существует хотя бы один переход компонентного автомата A_i, для которого

$$\delta_i(pr_i s, c) \neq \delta_i'(pr_i s, c),$$

где $c \in \{0,1\}^{l_i+k_i}$.

Поскольку $\bar{\alpha}_A$ идентифицирует все переходы компонентного автомата A_i, то неисправность обязательно искажает выходную реакцию. Следовательно, она обнаружима по выходу сети N.

Утверждение доказано.

Пример I. Прямая таблица переходов МПА A приведена в таблице I. Декомпозиция МПА A описана в виде таблиц переходов компонентных автоматов A_1 и A_2 (табл. 2 и 3). Исходным состоянием A считаем $s_0 = I$, исходным состоянием сети $N - t_0 = (B_1, C_1)$. Сеть N изображена на рис. I. Заметим, что функции f_1 и f_2 в данной реализации представляют тождественные отображения. A имеет диагностическую последовательность $z = \bar{x}_1 \bar{x}_2 x_3 x_4$.

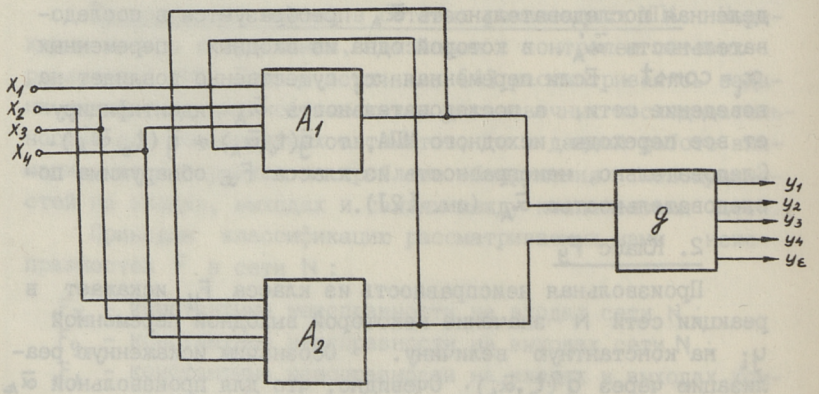


Рис. 1.

Таблица 2

№	$B_н$	$B_к$	$X(B_н, B_к)$	C_i
I	B_I	B_2	I	-
2	B_2	B_I	$\bar{x}_I\bar{x}_3$	C_I
3		B_I	x_3	C_2
4		B_I	I	C_3
5		B_2	$x_I\bar{x}_3$	C_I
6		B_2	$\bar{x}_3$	C_2
7		B_3	x_3	C_I
8	B_3	B_I	$\bar{x}_I\bar{x}_4$	-
9		B_2	$x_I\bar{x}_4$	-
I0		B_2	x_4	-

Таблица 3

№	$C_н$	$C_к$	$X(C_н, C_к)$	B_i
I	C_I	C_I	x_I	B_I
2		C_I	$\bar{x}_I\bar{x}_2$	B_I
3		C_I	I	B_2
4		C_I	$\bar{x}_4$	B_3
5		C_2	x_4	B_3
6		C_3	$\bar{x}_I\bar{x}_2$	B_I
7	C_2	C_I	I	-
8	C_3	C_I	I	-

В таблице 4 приведено соответствие между состояниями МПА A и сети N .

Рассмотрим некоторое подмножество неисправностей F', а именно, константные неисправности на выходах $y_1=0$ и

$y_4 = 1$, неисправности на выходе компонентных автоматов A_1 и A_2, константные значения входов $x_1 = 0$ и $x_4 = 0$. В таблице 5 описан начальный отрезок теста α_A, позволяющий обнаружить неисправности класса F'.

В первой строке приведена последовательность α_A, во второй ее произвольно доопределенный вариант $\bar{\alpha}_A$, а в третьей - состояния МПА A,

Таблица 4

Λ_i	(B_j, C_k)
I	(B_1, C_1)
2	(B_2, C_1)
3	(B_3, C_1)
4	(B_2, C_2)
5	(B_2, C_3)

Таблица 5

I	α_3	α_{12}	α_2	α_6	α_9	α_{10}	α_1	α_8
2	z	z	$\bar{x}_1 x_2 x_3 x_4$	z	z	z	$x_1 \bar{x}_2 x_3 x_4$	z
3	I	5	I	2	3	4	I	2
4	$y_1 y_2$	$y_3 y_4$	$y_1 y_3$	$y_1 y_2 y_3$	$y_2 y_3 y_4$	y_1	$y_1 y_2$	$y_1 y_2 y_3$
5	y_2							
6	$y_1 y_2 y_4$							
7	$y_1 y_2$	y_ε						
8	y_ε							
9	$y_1 y_2$	$y_3 y_4$	$y_1 y_3$	$y_1 y_2 y_3$	$y_2 y_3 y_4$	y_1	$y_1 y_2$	$y_3 y_4$
I0	$y_1 y_2$	$y_3 y_4$	$y_1 y_3$	$y_1 y_2 y_3$	$y_3 y_4$			

через которые он проходит при подаче входной последовательности $\bar{\alpha}_A$, причем начальное состояние $z_0 = 1$. В четвертой строке приведена реакция исправной сети на входную последовательность $\bar{\alpha}_A$. Строки с пятой по десятую описывают реакцию сети N на последовательность $\bar{\alpha}_A$ в присутствии неисправностей класса F' в порядке их описания, причем реакция описана до появления некорректного выходного значения.

Тестирование сетевых реализаций на базе ПЛМ. Пусть сеть N, являющаяся декомпозицией для МПА A, реализована на базе ПЛМ типа И-ИЛИ и D-триггерах.

Разделим возможные неисправности в ПЛМ (см. рис. 3 - 5) на два следующих типа: во-первых, это неисправности типа "появление соединения (ПС)" и во-вторых, это неисправности типа "нарушение соединения (НС)". Неисправность типа ПС - это неисправность, в результате которой появляется лишнее соединение в точке пересечения линий ПЛМ; неисправность типа НС - это неисправность, в результате которой исчезает нужное соединение в точке пересечения линий ПЛМ. В источнике [5] показано, что константные неисправности на линиях ПЛМ и короткое замыкание между линиями сводятся к неисправностям перечисленных типов.

В дальнейшем предлагаем следующую классификацию неисправностей в ПЛМ. Множество рассматриваемых неисправностей G состоит из следующих классов:

1. G_V - неисправности типа НС и ПС в схеме ИЛИ ПЛМ.
2. G_n - неисправности типа ПС на линии, соответствующей противоположному значению любой из существенных [5] переменных терма.
3. G_H - неисправности типа НС на линии, соответствующей любой из существенных переменных терма.
4. G_C - неисправность типа ПС на линии, соответствующей несущественной переменной терма.

Утверждение 2. Существует проверяющий тест $\bar{\alpha}$ относительно множества неисправностей G.

Докажем утверждение относительно приведенных классов неисправностей.

I. Класс G_V.

Неисправность из класса G_V может присутствовать в любом из компонент сети. Если рассматриваемая неисправность присутствует в ПЛМ, реализующая функцию g, то для произвольного доопределения $\bar{\alpha}_A$ справедливо следующее:

$$\bar{g}(t_0,\alpha_A) \neq \bar{g}'(t_0,\bar{\alpha}_A) \quad (1),$$

где t_0 - исходное состояние сети;

$\bar{q}'(t_0, \bar{\alpha}_A)$ - функция выходов неисправной сети.

Вследствие этого доказательство очевидно.

Неисправность из множества G_v в ПЛМ-ах, реализующих функции соединения $\{f_i\}, 1 \le i \le k$, приводит к тому, что меняется внутренний алфавит некоторого компонентного автомата, а следовательно, меняется и входная последовательность этого компонентного автомата, соответствующая доопределенному тесту $\bar{\alpha}_A$. Поскольку тест идентифицирует все переходы компонентного автомата, то условие (I) выполняется и неисправность обнаруживается.

Доказательство для случая, когда неисправность из множества G_v присутствует в ПЛМ реализующих компонентные автоматы $\{A_i\}$, $1 \le i \le k$, аналогичное.

2. Класс $G_п$.

Неисправности из класса $G_п$ приводят к тому, что соответствующий неисправности терм (интервал) принимает константное нулевое значение, а следовательно, искажает некоторые выходы ПЛМ. Доказательство аналогично классу G_v.

3. Класс $G_н$.

Вследствие неисправности из класса $G_н$ неисправный терм не зависит уже от некоторой существенной для него переменной. Пусть неисправность из класса $G_н$ присутствует в ПЛМ, реализующей некоторый компонентный автомат A_i. Из определения неисправности следует, что в ПЛМ существует терм t_j, который превратился в терм t_j', где $t_j \subset t_j'$. Поскольку исходный МПА полностью определенный, то в рабочей области полностью определены и компонентные автоматы A_i. Вследствие неисправности нарушается ортогональность конъюнкции (термов) и, следовательно, существует терм t_k, для которого $t_k \cap t_j' \neq \Phi$.

Если рассматриваемая неисправность искажает работу сети в рабочей области, то она обнаруживается при верификации терма t_k. Поскольку каждый терм соответствует некоторому переходу автомата A_i, а согласно источника [1] тест α_A идентифицирует все интервалы переходов, то существует доопределенная $\bar{\alpha}_A$, обнаруживающая данную неисправность.

Доказательство утверждения для случаев, когда неисправность присутствует в элементах, реализующих функции $\{f_i\}$, $1 \le i \le k$ и функцию g, проводится аналогично.

4. Класс G_C.

Анализ последнего класса неисправностей G_C показывает, что в ее результате некоторый интервал t_j, реализуемый на одном терме ПЛМ, дробится на два интервала $t_j^1 = t_j \cdot x_i$ и $t_j^2 = t_j \cdot \bar{x}_i$, где x_i - несущественная входная переменная данного терма t_j (или его инверсия), которому соответствует рассматриваемая неисправность. На интервале t_j^1 значение терма совпадает с правильным, а на интервале t_j^2 неисправность скажает выход ПЛМ. Следовательно, становится необходимым верификация обоих названных интервалов.

Пусть имеется два доопределения $\bar{\alpha}_A^1$ и $\bar{\alpha}_A^2$ теста α_A. В тесте $\bar{\alpha}_A^1$ все интервалы доопределены с помощью прямых несущественных переменных, а в тесте α_A^2 - с помощью инверсных. В таком случае тест $\bar{\alpha}_A^1$ обнаружит неисправности типа ПС на инверсных линиях несущественных переменных, а тест $\bar{\alpha}_A^2$ - соответственно на прямых линиях. Доказательства одинаковы для всех компонентов сети. Из вышесказанного следует, что существует проверяющий тест $\bar{\alpha}$, который позволяет обнаружить любую неисправность из множества G. Тем самым утверждение доказано. Очевидно, что последовательность $\bar{\alpha}$ является проверяющим тестом для множества неисправностей $F \cup G$. Длина теста $d(\alpha)$ пропорциональна величине

$$(|S| + d(z))H,$$

где $|S|$ - мощность множества состояний;
$d(z)$ - длина диагностической последовательности;
H - длина прямой таблицы переходов МПА.

Пример 2. МПА A (табл. I) реализован в виде сети N (рис. 2). ПЛМ комбинационного типа, реализующие компонентные автоматы A_1 и A_2 и функцию выходов g, изображены на рис. 3-5 соответственно. Через y_ε обозначен дополнительный выход для обнаружения выхода сети из ее рабочей области. Рассмотрим следующее множество неисправностей в компонентном автомате A_1 (рис. 3): $G_V = \{5\}; G_M = \{3\}; G_H = \{4\}; G_C = \{1,2\}$. Процесс обнаружения этих неисправностей опи-

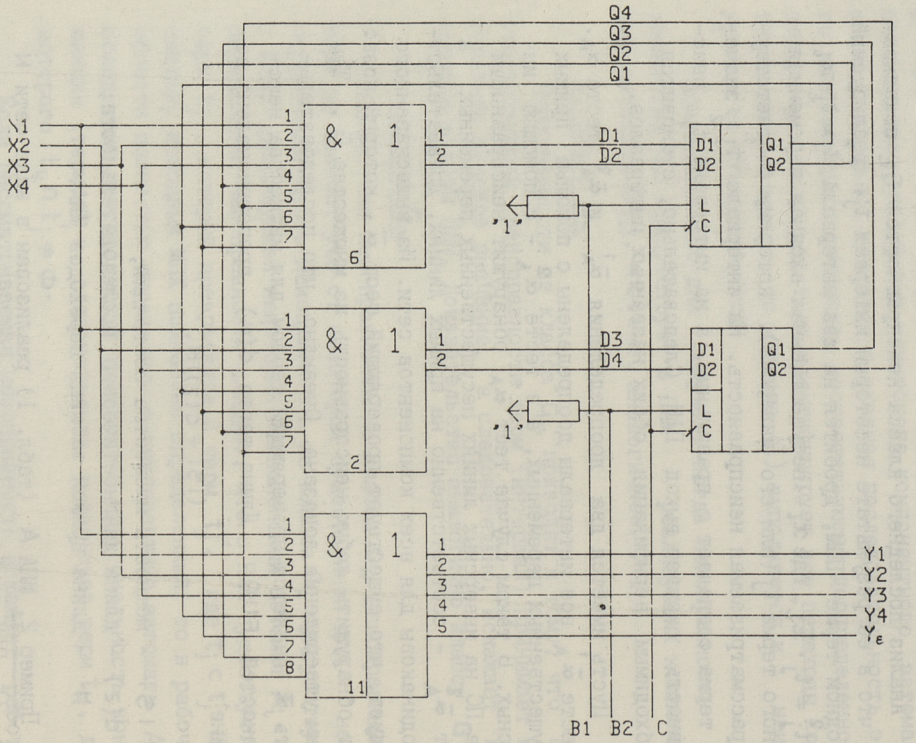


Рис. 2.

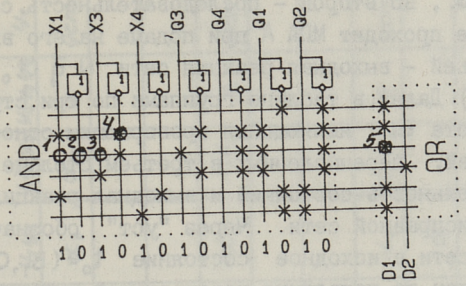


Рис. 3.

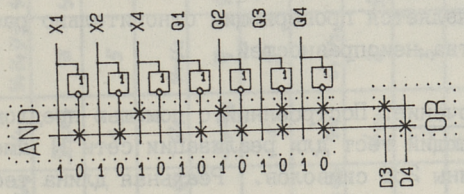


Рис. 4.

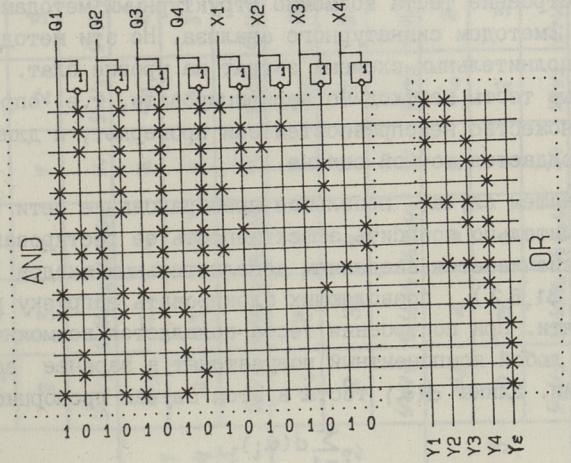


Рис. 5.

сан в таблице 6. В первой строке приведен начальный отрезок теста $\bar{\alpha}$, во второй - последовательность состояний, через которые проходит МПА A при подаче на его входы теста $\bar{\alpha}$, в третьей - выходная реакция сети N $\bar{q}(t_0, \alpha)$, где $t_0 = (B_1, C_1)$. Далее, в таблице группами по три строки следуют: отрезок теста $\bar{\alpha}$, являющийся проверяющим относительно неисправностей, перечисленных в третьем столбце таблицы; последовательность состояний и выходная реакция, соответствующие неисправной сети. Через "уст" обозначен сигнал установки сети в исходное состояние $t_0 = (B_1, C_1)$. Каждый тест приведен до появления некорректной выходной реакции, т.е. до обнаружения неисправностей. Конкатенация тестов $\bar{\alpha}^1_A \bar{\alpha}^2_A$ является проверяющим относительно рассматриваемого множества неисправностей.

Заключение. Построенный с помощью предложенного метода проверяющий тест для реализации сети N имеет верхнюю оценку длины 288 символов. Реальная длина теста 50 символов. Если для построения теста пользоваться только абстрактными методами [2], то его длина не более 960 символов. Построение теста возможно структурными методами, например, методом сигнатурного анализа. Но эти методы требуют дополнительных схемных затрат на уровне плат. Построенные тесты необходимо верифицировать, т.е. определять, какое множество неисправностей они проверяют, и длина теста не поддается точной оценке.

В нашем случае небольшое преобразование сети N может значительно повысить эффективность ее тестирования. Это обеспечивается введением дополнительных входов (рис. 2 - входы B1, B2), позволяющих блокировать загрузку регистров памяти. При построении теста появляется возможность оставлять любой компонентный полуавтомат в заранее заданном состоянии. Длина $d(\alpha)$ теста в этом случае пропорциональна

$$\sum_{i=1} d(\alpha_i),$$

где $d(\alpha_i)$ - длина теста для полуавтомата A_i.

Таблица 6

1	α_A		α_2	z	z	z	α_2	z	z	α_{11}	z	усл	α_1	α_6	z	усл
2	S		2	3	4	1	2	3	4	2	3	1	2	3	4	1
3	β		y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_1	y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_4	$y_1y_2y_3$	y_3y_4	y_1y_2	$y_1y_2y_3$	$y_2y_3y_4$	y_1
4	$\bar{\alpha}_A^1$	1	α_2	z	z	z	α_2	z	z	$\bar{x}_1x_3$	z	усл	α_1	α_6	z	усл
5	S'	4	2	3	4	1	2	3	4	1	5					
6	β'	5	y_1y_3	$y_1y_2y_3$	$y_1y_3y_4$	y_1	y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_4	y_1y_2					
7	$\bar{\alpha}_A^2$	2	α_2	z	z	z	α_2	z	z	x_1x_3	z	усл	α_1	x_1x_3	z	усл
8	S'		2	3	4	1	2	3	4	1	5					
9	β'		y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_1	y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_4	y_1y_2					
10	$\bar{\alpha}_A^3$	3	α_2	z	z	z	α_2	z	z	α_{11}	z	усл	α_1	x_1x_3	z	усл
11	S'		2	3	4	1	2	3	4	2	3	1	2			
12	β'		y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_1	y_1y_3	$y_1y_2y_3$	$y_2y_3y_4$	y_4	$y_1y_2y_3$	y_3y_4	y_1y_2	y_ϵ		

Литература

I. Круус М.Э. Контрольные эксперименты с микропрограммными автоматами // Тр. Таллиннск. политехн. ин-та. 1985. № 601. С. 45-50.

2. Hennі F.C. Fault-detecting experiments for sequential circuits // Proc. 5-th Ann. Symp. Switching Circuits Theory and Logical Design. 1964. P. 95-100.

3. Кеэваллик А.Э., Лейс П.Л. О декомпозиции конечных автоматов / Материалы семинара "Оптимизация в проектировании дискретных устройств". Л., 1976. С. 52-69.

4. Кеэваллик А.Э., Круус М.Э. Метод нахождения диагностических и установочных последовательностей для сетей автоматов // Тр. Таллиннск. политехн. ин-та. 1986. № 626. С. 43-61.

5. Nripendra N.B., James J. A testable PLA design with minimal hardware and test set // International Test Conference. 1985.

A. Keevallik, L. Kaširova, M. Kruus

Kontrolltesti süntees automaatide võrgu jaoks

Kokkuvõte

Artiklis käsitletakse diskreetse juhtseadme kontrolltesti sünteesi probleeme. Eeldatakse, et juhtseade on realiseeritud komponentautomaatidest koosneva võrguna programmeeritavatel loogikamaatrikseil. On tõestatud, et sünteesitud test võimaldab avastada käsitletavate klasside rikkeid nii võrgu komponentide vahelistel sidemetel kui ka komponentide sees. On toodud testi pikkuse hinnang. Teoreetilisi resultaate on illustreeritud näitega.

A. Keevallik, L. Kashirova, M. Kruus

Checking Test Derivation for Automata Network

Abstract

A method of checking test derivation for automata network has been developed. All components of this network are realized on programmable logic arrays. It is shown that the derivated test permits us to detect the faults of the classes observed. The upper bound of the test length is found. Theoretical conclusions are illustrated by examples.

Nr. 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 62.507

A. Keevallik, T. Lausmaa

INFORMATIONAL OPERATORS OF FINITE AUTOMATA

In this paper the technic of informational operators founded by J. Hartmanis and R. Stearns [1] for analyzing finite automata is developed further. It is shown that properties of operators m and M [1] showing the quality of information flow in finite automata are valid also in case of their iteration.

As problems concerning the realization of output function of finite automata are not dealt with in this paper we can define the finite automaton as a triplet $A = \langle I, S, \delta \rangle$, where I is an input alphabet, S is the set of internal states and $\delta: S \times I \to S$ is the next-state function.

The partition of finite set $X = \{x_1, x_2, \ldots, x_m\}$ into disjoint blocks $B_i^{(1)}$, $B_i^{(2)}$, ..., $B_i^{(\alpha_i)}$, $B_i^{(m_i)}$ is denoted by $\pi_i(X)$. The zero partition (each block contains only one element) is denoted by 0_X and the unit partition (containing only one block) by 1_X. $x_i \equiv x_j(\pi)$ means that elements x_i and x_j belong to the same block of the partition π. For any partitions $\pi_i(X)$ and $\pi_j(X)$ we define

$$\pi_i \cdot \pi_j \underset{Df}{=} \{B_i^{(\alpha)} \cap B_j^{(\beta)} \mid B_i^{(\alpha)} \in \pi_i;\ B_j^{(\beta)} \in \pi_j;\ B_i^{(\alpha)} \cap B_j^{(\beta)} \neq 0\} \text{ and}$$

$$\pi_i + \pi_j \underset{Df}{=} \prod \{\pi_k(X) \mid \pi_k \geqslant \pi_i, \pi_j\}, \text{ where } \pi_i \leqslant \pi_j \underset{Df}{\Leftrightarrow} \pi_i \cdot \pi_j = \pi_i.$$

The set of partitions on X and operators "." and "+"

defined above form a lattice which is denoted by $L(X)$.

Two partitions π_{i_1} and π_{i_2} on the set of states S form a partition pair [1] iff $(\forall y \in I)(S' \equiv S''(\pi_{i_1}) \Rightarrow \delta(S', y) \equiv \delta(S'', y)(\pi_{i_2}))$.

The set of all partition pairs of an automaton A forms a lattice [1] which is denoted by $K_A(S)$. In the $K_A(S)$ the operators m_A and M_A [1] are defined as follows

$$m_A(\pi) \underset{Df}{=} \prod\{\pi_i | \langle \pi, \pi_i \rangle \in K_A(S)\},$$

$$M_A(\pi) \underset{Df}{=} \sum\{\pi_i | \langle \pi_i, \pi \rangle \in K_A(S)\}.$$

If the inequality $\pi_i(S) \geq m_A(\pi_i(S))$ holds for the partition $\pi_i(S)$, then we say that π_i is an S.P. -partition [1].

Let us introduce now following definitions

$$m_A^0(\pi) \underset{Df}{=} \pi; \quad m_A^1(\pi) \underset{Df}{=} m_A(\pi); \quad m_A^k(\pi) \underset{Df}{=} m_A(m_A^{k-1}(\pi));$$

$$M_A^0(\pi) \underset{Df}{=} \pi; \quad M_A^1(\pi) \underset{Df}{=} M_A(\pi); \quad M_A^k(\pi) \underset{Df}{=} M_A(M_A^{k-1}(\pi)).$$

Theorem 1. Let $A = \langle I, S, \delta \rangle$ be an arbitrary automaton. For any $\pi_i(S), \pi_j(S)$ and $K \geq 0$ we have

(i) $\pi_i \geq \pi_j \Rightarrow m_A^k(\pi_i) \geq m_A^k(\pi)$;

(ii) $m_A^k(\pi_i + \pi_j) = m_A^k(\pi_i) + m_A^k(\pi_j)$;

(iii) $m_A^k(\pi_i \cdot \pi_j) \leq m_A^k(\pi_i) \cdot m_A^k(\pi_j)$;

(iv) $\pi_i \geq \pi_j \Rightarrow M_A^k(\pi_i) \geq M_A^k(\pi_j)$;

(v) $M_A^k(\pi_i + \pi_j) \geq M_A^k(\pi_i) + M_A^k(\pi_j)$;

(vi) $M_A^k(\pi_i \cdot \pi_j) = M_A^k(\pi_i) \cdot M_A^k(\pi_j)$;

(vii) $M_A^k(m_A^k(\pi_i)) \geq \pi_i$;

(viii) $m_A^k(M_A^k(\pi_i)) \leq \pi_i$;

(ix) $M_A^K(m_A^K(M_A^K(\pi_i))) = M_A^K(\pi_i)$;

(x) $m_A^K(M_A^K(m_A^K(\pi_i))) = m_A^K(\pi_i)$.

Before starting to prove the theorem we define an automaton $A_K \underset{Df}{=} \langle I^K, S, \delta_K \rangle$, where $\underbrace{I^K = I \times I \times \ldots \times I}_{k\text{-times}}$, and for every $\langle y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(K)} \rangle \in I^K$ and $s_\alpha \in S$ the next-state function is defined as

$$\delta_K(\langle y_i^{(1)}, y_i^{(2)}, \ldots, y_i^{(K)} \rangle, s_\alpha) \underset{Df}{=} \delta(y_i^{(K)}, \delta(y_i^{(K-1)}, \delta(\ldots \delta(y_i^{(2)}, \delta(y_i^{(1)}, s_\alpha))\ldots)$$

First of all, let us prove the following lemmas.

Lemma 1. If $\langle \pi', \pi'' \rangle \in K_{A_i}(S)$ and $\langle \pi'', \pi''' \rangle \in K_A(S)$, then $\langle \pi', \pi''' \rangle \in K_{A_{i+j}}(S)$.

Proof. Following the definition $s_\alpha \equiv s_\beta(\pi') \Rightarrow$

$$\Rightarrow (\forall y \in I^i)(\delta_i(y, s_\alpha) \equiv \delta_i(y, s_\beta)(\pi'')) \Rightarrow$$

$$\Rightarrow (\forall yz \in I^{i+j})(\delta_j(z, \delta_i(y, s_\alpha)) \equiv \delta_j(z, \delta_i(y, s_\beta))(\pi''')) \Rightarrow$$

$$\Rightarrow (\forall yz \in I^{i+j})(\delta_{i+j}(yz, s_\alpha) \equiv \delta_{i+j}(yz, s_\beta)(\pi''')) \Rightarrow$$

$$\Rightarrow \langle \pi', \pi''' \rangle \in K_{A_{i+j}}(S).$$

Corollary. Let A be an arbitrary automaton. Then for π on S and $K \geq 1$ we have $m_A(m_{A_K}(\pi)) \geq m_{A_{K+1}}(\pi)$ and $M_A(M_{A_K}(\pi)) \leq M_{A_{K+1}}(\pi)$.

Lemma 2. If $\langle \pi', \pi''' \rangle \in K_{A_{i+j}}(S)$, then there is a $\pi''(S)$ such that $\langle \pi', \pi'' \rangle \in K_{A_i}(S)$ and $\langle \pi'', \pi''' \rangle \in K_{A_j}(S)$.

Proof. First we shall show that if the identity $\pi'' = m_{A_i}(\pi')$ holds then $\langle m_{A_i}(\pi'), \pi''' \rangle \in K_{A_j}(S)$. Let $s_h \equiv s_K(m_{A_i}(\pi'))$. Let us suppose that there is a $y^\alpha \in I^j$

such that $\delta_j(y^\alpha, S_h) \not\equiv \delta_j(y^\alpha, S_k)(\pi''')$. Then for any $S', S'' \in S$ and $y \in I^i$ satisfying the conditions $\delta_i(y, S') = S_h$ and $\delta_i(y, S') = S_k$ we have $S' \not\equiv S''(\pi')$ (if this is not the case then we have a contradiction). Let us denote by $\pi_{i,j}$ a partition where beside one state blocks there is a block $\{S_i, S_j\}, S_i, S_j \in S$. It is not hard to see that $m_{A_i}(\pi')$ can be expressed in the form $\Sigma(\pi_{i,j})$, where $\pi_{i,j}$ is determined by the fact that there exist $S', S'' \in S$ and $y \in I_i$ which with $S' \equiv S'''(\pi')$ give $\delta_{A_i}(y, S') = S_i$ and $\delta_{A_i}(y, S'') = S_j$. From the above and $S_h \equiv S_k(m_{A_i}(\pi'))$ it follows that there are $S^{r_1}, S^{r_2}, \ldots, S^{r,4p} \in S$ and $y^{u_1}, y^{v_1}, y^{v_2}, \ldots, y^{up}, y^{vp} \in I_i$ such that we obtain

$$S^{r_1} \equiv S^{r_2}; \quad S^{r_3} \equiv S^{r_4}; \quad \ldots, S^{r,4p-1} = S^{r,4p} \qquad (1)$$

and

$$\delta_i(y^{u_1}, S^{r_1}) = S_h; \quad \delta_i(y^{u_1}, S^{r_2}) = S_h^1;$$
$$\delta_i(y^{v_1}, S^{r_3}) = S_h^1; \quad \delta_i(y^{v_1}, S^{r_4}) = S_h^2;$$
$$\delta_i(y^{u_2}, S^{r_5}) = S_h^2; \quad \delta_i(y^{u_2}, S^{r_6}) = S_h^3;$$
$$\delta_i(y^{v_2}, S^{r_7}) = S_h^3; \quad \delta_i(y^{v_2}, S^{r_8}) = S_h^4;$$

$$\delta_i(y^{up}, S^{r,4(p-1)+1}) = S_h^{2p-2}; \quad \delta_i(y^{up}, S^{r,4(p-1)+2}) = S_h^{2p-1};$$
$$\delta_i(y^{vp}, S^{r,4p-1}) = S_h^{2p}; \quad \delta_i(y^{vp}, S^{r,4p}) = S_k.$$

It is not difficult to notice that for any $y^\alpha \in I_j$ there exist $S_\alpha^1, S_\alpha^2, \ldots, S_\alpha^{2(p+1)}$ such that

$$\delta_{i+j}(y^{u_1} y^\alpha, S^{r_1}) = S_\alpha^1; \quad \delta_{i+j}(y^{u_1} y^\alpha, S^{r_2}) = S_\alpha^2;$$
$$\delta_{i+j}(y^{v_1} y^\alpha, S^{r_3}) = S_\alpha^2; \quad \delta_{i+j}(y^{v_1} y^\alpha, S^{r_4}) = S_\alpha^3;$$
$$\delta_{i+j}(y^{u_2} y^\alpha, S^{r_5}) = S_\alpha^3; \quad \delta_{i+j}(y^{u_2} y^\alpha, S^{r_6}) = S_\alpha^4;$$
$$\delta_{i+j}(y^{v_2} y^\alpha, S^{r_7}) = S_\alpha^4; \quad \delta_{i+j}(y^{v_2} y^\alpha, S^{r_8}) = S_\alpha^5;$$

$$\delta_{i+j}(y^{up}y^{\alpha}, s^{r,4(p-1)+1}) = s_{\alpha}^{2p-1};\ \delta_{i+j}(y^{up}y^{\alpha}, s^{r,4(p-1)+2}) = s_{\alpha}^{2p};$$

$$\delta_{i+j}(y^{vp}y^{\alpha}, s^{r,4p-1}) = s_{\alpha}^{2p+1};\ \delta_{i+j}(y^{vp}y^{\alpha}, s^{r,4p}) = s_{\alpha}^{2p+2}$$

and therefore $s_{\alpha}^{1} \equiv s_{\alpha}^{2p+2}(\pi''')$ for $\langle \pi', \pi''' \rangle \in K_{A_{i+j}}(S)$. Sc $\delta_j(y^{\alpha}, s_h) \equiv \delta_j(y^{\alpha}, s_k)$ (π''') which shows that we have reached a contradiction. This completes the proof.

Corollary. For any automaton A we have $m_A(m_{A_K}(\pi)) \leq$ $\leq m_{A_{K+1}}(\pi)$ and $M_A(M_{A_K}(\pi)) \geq M_{A_{K+1}}(\pi)$.

Proof. Whereas from lemma 2 we obtained that for any partition pair $\langle \pi, \pi_i \rangle \in K_{A_{K+1}}$ there exists a π' such that $\langle \pi, \pi' \rangle \in K_{A_K}$ and $\langle \pi', \pi_i \rangle \in K_A$ then $\langle m_{A_K}(\pi), \pi_i \rangle \in K_A$. Hence for any partition pair $\langle \pi, \pi_i \rangle \in K_{A_{K+1}}$ we get $m_A(m_{A_K}(\pi)) \leq \pi_i$ which implies $m_A(m_{A_K}(\pi)) \leq m_{A_{K+1}}(\pi)$. The proof of inequality $M_A(M_{A_K}(\pi)) \geq M_{A_{K+1}}(\pi)$ is analogous.

The corollaries 1 and 2 allow us to conclude.

Proof of theorem 1. For the proof it is sufficient to show that the iterative operators m_A^K and M_A^K are identical to the operators m_{A_K} and M_{A_K} from which by the theorem 3.1 [1] directly follows the statement of this theorem. The proof that these identities hold is carried out by induction. It is clear that for K=0 and K=1 they are true in the trivial way. There remains still to show that from the validity of these identities for a K follows their validity for K+1 as well. From the corollaries for lemmas 1 and 2 we directly get $m_A(m_{A_K}(\pi)) = m_{A_{K+1}}(\pi)$ and $M_A(M_{A_K}(\pi)) =$ $= M_{A_{K+1}}(\pi)$. Let us suppose now that $m_A^K(\pi) = m_{A_K}(\pi)$. We show that from this follows $m_A^{K+1}(\pi) = m_A(\pi)$. In fact, whereas $m_A^{K+1}(\pi) = m_A(m_A^K(\pi))$ and $m_{A_{K+1}}(\pi) = \Pi\{\pi_i | \langle \pi, \pi_i \rangle \in K_{A_{K+1}}(S)\}$

we have $m_A^K(\pi) = m_{A_K}(\pi) \Rightarrow m_A(m_A^K(\pi)) = m_A(m_{A_K}(\pi)) \Rightarrow$

$$\Rightarrow m_A^{K+1}(\pi) = \Pi\{\pi_i \mid \langle m_{A_K}(\pi), \pi_i \rangle \in K_A(S)\} =$$

$$= \Pi\{\pi_i \mid \langle \pi, \pi_i \rangle \in K^K_{A_{K+1}}(S)\} = m_{A_{K+1}}(\pi).$$

In an analogous way one can prove $M_A^K(\pi) = M_{A_K}(\pi) \Rightarrow M_A^{K+1}(\pi) = M_{A_{K+1}}(\pi)$. From this it follows for any $K \geq 1$ and $\pi(S)$ that $m_{A_K}^{K+1}(\pi) = m_A^K(\pi)$ and $M_{A_K}(\pi) = M$. The theorem is proved.

From the theorem 1 it follows that the properties of operators m and M are more general than those for automata with input word length 1 and represent universal properties of information flow in finite automata not depending on the length of input words. So the informational processes in finite automata are determinated not only by the operators m and M, but by a finite class of operators $m^{(0)}, m^{(1)}, m^{(2)}, \ldots, m^{(K)}$ and $M^{(0)}, M^{(1)}, M^{(2)}, \ldots, M^{(K)}$.

This enables us to give a more detailed analysis of the structural properties of finite automata. Particularly it serves for the more effective application on pair algebra for finding the checking sequences for automata and carrying out the iterative decomposition of automata.

References

1. Hartmanis J., Stearns R.E. Algebraic Structure Theory of Sequential Machines. New-York: Prentice-Hall, Inc. Englewood, Cliffs, 1966.

A. Keevallik, T. Lausmaa

Informational Operators of Finite Automata

Abstract

In this paper the technic of informational operators founded by J. Hartmanis and R. Stearns for analyzing finite automata is developed further. It is shown that properties of operators m and M showing the qualities of information flow in finite automata are valid also in case of their iteration.

A. Keevallik, T. Lausmaa

Lõpliku automaadi informatsioonilistest operaatoritest

Kokkuvõte

Artiklis on üldistatud tükelduse paari mõistet ja näidatud, et J. Hartmanise ja R. Stearnsi poolt tõestatud informatsioonilised võrratused kehtivad ka juhul, kui automaadi sisendsõnade pikkus on suurem kui üks. Saadud resultaadid avavad võimaluse automaadi struktuursete omaduste sügavamaks analüüsiks. Samuti osutub võimalikuks siduda automaadi kontrolleksperimentide leidmist tema dekomponeerimisega.

Nr. 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 62.507

B.Berkman, A.Sudnitsyn, J.Udre

Transforming VHDL Descriptions from Behavior to Structure

1. Introduction

With the growing use of CAD systems the problem of source data specification in design of digital circuits becomes actual whenever more. The choice of the right way for representing this kind of data gives the designer additional capabilities in verification of source data as well as the results of design. It also provides automation of some design steps, which were done manually in the recent systems. One of those steps is sharing a unit into the control part and the operational part. Our approach to this sharing is based on the methodology proposed by V.Glushkov [5]. Glushkov's essential contribution was to show that an algorithm can be implemented as a control automaton which cooperates with the operational automaton performing the data processing.

For the support of that approach the specification format has to meet some requirements:

- behavioral description supporting high-level abstract descriptions;
- features for explicit and/or implicit structure description;
-mechanisms providing hierarchical descriptions;
-any level of abstraction in object representation.

The presence of such features makes us able to use top-down design strategy with separating of control and data parts of a system. But the recent formats for digital unit specifications didn't meet these requirements. That's why many hardware makers, science labs and universities in the whole world develop new high-level languages for use in the areas of hardware specification, simulation and verification - so called Hardware Description Languages (HDL). A modern Hardware Description Language provides also some additional features for the design process control: exact timing and delay definition, layout, technology dependencies, etc.

2. The VHSIC Hardware Description Language

Design effort of the VHDL (Very high speed integrated circuits Hardware Description Language) [6] started in 1983 under contract of the US Department of Defense (DoD). The team of Intermetrics, IBM and Texas Instruments was

awarded the work to design VHDL and it's environment software. Now VHDL is one of the most advanced hardware description languages. It concernes both traditional description facilities (synchronous, asynchronous and mixed circuits, views range from gates to algorithms, combinational and sequential machines description etc.) and data abstraction features (data typing capabilities as the programming languages, Ada [4] for example, have).

The main important VHDL features are following:

a) the design entity concept: VHDL provides alternative design representations through separate mechanisms for system interface description and another one for establishing relations between interface and multiple system representations;

b) mixed concurrent and sequential behavioral description as against to prior HDLs which require separating them or even provide only one way of behavioral description;

c) the package concept (similar to Ada) is a unique feature for collecting together definitions and utility functions concerned with the specific problem;

Advanced VHDL facilites for the abstract data representation provide the most natural description for those, who deal with digital design - high-level program. Since all problems of that's implementation into real hardware would be solved by CAD, which accepts VHDL code as the source data. Therefore design transfer onto new technology basis become more easier.

However VHDL have some shortcomings, as any real language has. For example all VHDL statements are defined only on specific built-in types: "AND" — on BIT and BOOLEAN, "+" — on INTEGER and so on. It isn't difficult to write down some functions evaluating needed operations on any type of data, but the readability of code will be reduced by this way. The description of mixed sequential — concurrent behavior doesn't seem to be natural as well because encapsulating of sequential statements into a block of concurrent statements is provided but the opposite is prohibited (at least in VHDL 7.2). It could be done, of course, through definition of separate blocks with following activation of them by special signals. But this way can not to be accepted because of having to describe signals which are not actually presented in the unit.

3. Description of the Digital Units in VHDL and Separation of Control and Data Parts

Consider as example a specialized digital unit which performs an operation of computing the greatest common divisor (GCD) of two integers corresponding to

Euclid algorithm [9]. The gist of this algorithm is computing the remainder from division of the greater number by the less one and further exchanging the greater number with the less one and this less number with the division reminder. This converging process is looped until the division remainder is equal to zero. That means the termination of the algorithm with the current less number as the result (GCD). The program in VHDL which presents the behavioral description of the unit to be designed is shown in Fig.2. The description of the system interface is presented in Fig.1 where input and output signals are specified. Since the description of simultaneously executed operations is complicated inside the VHDL process statement, such operations are simply grouped into one line. Notice that in all VHDL descriptions we mark the VHDL keywords with bold font and with '--' beginning rows are comments.

```
entity EUCLID is
  port
        ( Start: in BIT;       -- Start signal
          OP1:in INTEGER;  -- The 1st operand bus
          OP2:in INTEGER;  -- The 2nd operand bus
          ANSW: out INTEGER;  -- Answer bus
          Ready:out BIT;       -- Answer-is-ready signal
        )
end EUCLID;
```

Fig.1. The interface description for specialized digital unit.

To synthesize a structural description of the unit from its behavioral description this last one should be firstly partitioned into control and operational (data) parts (or flows) [5,12].

The volume and the functions of the control part depends on the behavioral description as well as on the staff of the data part. The data part is specified by the set of operations presented in the behavioral description and by the set of basic elements which it will be implemented by. These basic elements may be either standard chips or macros in the case of VLSI design. Consider in our example the data part that is based upon some ALU which completes four arithmetic operations (addition, subtraction, multiplication and division) with registers R1 and R2 for storing the intermediate results and with control buses for data transfer (see the structure in Fig.3). The presence of such an ALU means actually that all basic

operations are implemented with certain element(s) of the structure and since a structural description of the data part is achieved. Notice that if this ALU chip or macro doesn't exist we need to synthesize it on the *next design step* basing upon its behavioral description and existing (or virtual) elements of *the lower level* — e.g. adders, shift registers and counters. It is obvious that it would in its turn lead to appearing *the control part of the lower level* and so on. Such a *top-down design* methodology is widely used when large software projects are developed [7].

```
architecture COMMON of EUCLID is
        process
        variable Temp1,Temp2: INTEGER;  -- Temp. variables
        variable C: INTEGER;            -- An accumulator
        begin
           wait for Start'Rising;  -- Waiting for the start
           Temp1:=OP1; Temp2:=OP2; C:=Temp1-Temp2;
           if C < 0 then           -- if A < B then
             Temp2:=OP1; Temp1:=OP2; -- exchange operands
           end if;
           while C != 0 loop
              C:=Temp1/Temp2;     -- Calculation
              C:=C*Temp2;         -- of the
              C:=Temp1-C;         -- reminder
              if C != 0 then
                Temp1:=Temp2; Temp2:=C;
              end if;
           end loop;
            ANSW <= Temp2; Ready <= '1'; -- Answ. output
        end process;
end COMMON;
```

Fig.2. The abstract description of specialized digital unit.

The VHDL description of the unit data part is shown in Fig.4. It is considered that R1 and R2 are Master-Slave registers that allows to exchange their contents during one clock cycle. It is also considered that "Start" is a strobe signal whose rising starts the unit. The main mechanism used in the description is the parallel signal assignment. The control part is presented there as a component with input

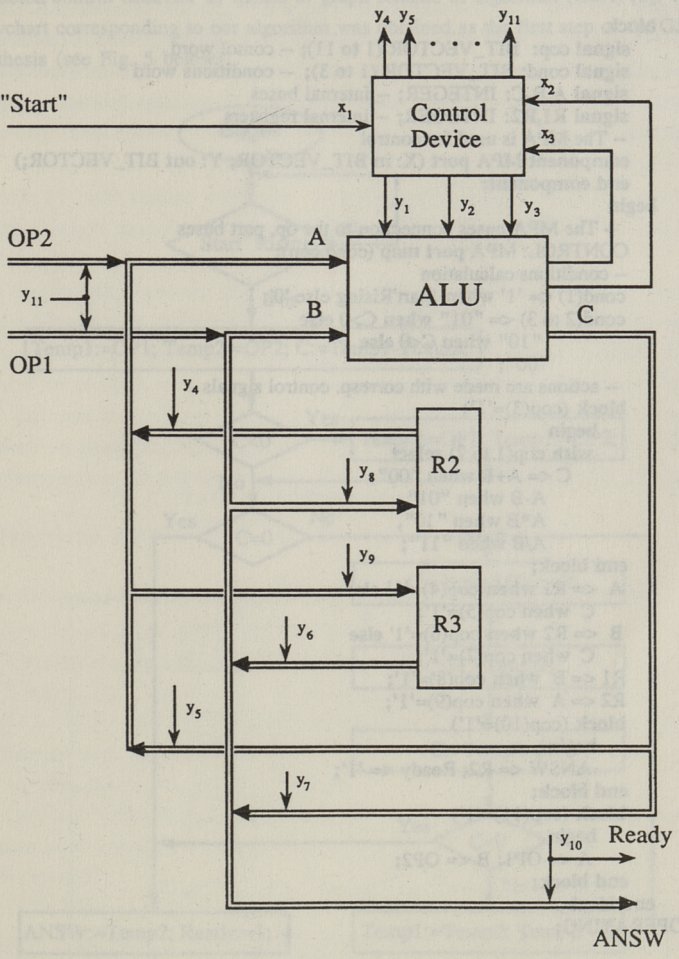


Fig.3. The structure of specialized digital unit.

and output buses. Via the input bus the control part obtains condition signals from the data part and via the output bus control signals from the control part proceed to the data part.

architecture OPERATING **of** EUCLID **is**

```
    block
      signal cop: BIT_VECTOR (1 to 11); -- contol word
      signal cond: BIT_VECTOR (1 to 3); -- conditions word
      signal A,B,C: INTEGER; -- internal buses
      signal R1,R2: INTEGER; -- internal registers
      -- The MPA is used for control
      component MPA port (X: in BIT_VECTOR; Y: out BIT_VECTOR;)
      end component;
    begin
      -- The MPA buses connection to the op. part buses
      CONTROL: MPA port map (cond,cop);
      -- conditions calculation
      cond(1) <= '1' when Start'Rising else '0';
      cond(2 to 3) <= "01" when C>0 else
                      "10" when C<0 else
                      "00";
      -- actions are made with corresp. control signals
      block (cop(3)='1')
        begin
        with cop(1 to 2) select
          C <= A+B when "00",
               A-B when "01",
               A*B when "10",
               A/B when "11";
      end block;
      A  <= R1 when cop(4)='1' else
        C when cop(5)='1';
      B  <= R2 when cop(6)='1' else
        C when cop(7)='1';
      R1 <= B when cop(8)='1';
      R2 <= A when cop(9)='1';
      block (cop(10)='1')
        begin
        ANSW <= R2; Ready <= '1';
      end block;
      block (cop(11)='1')
        begin
        A <= OP1; B <= OP2;
      end block;
    end block;
end OPERATING;
```

Fig.4. The operational (data) part description.

On every description level after the (regular) structure of data part is defined it is possible to extract the remaining control part from the current level behavioral description. Naturally this extracted control part description may be at first only behavioral one and the methods of finite automata synthesis are required for control part implementation. In this stage it is convenient to represent the

extracted control behavior by means of graph-scheme of algorithm (GSA) [2]. The flowchart corresponding to our algorithm was obtained as the first step of the GSA synthesis (see Fig. 5 below).

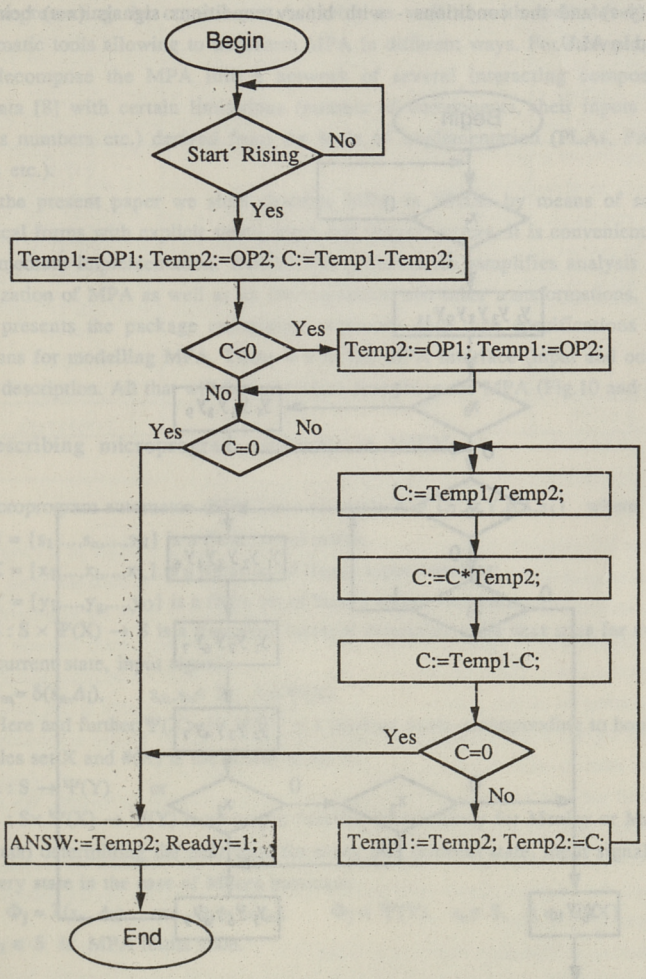


Fig.5 The flowchart for the LCD computation.

On this flowchart simultaneously executed statements are grouped into the common blocks. Fig.6 represents the GSA we got from this flowchart by replacing the computational statements (actions of ALU) with the corresponding control signals (y-s) and the conditions - with binary conditions signals (x-s) being generated in ALU.

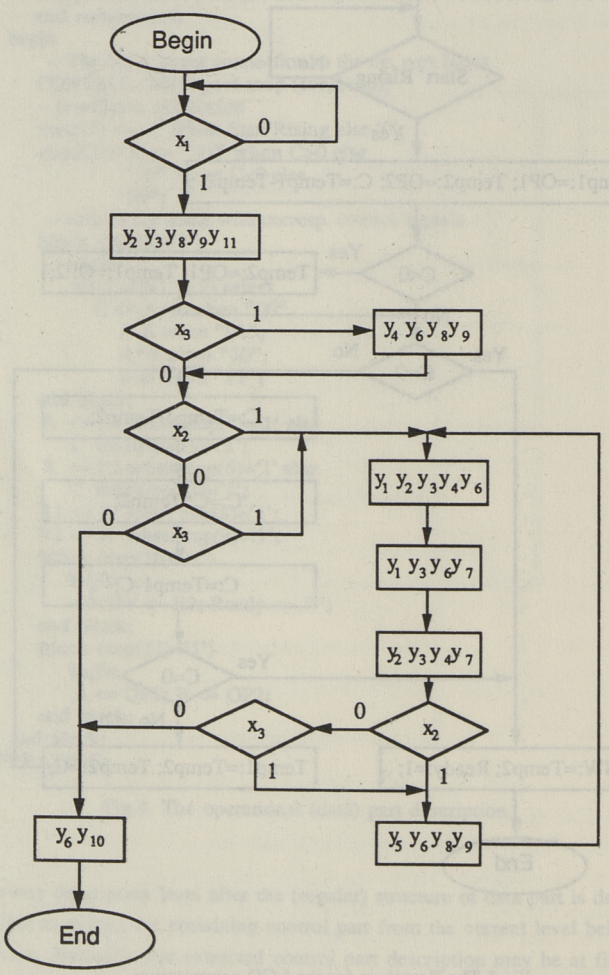


Fig.5 The flowchart for the LCD computation.

The GSA model well represents the control part behavior in terms of binary signals and the corresponding microprogram automata (MPA) [2] may be simply obtained from GSA. The choice of MPA model is concerned with its compactness when representing the control part behavior as well as with well developed mathematic tools allowing to transform MPA in different ways. For example, we may decompose the MPA into a network of several interacting component automata [8] with certain limitations (number of components, their inputs and outputs numbers etc.) derived from the basis of implementation (PLAs, PALs, ROMs etc.).

In the present paper we shall describe MPA in VHDL by means of some canonical forms with explicit states, input and output vectors. It is convenient for its immediate implementation. Besides this presentation simplifies analysis and optimization of MPA as well as its decomposition and other transformations. The Fig.9 presents the package containing necessary data type specifications and functions for modelling MPA. There is also the MPA interface (input and output ports) description. All that will be used when describing the MPA (Fig.10 and 11).

4. Describing microprogram automata in VHDL

Microprogram automaton (MPA) is a six-tuple $A = \{S,X,Y,\delta,\lambda,s_1\}$ where

1. $S = \{s_1,...,s_n,...,s_N\}$ is a finite set of states;
2. $X = \{x_1,...,x_l,...,x_L\}$ is a finite set of binary input variables;
3. $Y = \{y_1,...,y_g,...,y_G\}$ is a finite set of binary output variables;
4. $\delta : S \times \Psi(X) \rightarrow S$ is a transition function determining the next state for every pair (current state, input signal):

$s_m = \delta(s_n,\Delta_i)$, $s_m,s_n \in S$, $\Delta_i \in \Psi(X)$.

Here and further $\Psi(X)=\{0,1\}^{\#(X)}$ is a boolean room corresponding to boolean variables set X and #(X) is the power of set X.

5. $\lambda : S \rightarrow \Psi(Y)$ or

$\lambda : S \times \Psi(X) \rightarrow \Psi(Y)$ is an output function (respectively for Mealey or Moore automata) determining the next state for every pair (current state, input signal) or for every state in the case of Moore automata:

$\Phi_j = \lambda(s_n, \Delta_i)$ or $\Phi_j = \lambda(s_n)$; $\Phi_j \in \Psi(Y)$, $s_n \in S$, $\Delta \in \Psi(X)$

6. $s_1 \in S$ is MPA initial state.

Present state	Next state	Input condition	Output value
1	1	$\neg x_1$	—
	2	x_1	$y_2y_3y_8y_9y_{11}$
2	1	$\neg x_2 \neg x_3$	y_6y_{10}
	3	x_3	$y_4y_6y_8y_9$
	4	$x_2 \neg x_3$	$y_1y_2y_3y_4y_6$
3	1	$\neg x_2 \neg x_3$	y_6y_{10}
	4	x_2	$y_1y_2y_3y_4y_6$
	4	x_3	$y_1y_2y_3y_4y_6$
4	5	1	$y_1y_3y_6y_7$
5	6	1	$y_2y_3y_4y_7$
6	1	$\neg x_2 \neg x_3$	y_6y_{10}
	7	x_2	$y_5y_6y_8y_9$
	7	x_3	$y_5y_6y_8y_9$
7	4	1	$y_1y_2y_3y_4y_6$

Fig.7. The MPA transition table.

In this definition: N is the number of MPA states, L is the number of it's inputs, G is the number of it's outputs. The transition table of the MPA obtained from our GSA is shown in Fig.7. The transition table is a well-known form for representing MPAs. It consists of four columns: the first two are present state column and target state column. The third is the column of binary conjuctions of input variables corresponding to certain present state-target state pairs. The fourth is the column of output signals associated with the current state (in case of Moore automata) or with the state-to-state transition (in case of Mealy automata).

We will use two methods for MPA program implementation in VHDL. We call them *compilation method* and *interpretation method.*

The compilation method is based on the correspondense between the automata states and the program blocks with unique labels. Each of those blocks consist of the following operations:

a) reading and latching of the input word;

b) finding the next state by value of input word, jump to label corresponding to the new state with simultaneous output word generation;

The most important part of this algorithm is, of course, search for the next state because it means the evaluation of the Boolean functions set. If s_m is the present

state and $\{s_r\}$ is the set of the next ones where transitions from s_m are possible ($r \in R_m \subseteq \{1,..,N\}$) then there is the set of L-variable Boolean functions $F = \{f_{rm} \mid r \in R_m\}$ (L is the number of MPA binary inputs). It is necessary to evaluate which of these functions has value "true" for a given input combination Δ_i. The well-known method of solving this problem is representation of system F in the form of binary decision diagram [1]. The binary decision diagram H is labelled acyclic binary diagram (binary tree in other words), where each nonterminal node has two outcoming arcs labelled x_j and $\neg x_j$, $x_j \in X_m$. X_m is the set of input variables which are essential for the state s_m. Any leave of this tree corresponds to the state where the transition has to be made if the input combination Δ_i agrees to the values of variables marking the way from the root to this leave. A section of MPA transition table and corresponding binary decision diagram are given in Fig.8. Input variables x_2 and x_3 are essential there.

3	1	$\neg x_2 \neg x_3$	$y_6 y_{10}$
	4	x_2	$y_1 y_2 y_3 y_4 y_6$
	4	x_3	$y_1 y_2 y_3 y_4 y_6$

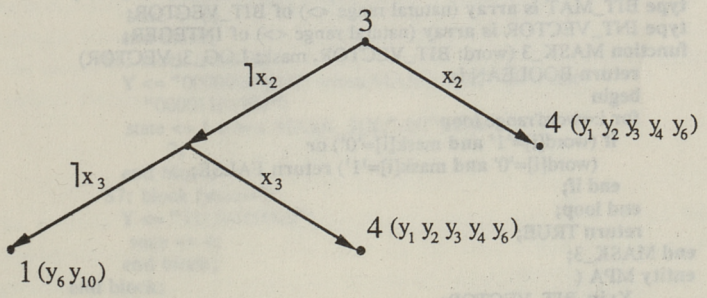


Fig.8. The MPA transition table fragment with corresponding binary decision diagram.

The use of essential inputs only is the main advantage of this method, because the set of essential inputs is just a part of the whole set of inputs. Actually, in case of using all inputs when consructing binary tree, the number of nodes in the tree (i.e. the number of statements in the program implementation — n_c) would be equal to

$$n_c = 2^{L+1} - 1 \qquad (1)$$

The assymptotic estimation of this number is however [10]:

$$n_c \approx \frac{2^L}{L} \qquad (2)$$

When implementing a real MPA this value would even be less. Using of binary trees with branching on single input variable needs, however, up to L_m bit operations, where the L_m is the number of inputs which are essential for the current MPA state (s_m). This method is convenient for the microprocessor implementation of binary programs because it doesn't depend on the relation between the width of MPA input word and the width of the microprocessor bus. When constructing a VHDL description for the binary programs the whole input word masking is certainly more appropriate. But in such a case the problem of essential bits outlinig arises. For the solving of this problem we'll use three-valued logic vectors (0,1,-), where '-' indicates that the corresponding input bit is not essential for a given transition. Because of the abovementioned shortcomings of VHDL statements the MASK_3 function has been developed which performs masking a binary word with three-valued vector (see Fig.9).

```
package mpa_supp is
    type LOG_3 is ('0','1','-');
    type LOG_3_VECTOR is array (natural range <>) of LOG_3;
    type LOG3_MAT is array (natural range <>) of LOG_3_VECTOR;
    type BIT_MAT is array (natural range <>) of BIT_VECTOR;
    type INT_VECTOR is array (natural range <>) of INTEGER;
    function MASK_3 (word: BIT_VECTOR, mask: LOG_3_VECTOR)
        return BOOLEAN is
        begin
        for i:=word'range loop
          if (word[i]='1' and mask[i]='0') or
            (word[i]='0' and mask[i]='1') return FALSE;
          end if;
        end loop;
        return TRUE;
    end MASK_3;
    entity MPA (
         X: in BIT_VECTOR;
         Y: out BIT_VECTOR;
        )
    end MPA;
end mpa_supp;
```

Fig.9. The package supporting the MPA description in VHDL.

```
use mpa_supp;
architecture COMPILE of MPA is
      block signal state: INTEGER;
      initialize state to 1;
```

```
    begin S1: block (state=1)
          Y <= "00000000000" when MASK_3(X,"-00") else
               "01100001101";
           state <= 1 when MASK_3(X,"0--") else
                2;
          end block;
         S2: block (state=2)

          Y <= "00000100010" when MASK_3(X,"-00") else
                "00010101100" when MASK_3(X,"--1") else
               "11110100000";
           state <= 1 when MASK_3(X,"-00") else
                "00010101100" when MASK_3(X,"--1") else
               "00010101100";
          end block;
         S3: block (state=3)
          Y <= "00000100010" when MASK_3(X,"-00") else
            "11110100000";
           state <= 1 when MASK_3(X,"-00") else
                4;
          end block;
         S4: block (state=4)
          Y <= "10100110000";
           state <= 5;
          end block;
         S5: block (state=5)
          Y <= "01110010010";
           state <= 6;
          end block;
         S6: block (state=6)
          Y <= "00000100010" when MASK_3(X,"-00") else
            "00001101100";
           state <= 1 when MASK_3(X,"-00") else
                7;
          end block;
         S7: block (state=7)
          Y <= "11110100000";
           state <= 4;
          end block;
    end block;
end COMPILE;
```

Fig.10. The MPA description using the compilation method.

The VHDL description of the MPA using the compilation method is represented in Fig.10. One can see that the VHDL statements in this description exactly correspond to the rows of MPA table (Fig.7).

```
use mpa_supp;
architecture INTERPRET of MPA is block
```

```
        signal X_temp: BIT_VECTOR (1 to 3);
        variable line_of_table: INTEGER;
        constant b_lines: INT_VECTOR := (1,3,6,9,10,11,14);
        constant n_state: INT_VECTOR :=
             (1,2,1,3,4,1,4,4,5,6,1,7,7,4);
        constant Xs: LOG_3_MAT :=
            ("0--","1--",
             "-00","--1","-10",
             "-00","-1-","--1",
              "---",
              "---",
             "-00","-1-","--1",
              "---");
        constant Ys: BIT_MAT :=
            ("00000000000","01100001101",
             "00000100010","00010101100","11110100000",
             "00000100010","11110100000","11110100000",
             "10100110000",
             "01110010000",
             "00000100010","00001101100","00001101100",
             "00001101100");
    begin
        IN_STATE: process
             begin
             wait until state /= 0;
              line_of_table := b_lines(state);
             X_temp <= X;
              state <= 0;
        end process;
        TRANS: process
            begin
            wait until state = 0;
            if MASK_3(X_temp,Xs(line_of_table)) then
               Y <= Ys(line_of_table);
                 state <= n_state(line_of_table);
             else
              line_of_table := line_of_table+1;
            end if;
        end process;
    end block;
end INTERPRET;
```

Fig.11. The MPA description using the interpretation method.

The interpretation method is based on representation of MPA in the form of data structure. This structure in the most cases of implementation is an array of word pairs (state word, output word). For performing a transition the current input word is to be added to the current state word to get the address of the pair (next state word, output word) in the array. This method is the fastest one, but the storage volume necessary to that is huge [10]:

$$n \sim N * 2^{L}(G + \log_2 N) \qquad (3)$$

In the case of a microprocessor implementation this volume can be reduced owing to the abovementioned fact that the number of essential bits for computing transitions from a given state s_m is only a small part of the MPA total inputs number (L). Therefore optimized layout of input bits onto the microprocessor ports would save core space very much. On the other hand when VHDL is used for interpretative description, it's impossible to use such trick (unless we describe microprocessor system for implementing of needed behavior). That's why we represent the MPA transition table immediately in the following form (the variable names correspond to Fig.11):

a) n_state array, whose length is equal to the length of the transition table's and which consists of next state numbers;

b) Xs array of the three-valued logic vectors for masking input words; each vector corresponds to the state number (from n_state) where transition is to be made when the result of masking is true;

c) b_lines array whose i-th element (i=1,...,N) is equal to the number of the first element of n_state belonging to the i-th section of the MPA transition table.

d) Ys array of output words assosiated with the lines of MPA table (the case of Mealy automata) i.e. with the elements of n_state and Xs;

The interperting functions are performed by two processes working in turns. The first process searches for the first line of section corresponding to the current state and the second one finds out next automata state due to the value of input word and outputs the binary vector associated with evaluated transition.

5. MPA implementation

The possible implementations of digital unit control part (represented by microprogram automata) differ depending on the chosen basis (random or programmable logic etc.). Different kinds of programmable logic arrays (PLA) [11, 12] are used today very intensively for control part implementation owing to their regular structure, good testability and relative simplicity of using. In particular when designing VLSI PLAs are used in prototype systems (in the form of standard or semicustom chips) as well as in target VLSI chip. If no restrictions of PLA size are presented then MPA may be implemented by a single PLA in a trivial way. The Fig. 12 shows the contents of the single PLA that implements our microprogram automaton. This implementation is obtained directly from the MPA

transition table implying that the memory elements are Delays and the states were encoded as follows:

1 - 000; 2 - 001, 3 - 010, 4 - 011, 5 - 100; 6 - 101. 7 - 110.

Present State	Inputs	Next State	Outputs
0 0 0	0 — —	0 0 0	0 0 0 0 0 0 0 0 0 0
0 0 0	0 — —	0 0 1	0 1 1 0 0 0 0 1 0 0 1
0 0 1	— 0 0	0 0 0	0 0 0 0 0 1 0 0 0 1 0
0 0 1	— — 1	0 1 0	0 0 0 1 0 1 0 1 1 0 0
0 0 1	— 1 0	0 1 1	1 1 1 1 0 1 0 0 0 0 0
0 1 0	— 0 0	0 0 0	0 0 0 0 0 1 0 0 0 1 0
0 1 0	— 1 —	0 1 1	1 1 1 1 0 1 0 0 0 0 0
0 1 0	— — 1	0 1 1	1 1 1 1 0 1 0 0 0 0 0
0 1 1	— — —	1 0 0	1 0 1 0 0 1 1 0 0 0 0
1 0 0	— — —	1 0 1	0 1 1 1 0 0 1 0 0 0 0
1 0 1	— 0 0	0 0 0	0 0 0 0 0 1 0 0 0 1 0
1 0 1	— 1 —	1 1 0	0 0 0 0 1 1 0 1 1 0 0
1 0 1	— — 1	1 1 0	0 0 0 0 1 1 0 1 1 0 0
1 1 0	— — —	0 1 1	1 1 1 1 0 1 0 0 0 0 0

Fig.12. Contents of the obtained PLA.

The description of PLA on VHDL is quite simple and one can find the example of it in [11].

Very often however the control unit can not be implemented by a single PLA because of the latter restrictions (especially in the case of standard PLAs). Among the different restrictions the number of binary inputs is the most significant and always causes the necessity of decompositions. Usually the subject of such a decomposition is a single "abstract" PLA implementing the control part and the game of decomposition is to obtain a network consisting of possibly fewer number of PLAs meeting given restrictions. But the decomposition on the PLA level can not be very usefull because some possibilities are irretrievably lost by the choice of the certain type of memory and state codes.

An alternative approach is used in [3] where the method for microprogram automata decomposition is proposed that results in a special kind of Mealy

automata networks with components meeting PLA restrictions. This method can be used also for implementations in the basis of programmable logic devices other than PLA if the same restrictions are essential for them.

Conclusion

In this paper we introduced the method for transforming VHDL behavioral descriptions of digital circuits to the structural descriptions. The top-down methodology of extracting control flow from the behavioral descriptions is proposed with further transforming the control part into the form of microprogram automata and implementing this latter.

While the method is primarily developed for the VHDL descriptions it can be easily adopted for some other hardware description language providing that it has facilities for presenting both behavior and structure. The transformations of the description of specialized digital unit for the greatest common divisor calculation are presented as an example.

References

1. Akers S.B. Binary Decision Diagrams // IEEE Trans. on Comput. Vol. C-27. June 1978. P. 509-516.
2. Baranov S.I. Microprogram Automata Design (graph-schemes and algorithms). Second edition (In Russian). Leningrad: Energy, 1979.
3. Berkman B. Microprogram Automata Decomposition into Networks of Mealey Automata // Tallinn Technical University Proceedings. No. 696. 1989. P. 67-75.
4. Gehani. N. An Advanced Introduction Including Reference Manual for the Ada Programming Language. N.-Y.: Prentice-Hall. 1983.
5. Automata theory and formal microprogram transformation. // Kibernetika. Vol. 1. 1965. P. 1-9.
6. IEEE Design & Test. Special issue on VHDL. Apr. 1986.
7. Jackson M.A. Principles of Program Design. N.-Y.: Academic Press. 1975.

8. Jaokobson G., Keevallik A., Leis P. Some Aspects of the Construction of Microprogram Automata Networks // Proceedings of the Second International Symposium on Discrete Systems. Dresden, 1977. P. 120–128.
9. Knuth D. The Art of Computer Programming. Vol. 2: Seminumerical Algorithms. N.-Y.: Addison-Wesley. 1963.
10. Lee C.Y. Representation of Switching Circuits by Binary Decision Diagrams. // The Bell System Technical Journal. Vol. 38. No. 4. 1959. P. 985–999.
11. Lipsett R., Shaefer C.F., Ussery C. VHDL: Hardware Description and Design. Norwell: Kluwer Academic Publishers. 1989.
12. Southard J.R. Mac Pitts: An Approach to Silicon Compilation. // IEEE Computer. Vol. 16. Dec. 1983. P. 74–82.

Nr. 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.32

V. Alango, T. Kont, R. Ubar

NEW TEST DESIGN TECHNIQUES FOR FAULT DETECTION IN DIGITAL OBJECTS

Keywords. Integrated circuits; microprocessors; automatic testing; test generation; simulation; failure detection; alternative graphs.

Introduction

The densities of digital devices have been doubling each year during the two last decades. The growing complexity of digital objects like VLSI, ASIC, microprocessor based devices has increased the need to develop new efficient and cost-effective test design methods. Description techniques which specify the actual implementation of these objects on the gate level are becoming practically unfeasible.

Another possible way to describe digital objects and to design tests for them is using the implementation-free functional level (Su, 1984). Earlier published works on functional testing mostly concentrate on microprocessor testing using hardware-description languages, graphs or binary decision diagrams (BDD). Surveys of different functional testing techniques for digital LSI/VLSI systems are given by Su (1984) and Abadir (1986).

There is a need for describing complex digital objects for test design purposes at different representation levels, which leads to using different languages, different fault models and therefore also to using different test generation methods. The result is that the whole test design process turns to be quite troublesome and expensive. On the other hand, traditional functional descriptions do not repre-

sent the structural aspects and implementation details of the object and therefore difficulties of measuring the fault cover of tests arise.

From this situation the motivation results to develop a new model for representing a wide class of digital objects and a new fault model which must be a simple one (like the traditional stuck-at fault model for gate-level descriptions), but covering a large class of practically important physical failures in digital devices.

In this paper a convenient way is proposed to specify the logical performance of a digital object by means of alternative graphs (AG). AG, introduced for testing purposes by Ubar (1976), serve as a concise description of the function involved and provide a straightforward means for automatic test design. AG differ from BDD (Akers, 1978) in their capability to represent the structure of the original object and also its functions at different representation levels. On the basis of AG, a new fault model is developed which can be regarded as a generalization of the classical stuck-at fault model for the higher level representations. The same uniform model of AG can be used to support a wide class of test design tasks: fault activation and propagation, two- or multi-valued simulation, fault cover analysis, testability analysis, fault detection probability calculation, fault localization. Traditionally, for these tasks different models and corresponding different model libraries must be used. The same uniform AG-approach is used for designing tests at different levels like transistor or logical circuits, microprogram or instruction set levels.

A l t e r n a t i v e G r a p h s

Let us consider a digital device as a set of Boolean functions $z_k = f_k(\underline{z}_k)$, where $Z = \{z_k\}$ is a set of Boolean variables representing inputs, outputs and interior points in the circuit. Each function f_k describes the behavior of the k-th component (module or subcircuit) of the device and will be represented by an AG G_k. G_k is an oriented graph $G_k = (M_k, \Gamma_k, Z_k)$ where M_k is a set of nodes, Γ_k is a relation on M_k, where $\Gamma_k(m)$ denotes a subset of successors of

the node m and $Z_k \subseteq Z$ is a set of arguments of the function f_k. Each node $m \in M_k$ is marked by a variable z_i (or inverted variable $\bar{z}_i$).

Let us define a parametric relation $\Gamma_k(m,z(m))$ which determines for each value of the node variable $z(m)$ exactly one successor $m^e = \Gamma_k(m,e)$, $e \in \{0,1\}$, $m^e \in \Gamma_k(m)$, for the node m. Each pattern $\underline{z}_k^t$ for $\underline{z}_k$ determines a transitive closure of mapping $\Gamma_k(m,z(m))$ as a path $l_k(\underline{z}_k^t) = \hat{\Gamma}_k(m^*, \underline{z}_k^t)$ from the starting node $m^* \in M_k$ up to some terminal node $m^T \in l_k(\underline{z}_k^t)$, where $\Gamma_k(m^T, z(m^T)) = \emptyset$. Let us say that G_k represents a function $z_k = f_k(\underline{z}_k)$ if the equation $f_k(\underline{z}_k) = z(m^T)$ takes place for each pattern of $\underline{z}_k$ and for the corresponding path $l_k(\underline{z}_k) = \hat{\Gamma}_k(m^*, \underline{z}_k)$ with $m^T \in l_k(\underline{z}_k)$.

Let us define AG for Boolean functions AND, NAND, OR, NOR, NOT as elementary AG (EAG). Figure 1 shows some EAG. The mapping $\Gamma_k(m,e)$ for nodes m is given by branches in Fig. 1, to the right for $e=1$, downwards for $e=0$.

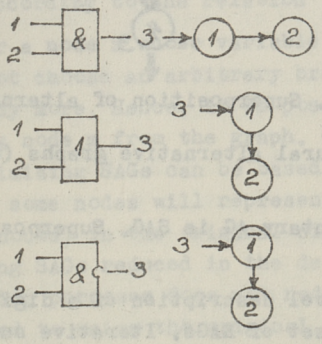


Fig. 1. Elementary alternative graphs

Structural Alternative Graphs

Let us define the following operation as the superposition of AGs. Consider two AGs G_k and G_i and the corresponding functions $z_k = f_k(\underline{z}_k)$, $z_i = f_i(\underline{z}_i)$, where $z_i \in Z_k$. The superposition of these functions gives $z_k = f_k(\underline{z}_k, f_i(\underline{z}_i))$. Analogically, we can replace all the nodes in G_k marked

with z_i (or $\bar{z}_i$) by G_i. Suppose, there is a node m in G_k marked by z_i, that has successors m^1, m^0. Define two sets of nodes for the graph G_i: $M^{T,e} = \{ m \mid m \in M_i \ \& \ \Gamma i(m,e)=\emptyset \}$, where $e \in \{0,1\}$.

The superposition of AGs, i.e. the replacement of m by G_i, lies in connecting all the branches which enter $m \in M_k$ with the node $m^* \in M_i$ and in creating the mappings $\Gamma_k (m,e) = m^e$, $m^e \in M_k$, for all the nodes $m \in M^{T,e} \subseteq M_i$ for both $e \in \{0,1\}$. An example of superposition of AGs is shown in Fig. 2.

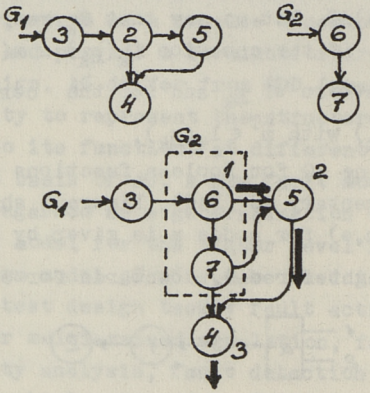


Fig. 2. Superposition of alternative graphs

Define now structural alternative graphs (SAG) recursively as follows.

Definition. Elementary AG is SAG. Superposition of SAGs gives also SAG.

The gate level description of a digital device can be represented by a set of EAGs. Iterative superposition of AGs in this set leads to compressing the AG-model. Each subcircuit can be represented in the extreme case always only by one SAG.

The following properties of SAGs are important.

1. Each node in a SAG represents some path in the original circuit. Hence, we can rise from the gate level to the higher levels without loosing accuracy of representing gate-level stuck-at faults. The task of test generation

for structural stuck-at faults on some path in a circuit can be substituted by the task of test generation for the corresponding node of AG. Some examples of representing paths in a digital circuit by the corresponding nodes of AG are depicted in Fig. 3.

2. For SAGs the following holds:

$$\forall m \in M_k: \; l(m^e, m^{T,\bar{e}}) \cap l(m^{\bar{e}}, m^{T,e}) \neq \emptyset, e \in \{0,1\} \qquad (1)$$

where $l(m_1, m_2)$ is a path on the AG from m_1 up to m_2, $m^e \in \Gamma_k(m,e)$ and $m^{T,e} \in M^{T,e}$. For example, if in Fig. 2 on the AG G_1 the path $l = (1,2,3)$ is activated with $z_6=1$, $z_5=0$, $z_4=0$, we could activate no additional path any more from the node 1 downwards, which would lead to some terminal node m^T with $z(m^T) = 1$. Using the property (1) helps us to reduce the area of searching for solutions during the test generation on the AG-model.

Optimization of SAGs. Paths on SAGs can consist of nodes which are marked by the same variable. Activating a path means setting values $e \in \{0,1\}$ to the variables $z(m)$ along the path according to the relation $\Gamma_k(m, z(m) = e) = m^e$. If we enter a node m whose variable $z(m)$ has been set already, we cannot choose an arbitrary branching direction from this node any more. Hence, it is possible to exclude the corresponding node m from the graph. On this fact the procedure of optimizing SAGs can be based. As a result, in optimized graphs some nodes will represent multiple paths (reconvergent fanouts) in the original circuit. It is simple to show that using SAGs reduced in the described way for the test generation purposes does not reduce the fault coverage, compared to using the original AGs.

For example, in Fig. 3 the following is illustrated: a digital circuit (the counterexample of Schneider (1967)), the corresponding SAG with 12 nodes, where each node represents a single path in the circuit and an optimized SAG with 7 nodes, where the nodes a and f represent multiple paths in the circuit.

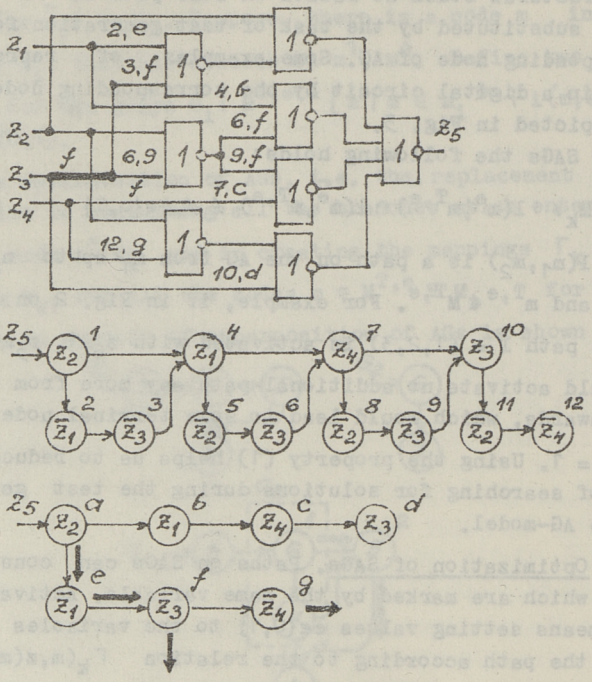


Fig. 3. Test generation for logical circuits on alternative graphs

Functional Alternative Graphs

Another way to generate AGs rests in using implementation-free descriptions of digital devices (Boolean expressions, truth tables, etc.). To create AGs, we can use the methods developed for BDDs (Akers, 1978). Since AGs obtained on the basis of functional implementation-free descriptions cannot represent the structure of the given device, it is appropriate to call this class of AG functional AG (FAG). In general, FAGs afford more concise description compared to SAGs and therefore they can be used successfully for generating sensibility conditions for fault propagation between inputs and outputs in modules considered as black-boxes. The role of SAGs remains in giving the possibility of representing and activating implementation-dependent faults

(or fault classes, in the case when paths are considered) in a given module (or subcircuit). So, a dynamic combination of FAGs and SAGs can contribute to effective test generation for large digital devices.

Test Design on Alternative Graphs

Simulation

The simulation process on AGs is equivalent to walking along a path in accordance with the mapping $\Gamma_k(m,z(m)=e) = {}_m e$ at given values of $z(m) \in Z_k$. We call the path traversed in such a way an activated path. As a result of the simulation, we reach a terminal node m^T where $\Gamma_k(m^T,z(m^T)) = \emptyset$. The value of the function f_k represented by AG for the given pattern $\underline{z}_k^t$ will be equal to the value of $z(m^T)$.

The upper bound of the number of steps in the simulation process (the number of traversed nodes) is $|M_k|$. The parallel simulation of a set of patterns is also possible. The length of this process is equal to $|M_k|$.

Fault Cover Analysis

Let a path l in G_k from starting node m^* up to some terminal node $m^{T,e}$ be activated by a given test pattern $T_i=\underline{z}_k^i$. Here $\underline{z}_k^i$ is a vector of values assigned to the vector $\underline{z}_k$. Let $M(l) \subseteq M_k$ be the set of nodes traversed on the path l. It is easy to understand that at the given test pattern T_i the faults at nodes $m \in M_k \setminus M(l)$ have no influence on the value of $z(m^{T,e})$ calculated for the function f_k on the graph G_k. On the other hand, for all other nodes $m \in M(l)$, there exists a test for a fault $z(m) \equiv \bar{D}$ at a node m if $\Gamma_k(m,z(m)) = m^D \in M(l)$, where $D \in \{0,1\}$ and if 1) $\Gamma_k(m,\bar{z}(m)) = \emptyset$ or 2) there exists another activated path $l' = \hat{\Gamma}_k(m^{\bar{D}},\underline{z}_k^i)$ from $m^{\bar{D}}$ up to another terminal node $m^{T,\bar{e}}$, where

$$z(m^{T,\bar{e}}) \neq z(m^{T,e}). \qquad (2)$$

The procedure described above is actually a new method for calculating Boolean derivatives. It is not difficult to understand that

$$z(m^{T,\bar{e}}) \neq z(m^{T,e}) \iff \partial z_k / \partial z(m) = 1. \qquad (3)$$

Let $R(T_i)$ be a set of faults covered by the test T_i. All these faults are related to the set of nodes M(1). The procedure of the fault cover analysis checks successively for all nodes $m \in M(1)$ if the conditions above are fulfilled. The upper bound of the number of steps in the analysis process is $|M_k|$. The length of the parallel simulation process for a set of patterns is equal to $|M_k|$.

The advantage of the method proposed here for the fault cover analysis lies in reducing the set of faults for which the calculating (3) is needed - only nodes $m \in M(1)$ have to be considered.

As an example, let us see Fig. 3. For the test pattern $T_i = 0111$ (z_1, z_2, z_3, z_4) we have $M(1) = \{1,4,5\}$ and $z(m^{T,o}) = \bar{z}_2 = 0$, where $m^{T,o} = 5$ is the terminal node reached by simulation. Only the node $4 \in M(1)$ will be tested (the fault $z_1 \equiv 1$) by this pattern, since there exists a path $l' = (7,10)$ with $z(m^{T,l}) = z_3 = 1$.

The procedure described can be efficiently used in deductive fault simulation in the case of using complex primitives.

Multi-Valued Simulation

The correctness of test sequences for sequential digital devices is proved by multi-valued simulation, where the alphabet of signals consists of more than two signals. For example, in addition to 0 and 1, there are often monotonic transitions considered: ε: $0 \to 1$, h: $1 \to 0$. The indeterminacy of signals is usually denoted by letter x.

The drawbacks of known methods are: 1) simulation is carried out on the gate-level, which leads to labour-consum-

ing procedures; 2) only gates with two inputs are allowed; 3) a dedicated multi-valued model library must be used; 4) changing the alphabet of signals leads to the need for a new model library.

These drawbacks can be removed by using AG models: AG represent gate-level information but in the more concise form and they remain invariant when using different sets of signal values for simulation.

The multi-valued simulation on AG lies in the concept of maximum of Boolean derivatives (Voolaine, 1982):

$$\delta(m) = \max_{z(m_i)} \{\partial z_k / \partial z(m)\}, \qquad (4)$$

where $m, m_i \in M_k$ and $m \neq m_i$. The function (4) can be regarded as the generalization of Boolean derivatives.

Consider an alphabet $A = \{0,1,x,\varepsilon,h\}$. The following can be generalized for extended cases of signal alphabets.

The calculation of (4) will be carried out using iteratively the recursive function

$$f(m) = F(z(m), f(m^1), f(m^0)). \qquad (5)$$

For $m \in M^T$ we have $f(m)=z(m)$. The value of $f(m^*)$ is to be searched. As special cases we have

$$f(m)=F(z(m)=e, f(m^1), f(m^0))=f(m^e),\ e \in \{0,1\},$$

$$f(m)=F(z(m)=x, f(m^1), f(m^0))=\begin{cases} x, \text{if } f(m^1) \neq f(m^0), \\ e \text{ otherwise} \end{cases}$$

The cases for $z(m)=\varepsilon$, h are shown in Table 1.

Table 1

Multi-Valued Simulation

$f(m^0)$ \ $f(m^1)$	ε	0	1	ε	h	x
	0	0	ε	ε	x	x
	1	h	1	x	h	x
	ε	x	ε	ε	x	x
	h	h	x	x	h	x
	x	x	x	x	x	x

$f(m^0)$ \ $f(m^1)$	h	0	1	ε	h	x
	0	0	h	x	h	x
	1	ε	1	ε	x	x
	ε	ε	x	ε	x	x
	h	x	h	x	h	x
	x	x	x	x	x	x

As an example, consider the simulation of the pattern T_i =

=ε000 (z_1, z_2, z_3, z_4) for the circuit in Fig. 3. The most important steps of the procedure are illustrated as a tree of traversing in Fig. 4. Here l_i are paths traversed by 2-valued simulation: l_1=(1,2), l_2=(3,4), l_3=∅, l_4=(7,8,9, 10,11,12), l_5=(5,6,l_4). The function (5) is used iteratively at the nodes 4 and 2: f(4)=F(ε,1,1) = 1, f(2)=F(h,1,0)= =h. So, the value z_5=h will be found at the node 2.

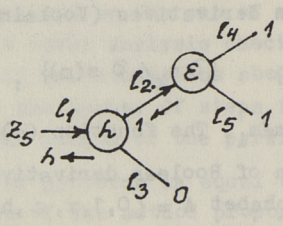


Fig. 4. Multivalued simulation

Test Generation

Path activation is the reverse process to simulation. To activate a path l_k between m_1 and m_2 in G_k, means to choose the proper values for the variables in Z_k(to choose a pattern $\underline{z}_k^t$), so that $l_k = \hat{\Gamma}_k(m_1, \underline{z}_k^t) \setminus \Gamma_k(m_2)$ should take place.

Test generation is based on using path activation procedures on AG. Test generation is actually the reverse process to fault cover analysis. To generate a test pattern for a variable z(m) at a node $m \in M_k$, means to activate three nonoverlapping paths:

1) starting path l^* from the node m^* up to the node m to be tested;
2) a 1-path l^1 from the node m^1 up to some node $m^{T,e}$ with $z(m^{T,e}) = e$;
3) a 0-path l^0 from the node m^0 up to some node $m^{T,\bar{e}}$ with $z(m^{T,\bar{e}}) = \bar{e}$.

It has been shown (Ubar, 1980) that in the case of AG the traditional problem of multiple path activation in

D-algorithm can be reduced to the problem of single path activation. This cuts considerably down the amount of work needed in test generation. Using the possibilities of optimizing SAGs as well as making use of the property (1) for SAGs are the ways to raise the productivity in searching path activation solutions.

As an example, see Fig. 3. Test generation on the gate level leads to troublesome procedures of activating multiple paths. In the case of the optimized AG, we need to test simply nodes, which represent either single or multiple paths in the original circuit. The procedure on AG is straightforward with no inconsistencies found. Results are shown in Table 2 in the form of 8 test patterns and the fault-cover tables for both AGs. As to many faults in the circuit and in the corresponding nonoptimized AG (25 %) remain because of the redundancy untested, we avoid when working with optimized AG a lot of unsuccessful trials (all trials on the optimized AG will be successful).

Table 2

Fault-Cover Table

T	z_1	z_2	z_3	z_4	abcdefg	1	2	3	4	5	6	7	8	9_1	0_1	1_1	2
1	1	1	1	1	0000	0		0			0				0		
2	0	0	0	0	1 000		0	0		0	0		0	0		0	0
3	0	1	1	1	1				1								
4	1	1	1	0	1						1						
5	1	1	0	1	1										1		
6	1	0	0	0	1	1	1										
7	0	0	1	0	1	No faults detected											
8	0	0	0	1	1												1

An example of activated paths for building a test for the node f (the test No 7) is given in Fig. 3. Note that this test cannot be found on the original AG as in the case of optimized AG.

Representing Digital Objects by AG at Different Levels

The most important advantage of AG is the possibility of representing different description levels for digital objects by the same apparatus and using uniform fault models and test design algorithms.

Consider a digital object in general case as a dynamic system $S=(Z,F)$, where Z is a set of variables with values belonging to finite sets $V(z_i)$, $z_i \in Z$, and F is a set of general transfer functions $z_k=f_k(\underline{z}_k)$, $Z_k \subseteq Z$. Compared to the logical level, we introduce the following generalizations into the AG:
1) internal nodes are marked by variables $z_t \in Z$, where $|V(z_i)| \geq 2$ (and therefore also $|\Gamma_k(m)| \geq 2$); 2) terminal nodes can be marked by arbitrary functions from F. Methods of generating such graphs are given by Ubar (1988).

Different physical failures in digital objects traditionally modelled at different description levels are represented in AG by faults at nodes (Ubar, 1988). The fault model used in AG can be regarded as a generalization of the classical stuck-at model for the gate-level (the former is defined for nodes of AG, the latter for Boolean variables). The new fault model makes it possible to handle multiple faults on different object description levels for complex digital objects thanks to the uniform fault representation (Ubar, 1988).

Boolean Level Description

Let us consider the modular approach, where a digital device is represented by a set of modules which can be more complex than logical gates but are described still by Boolean expressions.

Consider an operational unit in Fig. 5 with function $Y=F(I_1,I_2,X,S)$ where I_1,I_2 represent instruction fields, X is a set of flags and S is a data word. Using Boolean variables we have to introduce new sets of variables $I_1 \rightarrow \{I_{11}...I_{1m}\}$ and $I_2 \rightarrow \{I_{21}...I_{2n}\}$ instead of only I_1

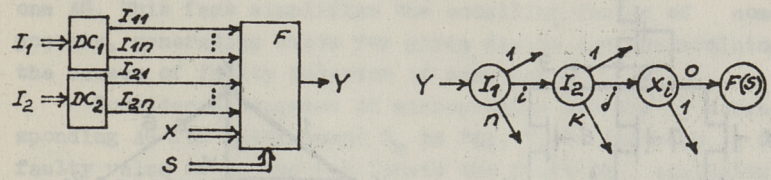


Fig. 5. AG for Boolean level descriptions

and I_2. As a result, the rising complexity of the model is accompanied by inconveniency of describing faults (especially multiple faults) since we should have to deal with multiple stuck-at faults for different subsets of variables.

In the case of AG we represent decoders DC_1 and DC_2 by only two nodes marked by I_1 and I_2 and it is not necessary to introduce Boolean variables instead of I_1 and I_2.

AG for Topological Level Descriptions

In testing digital circuits, the stuck-at fault model is usually used. But this model does not appear to be appropriate for MOS networks. Chiang (1982) proposed modified connection graphs as a network model for the topological level description of complex MOS networks. Opens and shorts on branches in these graphs are used to represent faults in the network.

Let us consider a transistor network of a MOS complex gate (Fig. 6) and the corresponding connection graph (CG). The CG can be easily replaced by an equivalent AG, as it is shown in Fig. 6.

The branches of CG are represented by nodes in AG. For adequate modelling of topological faults like opens and shorts, the nodes of CG are also represented by nodes in AG, where output branches are constantly activated. The open faults are modelled by cancelling the activation on branches from these nodes, while the shorts are modelled by introducing additional branches.

Thus, by adding specific constraints (constantly activated branches) into AG and modifying slightly the test

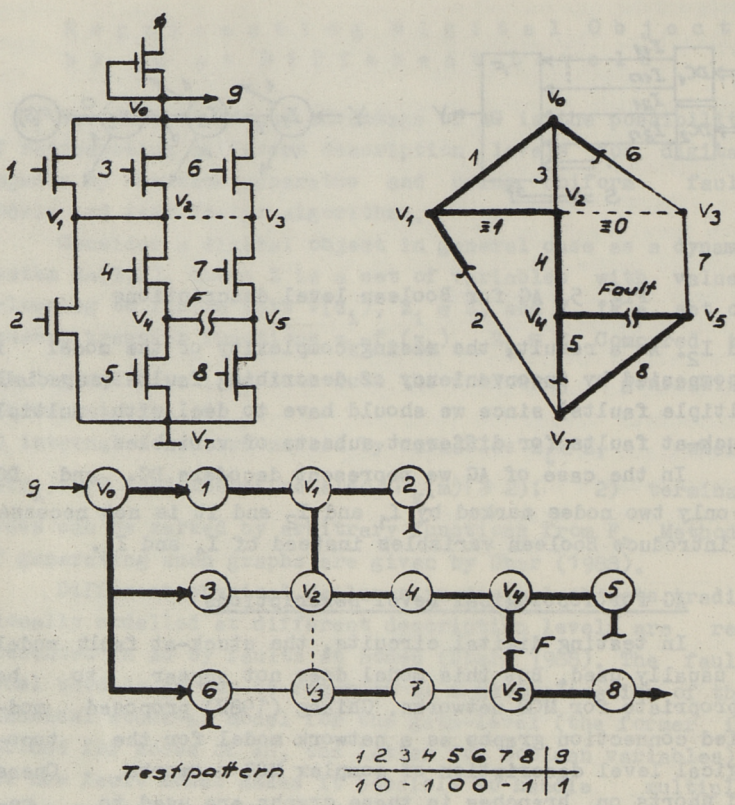


Fig. 6. AG for topological level

generation algorithms, it will be possible on the same methodological basis to solve the test generation tasks for gate level models as well as for transistor level models. For example, the activated paths on AG, needed for testing the open fault on the branch F, are shown by bold lines in Fig. 6. This activation represents the pattern T = = 10x100x1 (1,2,3,4,5,6,7,8) with the output value 1.

Microprogram Level Description

Microprograms (or procedural descriptions) are a good means to model the functionality of devices, but they are not suitable for test purposes. In microprograms the functions of components are distributed all over the microprogram while, in the case of AG,

the functions of each component are collected together in one AG. This fact simplifies the modelling faults of components, generating tests for given faults and determining the causes of faulty behavior of components.

Consider a fragment of microprogram and the corresponding AG for a component R_k in Fig. 7. In the case of faulty value of R_k we can locate the fault by analysing the corresponding activated path on AG.

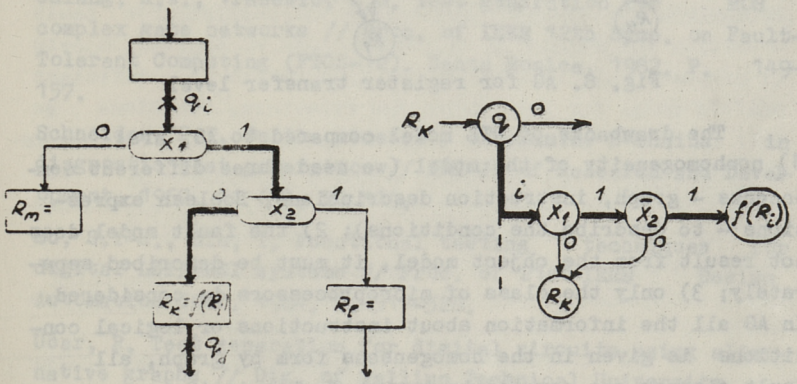


Fig. 7. AG for microprogram level

It is possible to represent the structure of the device on AG. In Fig. 7 the node q represents the address counter of microprogram ROM with its decoder, the nodes $x1, \bar{x}2$ describe some paths on the combinatorial part of the device and the node $F(R_i)$ represents the operational part. In the case of the faulty value of R_k, we can locate its cause by analysing the activated path on AG. In the given case (bold lines are depicting the activated path) the causes for the faulty R_k can be the faulty word q, faulty i-th output of the address decoder, faults on paths $x1$ or $\bar{x}2$ in the combinatorial circuits or faults in the function $f(R_i)$.

Register Transfer Level Descriptions

RTL descriptions are widely used for test generation for digital systems like microprocessors. An example of a RTL-model and the corresponding AG is shown in Fig. 8.

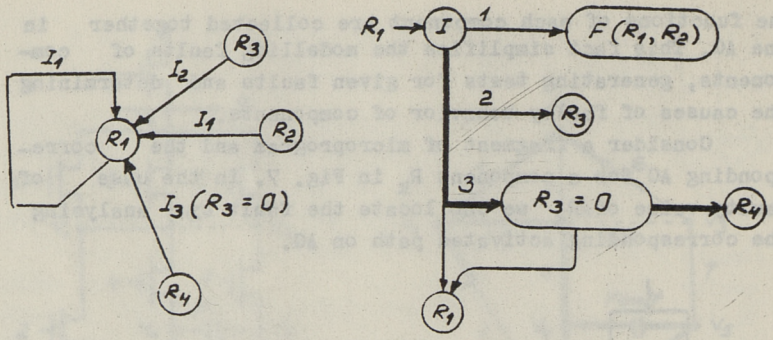


Fig. 8. AG for register transfer level

The drawbacks of RTL model compared to AG, are: 1) nonhomogeneity of the model (we need three different components - graph, instruction descriptions, Boolean expressions - to describe the conditions); 2) the fault model does not result from the object model, it must be described separately; 3) only the class of microprocessors is considered. In AG all the information about instructions or logical conditions is given in the homogeneous form by graph, all fault classes are covered by branch faults in AG.

Conclusion

New test generation and analysis methods were developed on the basis of AG. AG permit concise description of digital circuits, they differ from analogical BDD in ability to design tests for structural faults.

Standard procedures for path activating and traversing on AG were developed to support a wide class of test design tasks. As a result, only one universal library of models for solving all these tasks can be used. Traditional CAD systems for test are based on using a lot of different libraries. Another advantage of the proposed AG-model is the possibility of representing different description levels of digital objects by the same apparatus, using uniform fault models and test design algorithms.

References

Abadir, M.S., Reghbati, H.K. Functional test generation for digital circuits// IEEE Trans. on Comp. 1986. 35. P. 375-379.

Akers, S.B. Binary decision diagrams. IEEE Trans. on Computers, 1978. 27. P. 509-516.

Chiang, K.W., Vranesic, Z.G. Test generation for MOS complex gate networks // Proc. of IEEE 12th Symp. on Fault-Tolerant Computing (FTCS-12). Santa Monica, 1982. P. 149-157.

Schneider, P.R. On the necessity to examine D-chains in diagnostic test generation // IBM J. of Research and Development. 1967. 11 (1). P. 114.

Su, S.Y-H., Lin, T. Functional testing techniques for digital LSI/VLSI systems // Proc. of 21st IEEE Design Automation Conf. 1984. P. 517-528.

Ubar, R. Test generation for digital circuits using alternative graphs // Dig. of Tallinn Technical University. 1976. 409. P. 75-81 (in Russian).

Ubar, R. Beschreibung digitaler Einrichtungen mit AG für die Fehlerdiagnose // Nachrichtentechnik/Elektronic. 1980. 30. H.3. S. 96-102.

Ubar, R. Alternative graphs and technical diagnosis of digital objects // Electronic technic. 1988. Vol. 8. No 5 (132). P. 33-57 (in Russian).

Voolaine. A., Pall, M., Ubar, R. A general approach to multi-valued simulation of digital circuits on the AG model // Dig. of Tallinn Technical University. 1982. 530. P. 23-38.

V. Alango, T. Kont, R. Ubar

New Test Design Techniques for Fault Detection in Digital Objects

Abstract

A new generalized approach on the basis of alternative graphs to the test generation and analysis for fault detection and diagnosis in a wide class of digital objects is proposed. Alternative graphs (AG) permit an efficient uniform model for describing the structure, functions as well as faults in digital objects at different representation levels like transistor, logical, procedural or instruction set levels. Universal test generation algorithms for these levels are proposed. The same uniform model is used to support a wide class of test design tasks like fault activation and propagation, two- or multi-valued simulation, fault cover analysis, etc. Using AG allows to replace a set of traditional model libraries needed for all these tasks by one universal AG-library.

V. Alango, T. Kont, R. Ubar

Uued meetodid testide genereerimiseks diskreetsetele objektidele

Kokkuvõte

Esitatakse uus üldistatud meetod testide koostamiseks ja analüüsimiseks laiale objektide klassile. Meetod põhineb alternatiivsete graafide teoorial. Alternatiivsed graafid tagavad efektiivse ja universaalse mudeli diskreetsete objektide struktuuri, funktsioonide ja ka rikete esitamiseks erinevatel tasemetel (transistor-, loogika-, funktsionaalsel ja protseduursel tasemel). Kirjeldatakse universaalseid testide genereerimise meetodeid nendel tasemetel. Näidatakse mudeli ühtsust erinevate tehnilise diagnostika ülesannete lahendamise suhtes, mis võimaldab asendada suurt hulka erinevaid mudelite teeke erinevate ülesannete jaoks üheainsa universaalse teegiga.

Nr. 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

UDC 681.32

R. Ubar, T. Lohuaru, M. Männisalu,
P. Pukk, E. Vanamölder

TEST SYSTEM FOR FAULT DETECTION AND DIAGNOSIS IN MICROPROCESSOR CONTROL DEVICES

Keywords. Automatic testing; failure detection; digital circuits and systems; microprocessors; alternative graphs.

Introduction

Automatic test systems can be regarded as an important means to support the improving quality of digital devices. The development and application of these systems is expanding as testing requirements become more complex and the need for accurate and reliable test data grows.

A large number of test pattern generation methods for digital devices and systems like microprocessors, microcomputers, microcontrollers are known nowadays (Muehldorf, 1981; Su, 1984). Traditionally these objects are described using different functional models on the register-transfer-level, state-transformation-graph-level, microprogram level or behavioural level. There are also a lot of different functional fault models or fault classes given with respect to these specifications. Each of such fault classes needs a different dedicated method or algorithm for test generation. For example, Thatte (1978) introduced 15 different fault classes for functional fault level testing of microprocessors, El-Lithy (1980) used 17 fault classes for the same purpose. The necessity of using dif-

ferent test generation methods for different fault classes makes the process of the test generation quite troublesome, especially bearing the on-line algorithmic test generation in mind. Also, this approach used traditionally to generate tests for microprocessors cannot be used for digital systems in general.

From the situation described the motivation results to develop a new model for representing a wide class of digital devices and a new fault model which must be simple to support simple test generating algorithms to be implemented in hardware. In this paper a model of digital device on the basis of alternative graphs (Ubar, 1983) is considered and a new test program generation method using these graphs is proposed. The process of test generation is divided into two parts: 1) test program and test data generation (off-line process) and 2) test pattern generation (on-line process). The new test generation method has the following advantages compared to the known-methods: 1) it enables automatical test generation for the devices comprising interior clock generators, 2) tests are directed to multiple fault detection, which raises the reliability of testing and 3) tests are represented in a compact way to reduce the memory space and the time for loading test data needed.

A new architecture for test systems is proposed. The originality of the architecture lies in its capability for on-line testpattern generation in the algorithmic way with the help of hardware. Traditionally, algorithmic test generation strategies are used only for the case of random testing or for the case of simple and regular objects like memory devices, timers, counters, shifters with very simple deterministic test generation procedures.

A l t e r n a t i v e G r a p h M o d e l

Let us consider a digital device as a dynamic system $S = (Z,F)$ where Z is a set of variables with values belonging to finite sets $V(z_i)$, $z_i \in Z$ and F is a set of transfer functions $z_k = f_k(\underline{z}_k)$, $f_k \in F$, given in analytic, graphic, tabular or some other form on the set of variables Z. Here $\underline{z}_k$ is a vector with components $z \in Z_k \subseteq Z$. Such a digital

object can be represented in the form of alternative graph (AG) model $S_{AG} = (Z,G)$ where G is a set of alternative graphs and there is a one-to-one correspondence between G and F.

We represent AG-model for a component (module or part) of the digital device given by a function $z_k = f_k(\underline{z}_k)$ as an oriented graph $G_k = (M_k, \Gamma_k, Z_k)$ where M_k is a set of nodes, Γ_k is a relation where $\Gamma_k(m) \subset M_k$ denotes the set of successors of the node m and $Z_k \subseteq Z$ is a subset of variables (arguments of the function f_k). Each node $m \in M_k$ in a general case is labeled by a weight $e(m) = f_{k,m}(\underline{z}_{k,m})$ where $Z_{k,m} \subseteq Z_k$. For the nonterminal nodes, where $\Gamma_k(m) \neq \emptyset$, e(m) represents a simple variable from Z.

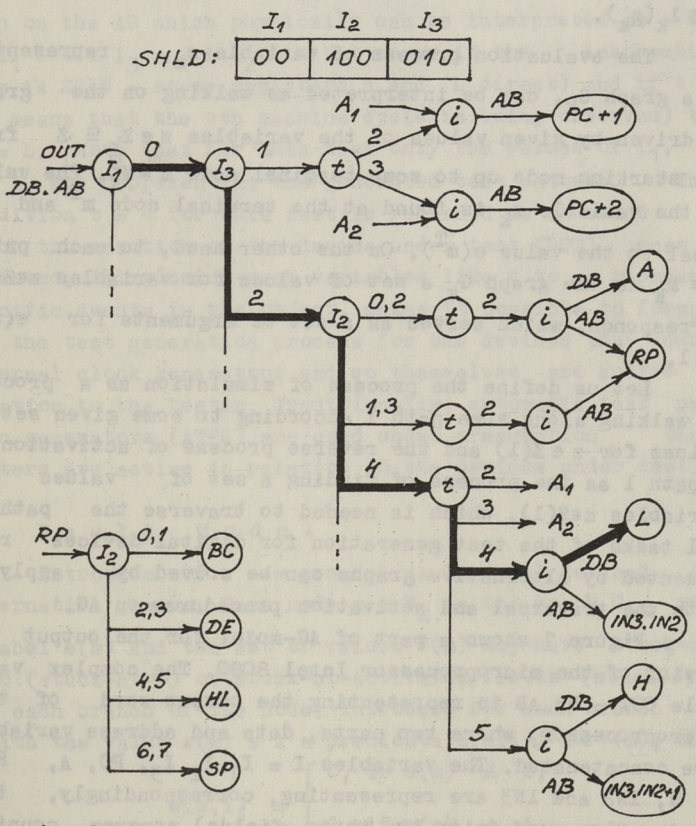


Fig. 1. Alternative graphs representing the output behavior of a typical microprocessor

Let us define the parametric mapping $\Gamma_k(m,e(m))$, so that each value of $e(m) \in V(m)$ determines exactly one and **only** one successor $m_i = \Gamma_k(m,e(m)) \in \Gamma_k(m)$ for the node m. If $\Gamma_k(m) = \emptyset$, then $\Gamma_k(m,e(m)) = \emptyset$ also. Each pattern z_k determines a transitive closure of the mapping $\Gamma_k(m,e(m))$ as a path $l_k(\underline{z}_k) = \hat{\Gamma}_k(m_k^*,\underline{z}_k)$ from the starting node m_k^* up to some terminal node $m_k^T \in l_k(\underline{z}_k)$ where $\Gamma_k(m_k^T) = \emptyset$.

Let us say that G_k represents a function $z_k = f_k(\underline{z}_k)$ if the equation $f_k(\underline{z}_k) = e(m_k^T)$ takes place for each pattern of $\underline{z}_k$ and the corresponding path $l_k(\underline{z}_k) = \hat{\Gamma}_k(m_k^*,\underline{z}_k)$ with $m_k^T \in l_k(\underline{z}_k)$.

The evaluation process of variable z_k, represented by a graph G_k, can be interpreted as walking on the graph G_k driven by given values of the variables $z \in Z_k \subseteq Z$ from the starting node up to some terminal node $m^T \in M_K^T$. The value of the variable z_k is found at the terminal node m^T and is equal to the value $e(m^T)$. On the other hand, to each path $l \subset M_k$ in the graph G_k a set of values for variables $z \in Z(l)$ corresponds, which serves as a set of arguments for $e(m)$, $m \in l$.

Let us define the process of simulation as a process of walking along some path l according to some given set of values for $z \in Z(l)$ and the reverse process of activation of a path l as the process of finding a set of values for variables $z \in Z(l)$, which is needed to traverse the path l. All tasks of the test generation for digital devices represented by alternative graphs can be solved by applying both the traversal and activation procedures on AG.

Figure 1 shows a part of AG-model for the output behavior of the microprocessor Intel 8080. The complex variable OUT = DB.AB is representing the output word of the microprocessor, where two parts, data and address variables are concatenated. The variables $I = I_1.I_2.I_3$, PC, A, RP, L, H, IN2 and IN3 are representing, correspondingly, the instruction word (with its three fields) program counter, accumulator, register pair, registers L and H and the input

data words loaded, correspondingly, during the second and third machine cycles. The variable RP is decoded by its own AG, where the BC, DE, HL represent register pairs and SP denotes the stack pointer. The variable t is modeling the time, its values correspond to the different machine cycles of the microprocessor. Physically, the condition t = = j means the time moment j when the corresponding activated path in AG actually will be carried into effect (the variable OUT will be equal to the value of the function in the terminal node of the activated path). The space variable i is introduced with the aim of compact describing functions for complex words (without using this variable we should have to describe DB and AB separately).

Bold lines in Fig. 1 are denoting an activated path on the AG which physically can be interpreted as a test: if $I_1 = 0$ & $I_3 = 2$ & $I_2 = 4$ (it means if the instruction code is SHLD - store registers H and L direct) and if $t = 4$ (it means that the 4th machine cycle is being observed) then DB = L. Note that for this test only the values of I_1, I_2 and I_3 are representing the generated test pattern. The condition t = 4 for this test is regarded as an event at which the reaction of the device under test (DUT) must be observed. Introducing such variables like t to represent specific events in the object, makes it possible to formalize the test generation process for the devices that contain internal clock generators and so themselves are active in relation to the tester. Traditionally, automatic test pattern generators (ATPG) are used under presumption that testers are active in relation to the devices under test.

F a u l t M o d e l

Introduce now a new general fault model defined on alternative graphs. Each node $m \in M_k$ in the graph G_k with a label e(m) and the set of values V(m) may have stuck-at-open (stuck-at-0) or stuck-at-activated faults (stuck-at-1) for each branch of the node. Introduce for each branch of m with the value e(m) = i a predicate e(m,i), $i \in V(m)$, where

$$e(m,i) = \begin{cases} 0, & \text{if } e(m) \neq i, \\ 1, & \text{if } e(m) = i. \end{cases}$$

The fault model for digital devices, represented by AG, consists of the following faults

1) stuck-at-predicates $e(m,i) \equiv D$, $i \in V(m)$, $D \in \{0,1\}$,

2) conditional stuck-at-activated

$$e(m, j \to V') = 1, \quad j \in V(m), \; V' \subseteq V(m),$$

where

$$e(m, j \to V') = \bigwedge_{i \in V'} e(m,i) \;\Big|\; e(m,j) = 1$$

and $e(m,j)$ is the condition needed for the appearing of the stuck-at-activated branches.

Depending on the accuracy of representing the structure of the device, the fault model described here can represent a wide class of structural and functional failures. For example, it is easy to show that in case of microprocessors the model described covers the broadly used fault model proposed by Thatte (1978).

Test Generation

The process of test generation for digital devices can be carried out on the AG-model by successively activating all the paths in the model so that all the nodes would be tested. The node will be tested if all its output edges are tested for stuck-at faults defined by the fault model in the previous section. In general, the test generation procedure consists of the following three parts:

1) activation of the path $l(m_k^*, m)$ from the starting node m_k^* up to the node m under test;

2) activation of nonoverlapping paths $l(m^j, m^{T,j})$ for each node $m^j \in \Gamma_k(m)$ up to some terminal node $m^{T,j}$;

3) solving the system of inequalities $\forall i,j \in V(m)$, $j \neq i$:

$$e(m^{T,j}) \neq e(m^{T,i}). \qquad (1)$$

Interpretation of these three steps for different classes of digital devices can be different. For the case of microprocessors, on the first and second steps an instruction sequence (test program) will be generated to set up the needed state of the device under test for executing the instruction to be tested and to observe (or transport the

result for observing) the result of the instruction onto the output; on the third step the generation of test data (a set of operands) is taking place - these operands are needed for activating the faults under test.

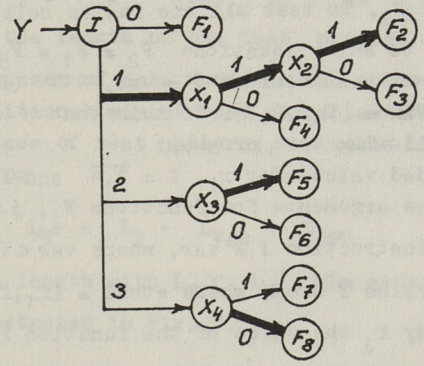


Fig. 2. Alternative graph representing the behaviour of a functional block

An example of carrying out the procedure described is shown in Fig. 2. The AG represented in Fig. 2 is equivalent to the following function:

$$Y = \begin{cases} F_1, \text{ if } I = 0, \\ F_2, \text{ if } I = 1 \text{ and } x_1x_2 = 2, \\ F_3, \text{ if } I = 1 \text{ and } x_1 = 1 \text{ and } x_2 = 0, \\ F_4, \text{ if } I = 1 \text{ and } x_1 = 0, \\ F_5, \text{ if } I = 2 \text{ and } x_3 = 1. \\ F_6, \text{ if } I = 2 \text{ and } x_3 = 0, \\ F_7, \text{ if } I = 3 \text{ and } x_4 = 1, \\ F_8, \text{ if } I = 3 \text{ and } x_4 = 0. \end{cases}$$

Here we can have the following interpretation: Y is some word representing a register in a microprocessor, I is the instruction word variable, x_i represents flags and F_j represents elementary functions (or microoperations). The bold lines in the graph are depicting all the activated

paths needed to test the faults for the output of the node I at the value I = 1. So, carrying out the two first steps of the procedure results in a test vector T = 11110 (I, x_1, x_2,x_3,x_4). On the third step we must find the needed data operands for functions F_i by solving the equations $F_2 \neq F_1$, $F_2 \neq F_5$ and $F_2 \neq F_8$. To test all the faults related to the node I, we have to solve equations $F_2 \neq F_1 \neq F_5 \neq F_8$ and carry out a set of test vectors T = var, 1110 ($\overline{I}$,x_1,x_2,x_3, x_4) where var $\in$ VAR = {0,1,2,3}. To this test corresponds actually the following test program:

1) load the needed values for x_i, $i = \overline{1,4}$ and for all registers serving as arguments for functions F_j, $j \in \{1,2,5,8\}$;

2) fulfill the instruction I = var, where var $\in$ VAR;

3) observe the value Y = et, where et$\in$ET = (f_1,f_2,f_5,f_7);

here we denote by f_j the value of the function F_j.

As we see, the program must be carried out cyclically for all values of var$\in$VAR. Here the values "var", "et" can be regarded as symbolic values for the variables, correspondingly, I and Y. Here, by "et" we represent the reference values (expected results of elementary test steps). During the execution of the program, the symbolic values "var" and "et" will be replaced by the corresponding real values from the arrays VAR and ET.

In general case, to solve the equation (1), we may need more than one set of data operands (initial states of the DUT). From this situation, the need for the next level cycle arises. It means, we have to carry out the test program generated (the cyclic one in relation to symbolic values var) again cyclically for all sets of operands generated during solving the equation (1). So, it is reasonable to introduce a new symbolic value "op" for data words (arguments of functions F_j). During the execution of the test program, the values "op" must be replaced by real values of operands from the corresponding array OP.

To sum up, in general case, the test generation procedure above leads to the construction of tests with a specific structure, consisting of the following components:

1) test program P;

2) array VAR of values to modify the P;
3) array OP of operands and
4) array ET of reference values (expected results of the tests).

The length of this structural test can be measured as

$$L_T = L_P + L_{VAR} + L_{OP} + L_{ET},$$

where L_P is the length of the test program and other L_i denote the number of data words in the corresponding arrays. For the traditional way of test organization as a simple linear sequence of test patterns, we would have the following estimation

$$L_{T'} = L_P \cdot L_{VAR} \cdot L_{OP},$$

for the test length with $L_T << L_{T'}$. The general test structure is illustrated in Fig. 3.

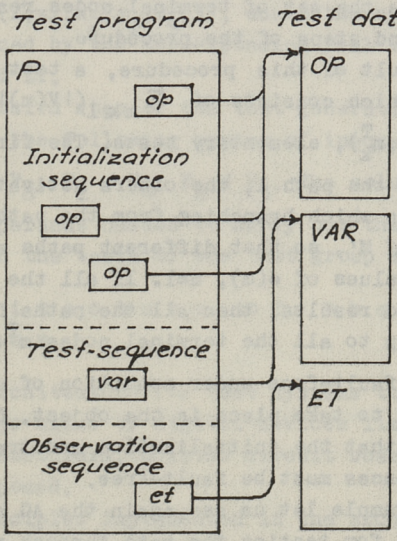


Fig. 3. Test data structure

The simple fault model defined on alternative graphs allows us to develop test generation strategies for the case

where multiple faults are allowed. The algorithm developed by Ubar (1982) for detecting multiple faults at the logical level, can be generalized for the case of the system level using the AG-approach in the following way.

The procedure of test generation consists of three parts:

1) activation of a not yet tested path $l(m_k^*, m_k^T)$ in the graph G_k;

2) activation of noncoincident paths $l(m^j, m^{T,j})$ between the nodes $m^j \in \Gamma_k(m)$ and the corresponding terminal nodes $m^{T,j}$ for all nodes $m \in l(m_k^*, m_k^T)$; the paths $l(m^j, m^{T,j})$ must not be contradictory with each other and with the path $l(m_k^*, m_k^T)$;

3) solving the system of inequalities

$$\forall m^{T,i}, m^{T,j} \in M': e(m^{T,i}) \neq e(m^{T,j}), \qquad (2)$$

where $M' \subseteq M_k^T$ is the set of terminal nodes reached during the first and second steps of the procedure.

As a result of this procedure, a test group will be constructed, which consists of $\sum_{m \in l}(|V(m)| - 1) + 1$, where $l = l(m_k^*, m_k^T)$, elementary tests. The first elementary test activates the path l, the others activate the corresponding other paths which branching from the path l reach terminal nodes $m^T \in M'$, so that different paths will correspond to different values of e(m), $m \in l$. If all the tests from this group have good results, then all the paths from the starting node m_k^* up to all the terminal nodes $m^T \in M'$ will be manifested as fault-free under condition of arbitrary multiple faults allowed to take place in the object. Note that here it is assumed that the initialization and transporting (observing) sequences must be fault-free.

As an example let us see again the AG shown in Fig. 2. The test group for testing the path through nodes I, x_1, x_2 up to the node F_2 under condition of arbitrary possible multiple faults in the graph, is shown in Table 1.

Table 1

Test Group for Detecting Multiple Faults

T	I	x_1	x_2	x_3	x_4	Y
1	0	1	1	1	0	f_1
2	1	1	1	1	0	f_2
3	2	1	1	1	0	f_5
4	3	1	1	1	0	f_8
5	1	0	1	1	0	f_4
6	1	1	0	1	0	f_3

The test group consists of six test patterns with the second one activating the path under test. In the symbolic form the test group can be represented more concisely T = = var,1,0,et $(I,x_1.x_2.x_3,x_4,Y)$, where the symbolic values must be replaced by the corresponding real values from the arrays VAR and ET.

On the third step of the test generation procedure, we need to solve the following system of inequalities: $F_2 \neq F_1$, $F_2 \neq F_5$, $F_2 \neq F_8$, $F_2 \neq F_4$, $F_2 \neq F_3$. The solution will give us the data operands needed to carry out the test program constructed on the basis of the test group in Table 1.

Architecture of the Test System

A new architecture for test systems to be used for testing a large class of digital devices like microprocessors, microcontrollers, printed circuit boards, microcomputers is proposed.

The principles implemented in the architecture, result from the test structure, composed of a test program and of three different data arrays, which was developed in the previous section and is depicted in Fig. 3.

The role of the test system lies in the execution of the second part of the test generation process - e.g. in

the on-line test pattern generation from the test data produced off-line in the way which was described in the previous section. Note that an arbitrary digital device can be represented by alternative graphs. So, the intermediate form for representing test data, proposed here and to be generated off-line on the basis of the AG-model, can be regarded as a general form for a wide class of digital devices. Hence, it can be concluded that the algorithmic test pattern generation method implemented in the test system is also general - it can be used for a wide class of digital devices including the microprocessor-based ones. Traditionally, algorithmic test generation strategies supported by hardware, were used only for the case of random testing or for the case of simple and regular objects like memory devices, timers, counters, shifters with very simple deterministic test algorithms but not for microprocessors.

The algorithmic hardware-supported test generation system proposed here consists of a test program memory block, independent data memory blocks for storing data

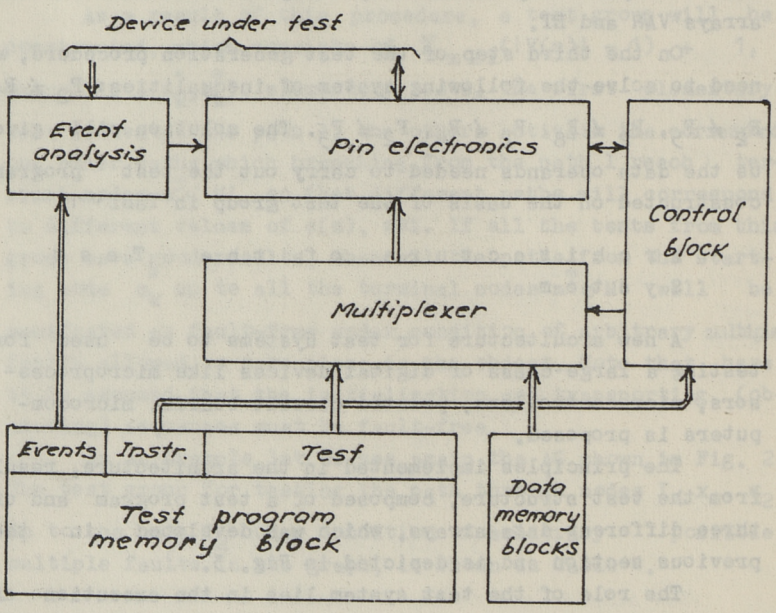


Fig. 4. Simplified block diagram of test system

arrays like VAR, OP and ET, a multiplexer, an event analysis block and a control unit. The simplified block diagram of the test system is represented in Fig. 4.

Test vectors are composed by the multiplexer and the control unit in the on-line mode using the information stored in the program memory and in the data memory. The program word contains two parts: the data (test vector) to be transferred through the multiplexer to the pin electronics and the instruction part. The instruction determines the operation mode of the multiplexer. In the normal mode only data from the program memory will be transferred and data memory will not be used. The other modes allow to mix the information from the program and data memories. In this case, the instruction determines a window in the test vector, where the data normally read from the program memory will be read from the data memory (the exact source - VAR, OP or ET memory block will be also determined by the instruction). The mode of mixing information from different sources corresponds to the case where symbolic values in the test program are to be replaced by the real numerical values from the arrays VAR, OP or ET.

Special features are implemented in the test system to support by the hardware event-driven testing. A part of the test vector is reserved for programming time moments at which the corresponding test vector is to be executed. Programming is actually reduced to only setting the corresponding values for some bits which describe the corresponding changes on the output pins of the DUT, typical of the time moment of interest. Using the possibility of event programming introduced into the test system, different interface protocols between asynchronously working tester and the DUT can be easily programmed. On the other hand, introducing the corresponding information related to the interface protocols (like variables t in Fig. 1) into the AG-description of the device, it will be easy to generate such conditional test programs automatically.

The event-oriented tester architecture makes it possible both to test devices which contain internal clock-generators and are not synchronized by test system and to test dynamically devices that work at higher clock rates than the tester itself.

Conclusion

New test generation and testing methods for microprocessor-based digital devices and digital systems in general were proposed. The class of digital objects for which hardware-supported algorithmic test generation is reasonable was expanded, compared to the present day practice. A reasonable compromise between software and hardware supports in automatic test generation process was found. As a result, a considerable gain in reducing the memory space and the time needed for loading test data was achieved.

To illustrate the results, let us see some characteristics of the test for a microprocessor Intel 8080, which was developed by the method presented in this paper. The test consists of a set of test programs with the total length of 573 test vectors and a set of data arrays with the total set of 3412 one or two byte operands. Due to the compact representation of test data, the gain in memory space compared to the traditional linear art of storing tests was 67.5 times.

References

El-Lithy, M., Husson, R. Bit-sliced microprocessors testing: A case study // Proc. of IEEE 10th Symposium on Fault-Tolerant Computing (FTCS-10). Kyoto (Japan), 1980. P. 126-128.

Muehldorf, E.I., Savkar, A.D. LSI logic testing // IEEE Trans. on Computers. 30. 1980. P. 1-17.

Su, S.Y.-H., Lin, T. Functional testing technique for digital LSI/VLSI Systems // Proc. of IEEE 21st Design Automation Conf. 1984. P. 517-528.

Thatte, S.M., Abraham, J.A. Test generation for microprocessors // IEEE Trans. on Computers. 1980. 29. P. 429-441.

Ubar, R. Test generation for multiple faults in combinational circuits (in Russian) // Dig. of Estonian Acad. of Sciences Phys. & Math. Tallinn, 1982. 31. P. 418-427.

Ubar, R. Test pattern generation for digital systems on the vector alternative graph model // Proc. of IEEE 13th Symposium on Fault-Tolerant Computing (FTCS-13). Milano (Italy) 1983. P. 347-351.

R. Ubar, T. Lohuaru, M. Männisalu,
P. Pukk, E. Vanamölder

Test System for Fault Detection and Diagnosis in Microprocessor Control Devices

Abstract

A new test generation method for microprocessors and digital systems in general on the basis of alternative graphs is developed. Tests are generated using symbolic simulation and organized in a compact way - in the form of symbolic test programs and data arrays. A new architecture for test systems is proposed. Architecture enables on-line generating deterministic test-patterns in the algorithmic way with the help of hardware by simple algorithms. Special features are implemented in the test generation method and in the test system to support event-driven testing, which makes it possible to test dynamically devices that work at higher clock rates than the tester.

R. Ubar, T. Lohuaru, M. Männisalu,
P. Pukk, E. Vanamölder

Mikroprotsessorseadmete kontrollisüsteem

Kokkuvõte

Esitatakse uus meetod testide genereerimiseks mikroprotsessorseadmetele, mis põhineb alternatiivsete graafide kontseptsioonil. Testid genereeritakse sümbolmodelleerimise teel ja organiseeritakse kompaktselt sümbolprogrammi ja andmemassiivide kujul. Esitatakse uus originaalne kontrollisüsteemi arhitektuur, mis on orienteeritud testvektorite aparatuursele generatsioonile eelloetletud andmetest. On loodud võimalused andmete asünkroonseks vahetamiseks testri ja kontrollitava seadme vahel, mis võimaldab testida testrist kiiremaid seadmeid.

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 681.32

В. Заугаров, М. Саарепера,
С. Сторожев

СИСТЕМА СИНТЕЗА ТЕСТОВЫХ ПРОГРАММ ДЛЯ ДИСКРЕТНЫХ ОБЪЕКТОВ ДИАГНОСТИРОВАНИЯ (ОД)

Введение

В данной работе предлагается система автоматизированного проектирования тестовых программ для вычислительных устройств, основанных на микропроцессорных БИС. Предлагаемая система на основе исходных данных, получаемых от пользователя, синтезирует тестовые программы в три этапа:

1) трансляция описания ОД на входном языке в графовую модель ОД;

2) генерирование символьных тест-векторов по графовой модели ОД;

3) формирование тестовых программ, доопределяя символьные тесты-векторы соответствующими исходными данными.

Вся теоретическая часть работы системы основана на теории альтернативных графов (АГ) [1].

1. Класс рассматриваемых ОД

Ограничения на класс рассматриваемых ОД накладывает способ представления ОД моделью АГ. Так как работа системы идет на уровне регистровых передач, то элементарной структурой АГ можно принять единичный граф, изображенный на рис. 1,

где OUT - выходная шина данных ОД;
IN - входная шина данных ОД;
REG - регистр ОД;
I - совокупность управляющих сигналов ОД;

F(REG,IN) - функция ОД;
#CONST - константа.

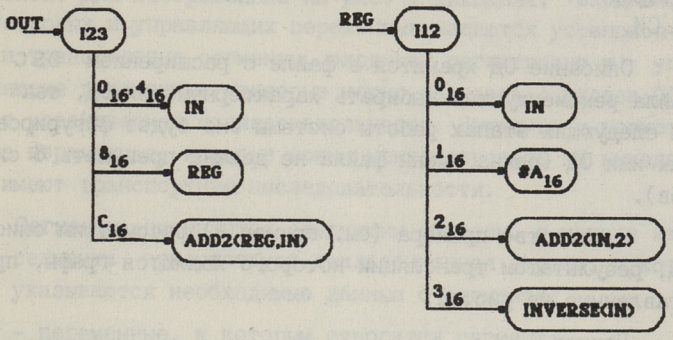


Рис. 1.

Суть предложенной структуры заключается в том, что при подаче управляющего воздействия на ОД будет осуществлена операция передачи данных из одного регистра (входа ОД) в другой регистр (выход ОД). Таким образом, вся модель ОД представляется множеством единичных графов.

2. Исходные данные

Подготовка исходных данных для системы синтеза тестов возложена на пользователя системы. В качестве исходной информации требуются следующие данные:

- входное описание ОД;
- описание установочных и транспортных последовательностей;
- описание функций обработки данных, выполняемых в ОД;
- описание локальных тестов, необходимых для проверки функций обработки данных.

2.I. Входное описание ОД

ОД описывается на входном языке, разработанном на кафедре ЭВМ ТТУ. Конструкции языка описания позволяют строить графы, тип которых представлен на рис. I. Приемник результата (переменная слева) получает значение источника

(переменная справа). Выбор источника осуществляется управляющими сигналами. Управление описывается при помощи операторов IF и SWITCH, подобных функционированию в языке СИ.

Описание ОД хранится в файле с расширением DSC. Имя файла рекомендуется выбирать характерным для ОД, так как на следующих этапах работы системы оно будет фигурировать как имя ОД (длина имени файла не должна превышать 6 символов).

В качестве примера (см. пример I) рассмотрим описание ОД, результатом трансляции которого являются графы, представленные на рис. I.

Пример I.

```
$WORD        8;
$INPUT       I, IN;
$OUTPUT      OUT;
$FOR         I(1)
    $FIELD   I01,I23,I47;
$REG         REG;
$FUNCTION       ADD2,INVERSE;
$FOR      I01:    I1_1=00H, I1_2=01H,
                  I1_3=02H, I1_4=03H;
$FOR      I23:    I2_1=00H,  I2_2=08H,
                  I2_3=00CH, I2_4=004H;
$FOR      I47:    I3_1=0A0H;

$BEGIN
  SWITCH(I01)
       {
       CASE I1_1: REG=IN;
       CASE I1_2: REG=I3_1;
       CASE I1_3: REG=ADD2(IN,2);
       CASE I1_4: REG=INVERSE(IN);
      }

  SWITCH(I23)
    {
       CASE I2_1,I2_4: OUT=IN;
       CASE I2_2:      OUT=REG;
         CASE I2_3:       OUT=ADD2(REG,IN);

    }
$END
```

2.2. Формирование установочных и транспортных последовательностей

В процессе генерирования символьных тест-векторов система работает с единичным графом. Входные данные для еди-

ничного графа доставляют установочные сегменты, а выводом результата тестового воздействия управляют транспортные сегменты. Для изображенных на рис. I выходных, входных, регистровых и управляющих переменных задаются установочные и транспортные сегменты. Входные, регистровые и управляющие переменные, которые можно задавать с входов ОД, имеют установочные последовательности. Выходные и регистровые переменные, которые можно контролировать на выходе ОД, имеют транспортные последовательности.

Сегменты описываются следующим образом. В начале каждого сегмента присутствует декларационная строка, в которой указываются необходимые данные о сегменте:

- переменные, к которым относится сегмент;
- тип сегмента (загрузочный или транспортный).

Каждая декларационная строка должна начинаться с символа '¤', после которого идет перечисление переменных. В конце строки, после символа ':' указывается тип сегмента (L - для загрузки, T - для транспорта).

После декларационной части идет тело сегмента, написанное на языке тестера пользователя. Если сегмент предназначен для:

I) загрузки операндов, то в нем должно присутствовать ключевое слово OPERAND;

2) транспортировки результата, то в нем должно присутствовать ключевое слово ETALON.

В случае, если сегмент служит для формирования кода команды, то применяется специальная форма: в фигурных скобках приводится список управляющих переменных, участвующих в управлении передачей данных.

Описание сегментов (см. пример 2) хранится в файле с расширением PRT .

Пример 2.

```
$ P40 : L
' LOAD reg  KIPa adr=40
BP 2,=(8)000040
BP 3,=(8)OPERAND

$ LMAS,PBA,KOP,AKAP,WR,PK,TO : L
'Load kom obmena ZA={KOP}
   BP 1,=(8){TO,PK,WR,AKAP}
   BP 1,=(8){PBA,LMAS}

$ MO : T
' Choose operand OZY
BP 1,=(8)ETALON
```

2.3. Описание функций ОД

Каждая функция, используемая при описании ОД, структурная организация которой не будет раскрыта, должна быть определена в файле описания функций ОД с расширением FUN (см. пример 3).Предлагаемый язык описания функций - СИ.

Пример 3.

```
int ADD2( a1 ,a2 )
int a1, a2;
{
   return(a1 +a2 );
}


int SUB2( a1, a2 )
int a1, a2;
{
       return( a1 - a2 );
}
```

Пример 4.

```
$ ADD2
   33 aa
   55 77
   33 aa

$ SUB2
   33 33
   55 77
   aa dd
```

2.4. Формирование локальных тестов

Для проверки функциональной части ОД составляются локальные тесты. Каждый локальный тест представляет собой набор операндов для проверки одной или более функций. Оформление локальных тестов осуществляется следующим образом. Перед перечислением операндов должна присутствовать декларационная строка, в которой приводится список функций, для которых эти операнды предназначаются. Каждая деклара-

ционная строка должна начинаться со специального символа'Ω'.

Локальные тесты хранятся в файле описания ОД (см.пример 4) с расширением .LT .

3. Очередность выполнения процедур системой синтеза

Последовательность выполнения процедур иллюстрируется на рис. 2. При этом для каждого отдельного этапа показаны как входные данные, так и результат в виде соответствующих входных и выходных файлов.

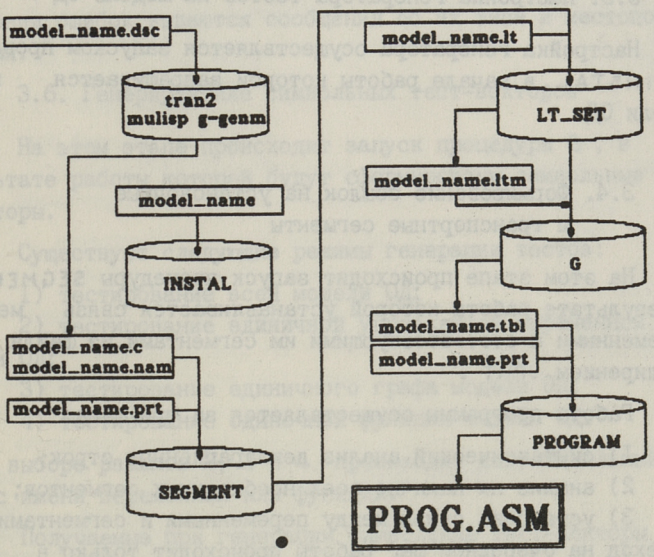


Рис. 2.

3.1. Контроль входного описания ОД

Запуск процедуры -TRAN2 <имя файла>. В результате работы осуществляется синтаксический анализ описания ОД. Каждая синтаксическая ошибка фиксируется выдачей сообщения, в котором указаны код ошибки и номер ошибочной строки. В случае благополучного выполнения этапа будет выдано сообщение YYPARSE=0 .

3.2. Логический анализ описания ОД

На этом этапе осуществляется проверка на соответствие представления моделями АГ описанного ОД.

Запуск процедуры - MULISP G-GENM. В случае благополучного выполнения процедуры будут сформированы два файла:

- один с расширением .C ;
- другой с расширением .NAM .

В противном случае будут выдаваться ошибки, возникающие при составлении модели ОД.

3.3. Настройка генератора тестов на модель ОД

Настройка генератора осуществляется запуском процедуры INSTAL, в начале работы которой запрашивается имя модели ОД.

3.4. Формирование ссылок на установочные и транспортные сегменты

На этом этапе происходит запуск процедуры SEGMENT, в результате работы которой устанавливается связь между переменными и соответствующими им сегментами из файла с расширением .PRT .

Работа программы осуществляется за три шага:

1) синтаксический анализ декларационных строк;
2) анализ на наличие всех необходимых сегментов;
3) установка связи между переменными и сегментами.

Переход на очередной шаг работы происходит только в том случае, если предыдущие прошли успешно. При возникновении ошибок сообщается их тип и местоположение.

3.5. Формирование ссылок на локальные тесты

На этом этапе происходит запуск процедуры LT-SET, в результате работы которой:

1) устанавливается связь между функциями ОД и соответствующими им локальными тестами;

2) формируется файл с локальными тестами (расширение .LTM) в удобной для системы форме.

Работа программы осуществляется за 4 шага:

1) синтаксический анализ декларационных строк;

2) анализ на присутствие всех локальных тестов;

3) контроль на соответствие множества аргументов функций количеству комплектов операндов;

4) установка связи между функциями ОД и локальными тестами.

Переход на очередной шаг работы происходит только в том случае, если предыдущие этапы прошли успешно. При возникновении ошибок выдаются сообщения об их типе и местоположении.

3.6. Генерирование символьных тест-векторов

На этом этапе происходит запуск процедуры G , в результате работы которой будут сформированы символьные тест-векторы.

Существуют следующие режимы генерации тестов:

1) тестирование всей модели ОД;

2) тестирование единичной управляющей переменной модели ОД;

3) тестирование единичного графа модели ОД;

4) тестирование одиночной функции модели ОД.

При выборе режимов 2, 3 и 4 происходит дополнительный запрос имени переменной или функции.

Получаемые при генерации символьные тест-векторы представляют собой следующую совокупность данных:

1) тип и номер вершины АГ-модели ОД;

2) список всех управляющих переменных и их значений.

В случае, если происходило тестирование управляющей переменной, то будет указано ее имя при помощи записи вида

" имя переменной : *VAR* ".

Значения, которые эта переменная принимает, находятся в разделе VARIATIONS ;

3) ссылки на сегменты загрузки и транспортировки;

где T-OP - содержит ссылку на сегмент загрузки операндов;
T-TR - на сегмент транспортировки результатов;
T-VAR - на сегмент формирования управляющего кода;

4) значения загружаемых операндов (раздел OPERANDS);

5) эталонные значения (раздел ETALONES).

Символьные тесты хранятся в файле (см. пример 5) с расширением .TBL .

Пример 5.

```
TEST FOR NODE I : 144

KOP : *VAR*
PG2 : 1
LMAS : 0
PBA : 0
P2 : 0
P1 : e0
PA : 4000
AKAP : 180
WR : 0
PK : 2000
TO : c000

T_OP : 15
T_VAR : 331
T_TR : 226
OPERANDS : 34
VARIABLES : 3d00
ETALONES : 34
```

Пример 6.

```
' LOAD reg  KIPa adr=40
BP 2,=(8)00040
BP 3,=(8)34

'Load kom obmena ZA=3d00
   BP 1,=(8)e440
   BP 1,=(8)fff

' Choose operand OZY
BP 1,=(8)34
```

3.7. Формирование тестовой программы

На заключительном этапе происходит запуск процедуры PROGRAM, в результате работы которой происходит непосредственное формирование тестовой программы. Исходной информацией для этого этапа являются файл символьных тест-векторов и файл с загрузочными и транспортными сегментами.

При формировании конечной программы существует возможность вставлять дополнительные программные фрагменты между:

1) тест-векторами;

2) загрузочным сегментом и сегментом с выполняемой командой;

3) сегментом с выполняемой командой и транспортным сегментом.

Описание дополнительных программных фрагментов осуществляется в файле описания загрузочных и транспортных сегментов. Описание дополнительных сегментов начинается с декларационной строки, первым символом которой должен быть знак ¤ . Затем, в зависимости от предполагаемого места расположения сегмента, могут присутствовать следующие ключевые слова:

- INSERT-BEG;
- INSERT-FIRST;
- INSERT-SECOND;
- INSERT-END.

Конечная тестовая программа помещается в файл PROG.ASM (см. пример 6).

4. Заключение

Особенностью предлагаемой системы автоматизированного проектирования тестов является:

- иерархичность при описании ОД;
- иерархичность при составлении тестов.

Описание ОД производится на двух уровнях: на системном и на операционном. На системном уровне операции обработки данных (функции ОД) задаются абстрактно через имена этих операций. На операционном уровне дается более подробное описание операций обработки данных. При этом последнее может быть задано как на функциональной (при помощи алгоритмов, микропрограмм или функциональных выражений), так и на структурной основе (при помощи вентильной реализации).

Составление тестов производится также на двух уровнях: на системном и операционном. На системном уровне генерирование тестов ведется на основе системного описания ОД, базируясь на локальных тестах, которые составлены для функций объекта, описанных на операционном уровне. Такие локальные тесты могут быть построены на основе как функционального, так и вентильного описания. В данной системе рассматривается еще один альтернативный вариант - ручное построение и ввод таких локальных тестов.

Еще одной особенностью системы является возможность эффективного распределения труда при проектировании тестов между ЭВМ и разработчиком тестов. Последнему резервирована возможность выбора разумных и эффективных процедур обмена информацией между тестером и ОД и связывания их с необходимыми установочными и транспортными (необходимыми для наблюдения реакции ОД) последовательностями.

Машинный поиск таких последовательностей может привести к длительному перебору случайных вариантов и необязательно к нахождению эффективных и корректных решений.

Литература

I. Убар Р.Р. Альтернативные графы и техническое диагностирование дискретных объектов // Электронная техника. Сер. 8. Вып. 5 (132). 1988. С. 33-57.

V. Zaugarov, M. Saarepera, S. Storožev

Testprogrammide sünteesi süsteem diskreetsete diagnostikaobjektide jaoks

Kokkuvõte

Artiklis kirjeldatakse testprogrammide automatiseeritud projekteerimissüsteemi mikroprotsessorlülituste ja seadmete jaoks. Süsteemi teoreetiliseks aluseks on alternatiivsete graafide teooria. Esitatakse lähteandmete klassifikatsioon antud süsteemi jaoks ja kirjeldatakse nende arvutisse sisestamise viise. Tutvustatakse süsteemi peamisi tööetappe.

V. Zaugarov, M. Saarepera, S. Storozhev

Test Program Generation System for Digital Objects

Abstract

Test program generation system for testing microprocessor LSIs and digital devices on their basis is proposed. The theory of alternative graphs serves as a theoretical basis for this system. The input data for the system are described and instructions for entering the data into the computer are given. The basic operation stages of the system are described.

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 681.32

С. Сторожев

АВТОМАТИЗИРОВАННОЕ ПОСТРОЕНИЕ ПРОГРАММ САМОДИАГНОСТИКИ МИКРОПРОЦЕССОРОВ

Введение

В данной статье рассматривается метод автоматизированного построения программ самодиагностики микропроцессоров (МП) на основе абстрактных тестов, получаемых с помощью системы автоматизированного генерирования тестов [1]. Предполагается, что получаемая программа является программой на языке Ассемблера тестируемого микропроцессора.

1. Организация процесса самотестирования

Для определения структуры программы самотестирования (ПС), ее функций и требований к ней необходимо определить механизм самого процесса самотестирования. Предполагается, что ПС находится на диске микро-ЭВМ, МП которой подлежит тестированию. Перед выполнением программы происходит ее загрузка в ОП, после чего осуществляется ее запуск. По мере работы ПС формируется массив результатов, который подлежит дальнейшему анализу.

2. Общий алгоритм получения ПС

Совокупность действий, необходимых для получения результатов самотестирования, разделяется на 6 последовательных этапов.

На первом этапе описывается тестируемый МП на входном языке системы генерации тестов.

На втором этапе формируется модель альтернативных графов [2] (АГ-модель) объекта диагностирования (ОД).

На третьем этапе осуществляется работа генератора тестов, результатом которой являются абстрактные тесты [I].

На четвертом этапе компилятором программ самотестирования (КПС) формируется ПС. Очевидно, что на основе только абстрактных тестов, которые представляют собой последовательность векторов значений определенных сигналов, сгенерировать ПС невозможно. Для этого необходимо получить от пользователя дополнительные сведения о языке Ассемблера тестируемого МП. Эту информацию можно разделить на 2 класса:

I) программные фрагменты для реализации программного тестера;

2) программы загрузки операндов и транспортировки результатов.

На пятом этапе происходит трансляция ПС и ее выполнение, в результате чего формируется массив результатов.

На последнем,шестом, этапе производится анализ полученного массива и делается заключение о техническом состоянии МП.

3. Общие требования к ПС

Основой к созданию КПС является структура ПС, поэтому выбор структуры есть первостепенная задача. Основными требованиями являются:

I) ПС должна удовлетворять требованиям, наложенным на общую организацию процесса самотестирования;

2) объем памяти, занимаемый ПС, должен быть как можно меньше;

3) ПС должна обладать хорошим быстродействием;

4) ПС должна быть простой для освоения;

5) как следствие п. 4, дополнительные данные о языке Ассемблера, вводимые пользователем, должны быть простыми.

4. Структура ПС

Генерируемая ПС состоит из двух частей: область программ и область данных, каждая из которых разбивается на более малые подобласти.

Область программ представляет собой следующую последовательность подобластей:

1) область, занимаемая программным тестером (ПТ), в функцию которого входит непосредственная организация процесса тестирования. Сам ПТ состоит из нескольких однотипных по структуре частей, каждая из которых управляет тестированием одной управляющей вершины АГ [2] или функции;

2) команда останова ПС;

3) область подпрограмм, в которой находится подпрограмма загрузки, подпрограмма транспортировки, связующая подпрограмма и вариационные подпрограммы (см. п. 4.3);

Область данных состоит из 5 подобластей:

1) область памяти, используемая для моделирования всей ОП МП (см. п. 4.2);

2) область вспомогательных ячеек памяти и счетчиков;

3) массив адресов вариационных подпрограмм (см. п. 4.3);

4) массив операндов;

5) массив результатов, в данном случае происходит лишь фиксация места, начиная с которого этот массив будет располагаться.

Работу КПС в общем виде можно разбить на 2 основных шага:

1) контроль поступающей информации как со стороны генератора, так и со стороны пользователя;

2) непосредственное формирование ПС.

4.1. Механизм организации подпрограмм (ПП)

Использование традиционного метода организации ПП в ПС приводит к большим манипуляциям по сохранению и восстановлению указателя стека, что увеличивает объем диагностируемого ядра объекта. Поэтому был предложен другой метод. Его суть заключается в том, что как для вызова ПП, так и для возврата из нее используется команда безусловного перехода (БП) (для МП КР580ИК80А - JMP addr). Тогда, если символический адрес ПП есть X, то ее вызов осуществляется командой JMP X.

Для организации возврата из ПП необходимо определить

адресную часть команды безусловного перехода перед вызовом ПП. Если символический адрес возврата из ПП есть XB, а команда, выполняющая возврат, имеет символический адрес XI, то механизм реализации будет иметь вид, представленный на рис. I.

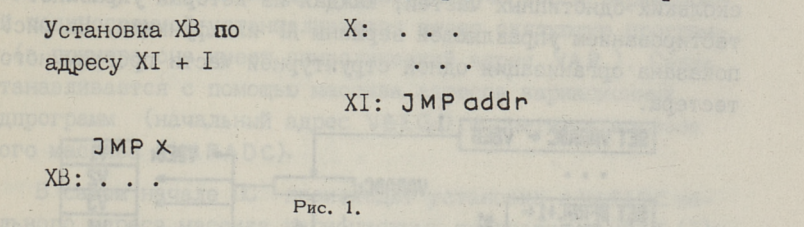


Рис. 1.

4.2. Тестирование команд, работающих с памятью

При организации выполнения команд, работающих с памятью, необходимо определить адреса ячеек памяти, откуда извлекаются операнды или куда заносится результат. При этом для моделирования всей памяти вводятся специальные ячейки, через которые происходит моделирование всех операций, связанных с выборкой памяти.

Сущность предлагаемого метода поясним на примере команды ADD M МП КР580ИК80А, которая использует I байт памяти. Зарезервируем ячейку памяти размером I байт и снабдим ее символическим адресом MX. Ее будем интерпретировать как дополнительный регистр МП, который, как и остальные регистры, необходимо загружать перед выполнением команды значениями из массива операндов. Для того, чтобы команда ADD M могла обратиться к указанной ячейке, необходимо определить регистры H и L. Программу, выполняющую эту функцию, будем называть специальной загрузкой. Для МП КР580ИК80А специальная загрузка состоит из одной строки: LXI H, MX. В действительности в МП КР580ИК80А для выполнения любой команды, работающей с памятью, необходимо 2 байта ОП.

Другими словами, нам удалось организовать моделирование ОП МП с помощью ограниченного числа ячеек.

Что касается вида терминальной вершины АГ для ADD M, то он следующий:

ADD(M(R1),A),

где A - аккумулятор;
M - указывает на необходимость специальной загрузки;
R1 - определяет формат операнда.

4.3. Организация программного тестера

Как отмечалось выше, программный тестер состоит из нескольких однотипных частей, каждая из которых управляет тестированием управляющей вершины АГ или функции. На рис. 2 показана организация одной структурной части программного тестера.

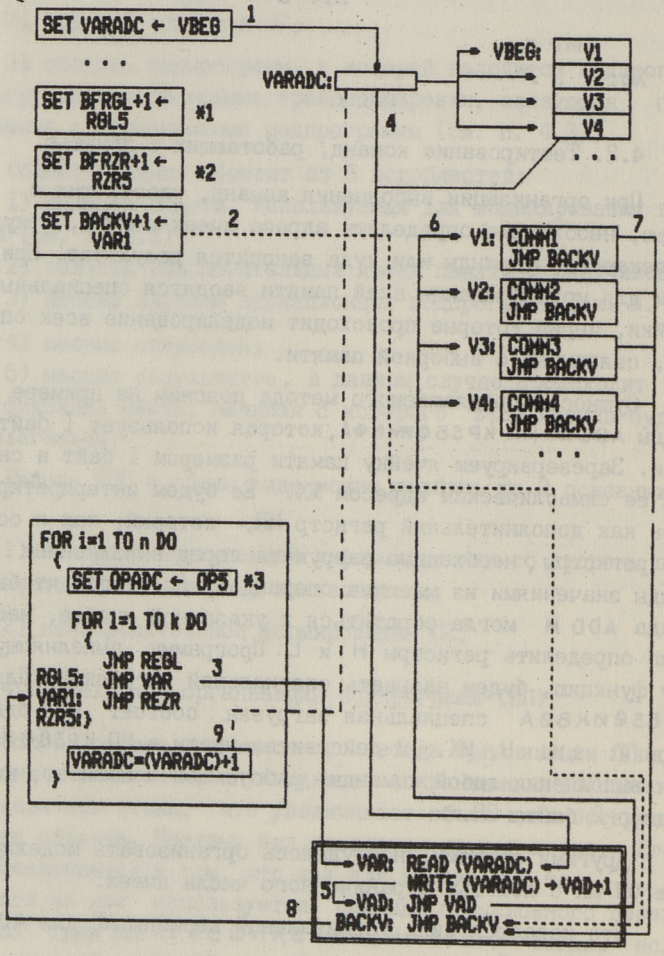


Рис. 2.

Все тестируемые команды оформлены в ПС в виде подпрограмм, которые в дальнейшем мы будем называть вариационны-

ми подпрограммами. Это позволяет гибко и с малыми затратами памяти организовать процесс тестирования (на рис. 2 вариационные подпрограммы отмечены символическими адресами V1, V2,...). Связь между структурной частью и вариационной подпрограммой устанавливается через связующую программу (в примере она имеет символический адрес VAR). Связь устанавливается с помощью массива адресов вариационных подпрограмм (начальный адрес VBEG) и счетчика адресов этого массива (VARADC).

В самом начале ПС происходит установка в VARADC начального адреса массива вариационных подпрограмм VBEG(I): SET VARADC<-VBEG.

Затем устанавливается адрес возврата из подпрограммы загрузки (*1) и подпрограммы транспортировки (*2), где RGL5 - адрес возврата из ПП загрузки, RZR5 - адрес возврата из ПП транспортировки, а BFRGL и BFRZR - адреса команд возврата из соответствующих ПП.

После этого начинается непосредственная работа по тестированию (n - количество вариационных ПП, к - количество комплектов операндов, необходимых для тестирования текущей управляющей вершины). Во время процесса *3 происходит установка начального адреса массива операндов в счетчик OPADC.

После входа во внутренний цикл осуществляется последовательный вызов ПП загрузки, связующей ПП и ПП транспортировки. При входе в связующую ПП выбирается адрес вариационной ПП (4), после чего он устанавливается в адресную часть команды вызова вариационной ПП(5), которая и осуществляет переход (6).

После выполнения вариационной ПП происходит возврат (7) в связующую ПП, а оттуда в главную программу.

После того, как одна команда обработает все комплекты операндов, изменяется содержимое счетчика VARADC (9) так, что при следующем вызове связующей ПП она будет устанавливать связь с другой вариационной ПП.

Литература

I. Коньт Т. Подход к автоматизации построения тестов для микропроцессорных БИС // Тр. Таллиннск. политехн. ин-та. 1988. № 674. С. 55-64.

2. Убар Р. Альтернативные графы и техническое диагностирование дискретных объектов // Электронная техника. Сер. 8. Вып. 5 (132). 1988. С. 33-57.

3. Gartner A. Optimierte Selbsttestprogramme für Mikroprozessoren. TH Aachen, 1984. 138 S.

S. Storožev

Mikroprotsessorite autodiagnostika programmide automaatsüntees

Kokkuvõte

Artiklis tutvustatakse mikroprotsessorite ja mikroarvutite autodiagnostika programmide automaatprojekteerimise meetodit. Meetod põhineb kontrollitava seadme käsusüsteemi kirjeldamisel alternatiivsete graafide abil. Nende alusel sünteesitakse abstraktsed testvektorid, millest hiljem kompileeritakse testprogramm. Esitatakse programse testri loomise metoodika testitava mikroprotsessori käsusüsteemi abil. Programne tester koos genereeritud testprogrammiga moodustavad autodiagnostika programmi.

S. Storozhev

Design of Self-Testing Programs for Microprocessors

Abstract

A new self-testing program design method for microprocessors or micro-computers is proposed. The method is based on using alternative graphs for describing instruction sets. On the basis of alternative graphs, abstract test-vectors are generated of which the final test-program will be compiled. A method for designing a tester as a program in the assembly language of the microprocessor is proposed. Both the tester and the test-program form a self-testing program.

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 681.32

В. Заугаров

ДЕКОМПОЗИЦИОННЫЙ МЕТОД ПРЕДСТАВЛЕНИЯ ОБЪЕКТОВ ДИАГНОСТИРОВАНИЯ МОДЕЛЯМИ ОБОБЩЕННЫХ АЛЬТЕРНАТИВНЫХ ГРАФОВ

Введение

В данной статье предлагается метод описания объектов диагностирования (ОД), который повышает возможность автоматизации построения тестов для широкого класса ОД: от микропроцессорных БИС до структур, содержащих микропроцессорные БИС в качестве компонентов. Метод основывается на представлении ОД совокупностью моделей обобщенных альтернативных графов (ОАГ) [1].

I. Модель ОД

Модель ОАГ была использована при автоматизации генерирования тестов для микропроцессорных БИС [2]. В предложенной автором работе некоторые важные свойства ОАГ не нашли применения, а именно:

1) возможность декомпозиции общего управляющего поля на подполя согласно предлагаемой структуре;

2) возможность взвешивания вершины внутренними переменными без памяти.

Поэтому в описании ОД моделью ОАГ есть существенные ограничения, не позволяющие данной системе стать универсальной. Фактически круг рассматриваемых ОД сводится только к описанию микропроцессорных БИС.

В работе [1] автором были выделены все возможные типы переменных, которыми могут быть взвешены вершины ОАГ. Наряду с входными и регистровыми переменными, применение ко-

торых в модели ОАГ было достаточным для описания моделей микропроцессорных БИС, речь шла также о внутренних переменных без памяти - сигнальных переменных.

В данной статье акцентируется внимание на вопросе включения сигнальной переменной в модель ОАГ, чтобы достичь универсальности в описании моделей ОД. Соответственно и система автоматизированного генерирования тестов на основе такой модели приобретает универсальный характер.

Приведем примеры применения сигнальной переменной в описании ОД и зависящие от этого способы представления ОАГ. На рисунке I приводятся примеры использования сигнальной переменной в функциональных вершинах ОАГ. На рис.2 приводятся примеры использования сигнальной переменной в управляющих вершинах ОАГ. Соответственно по пунктам:

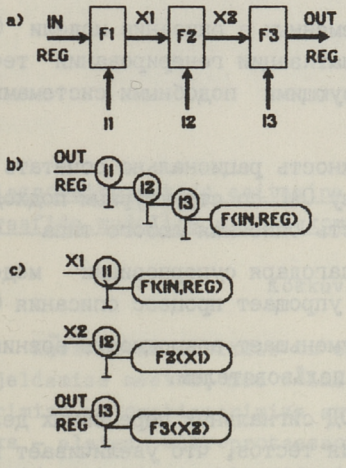


Рис. 1.

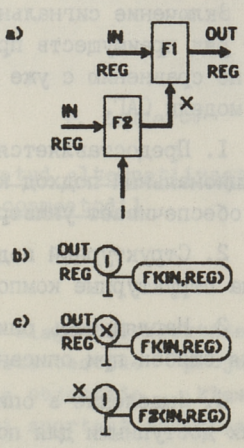


Рис. 2.

а) структура ОД;

б) модель ОД, построенная без применения сигнальных переменных;

в) модель ОД, построенная при помощи сигнальных переменных,

где IN,OUT - входные, выходные переменные;

REG - регистровые переменные;
I - управляющие переменные;
X - сигнальные переменные;
F(IN) - функция ОД.

Дается представление модели ОД без и с применением сигнальных переменных в модели ОАГ.

Так как сигнальные переменные не могут быть зафиксированы, то для них применяется процесс развертки, который заканчивается, когда все вершины, питающие рассматриваемую вершину, взвешены входными и регистровыми переменными, значения которых можно контролировать, используя установочные и транспортные последовательности.

2. Обоснование ввода сигнальной переменной

Включение сигнальных переменных в описание модели ОД дает ряд преимуществ при автоматизации генерирования тестов по сравнению с уже существующими подобными системами на модели ОАГ.

1. Предоставляется возможность рационально сочетать функциональный подход к синтезу ОАГ со структурным подходом, что обеспечивает универсальность описания любого типа ОД.

2. Структурный подход, благодаря суперпозиции модели ОД на структурные компоненты, упрощает процесс описания ОД.

3. Наглядность описания уменьшает возможность возникновения ошибок при описании ОД пользователем.

4. Выделение в описании ОД сигнальных переменных делает их доступными для построения тестов, что увеличивает процент локализации дефектов ОД.

5. Характерно, что ввод сигнальной переменной в модель ОАГ не противоречит основным положениям теории ОАГ, позволяет использовать предложенные в источнике [3] методы автоматизированного генерирования тестов для вершин, взвешенных переменными любого типа, и дает возможность отражать в модели ОАГ переменные из многозначного алфавита.

Литература

I. Убар Р. Универсальный подход к автоматизации проектирования тестов для широкого класса дискретных объектов // Тр. Таллиннск. политехн. ин-та. 1986. № 626. С. 70–92.

2. Коньт Т. Подход к автоматизации построения тестов для микропроцессорных БИС // Тр. Таллиннск. политехн. ин-та. 1988. № 674. С. 55–64.

3. Убар Р. Исследование и разработка методов тестового диагностирования дискретных систем: Дис. на соискание ученой степени доктора технических наук. Таллинн, 1986.

V. Zaugarov

Diagnoosiobjektide esitamine üldistatud alternatiivsete graafide mudelite abil dekompositsioonmeetodil

Kokkuvõte

Käesolevas artiklis on esitatud diagnoosiobjektide kirjeldamise meetod, mis võimaldab tõsta testide projekteerimise automatiseerimise astet laia objektide klassi jaoks – alates mikroprotsessori tüüpi suurtest integraalskeemidest kuni struktuurideni, mille komponentideks on need samad integraalskeemid. Meetod põhineb diagnoosiobjektide esitamisel üldistatud alternatiivsete graafide mudelitena.

V. Zaugarov

Decomposition Method for Representing Digital Devices by Generalized Alternative Graphs

Abstract

A method for representing digital devices, which allows to raise the level of automatization of test design processes for a wide class of devices, is proposed. This class of devices includes microprocessors and microprocessor systems. The method is based on the representation of digital devices by generalized alternative graphs.

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 621.382

С. Орро, Т. Ранг

МОДЕЛЬ МЕЖСОЕДИНЕНИЙ/ЛИНИИ ПЕРЕДАЧИ НА БАЗЕ ВТСП

Разработка межсоединений/линии передачи на базе ВТСП требует создания моделей, описывающих влияние технологических параметров на электрические характеристики межсоединений/линии передачи.

Линии передачи характеризуются следующими величинами:

I) волновое сопротивление линий

$$\rho_0 \cong z_0 \sqrt{\frac{z}{4}}\ ; \tag{1}$$

2) сопротивление Z и проводимость Y линий на единицу длины линий;

3) постоянная распространения γ

$$\gamma = \sqrt{ZY} = z / z_0\ , \tag{2}$$

$$\gamma = \alpha + j\beta\ ,$$

где β - постоянная сдвига фазы;

α - постоянная затухания.

Постоянная распространения записи от частоты

$$\gamma(f) = \alpha(f) + j\beta(f). \tag{3}$$

Можно распределить следующие компоненты постоянного распространения:

$$\alpha(f) = \alpha_{COND}(f) + \alpha_{DE}(f); \tag{4}$$

$$\beta(f) = \beta_{MODAL}(f) + \beta_{COND}(f). \tag{5}$$

Значение этих компонентов рассмотрим ниже.

Сопротивление и проводимость линии с единичной длиной определим по формуле

$$Z = j2\pi\mu_0 g_1 + z_s(f)\, g_2. \tag{6}$$

$$Y = \varepsilon_0 2\pi f (\varepsilon_{eff} j + \varepsilon_r \tan\delta)/g_1 . \qquad (7)$$

где μ, ε_0 – постоянные свободного пространства;

f – частота;

ε_{eff} – эффективное значение диэлектрической проницаемости подложки;

ε_r – диэлектрическая проницаемость подложки;

g_1, g_2 – константы, зависящие от геометрии линий;

$\tan\delta$ – тангенс угла потерь;

$z_s(f)$ – импеданс проводящей поверхности (электрода).

Рассмотрим эти величины для копланарной линии (см. рис.).

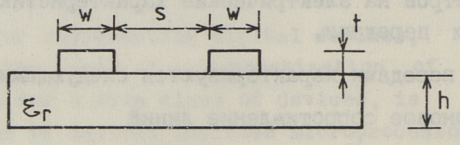


Рис. Структура линии передачи.

Эффективное значение диэлектрической проницаемости учитывает факт, что не все электрическое поле находится в диэлектрике.

$$\varepsilon_{eff} = \frac{\varepsilon_r + 1}{2}\left[\tanh\left\{1{,}785 \log\frac{h}{w} + 1{,}75\right\} + \frac{hw}{4}\{0{,}04 - 0{,}7K + 0{,}01(1 - 0{,}1\varepsilon_r)(0{,}25 + K)\}\right], \qquad (8)$$

где $$K = S/(S + 2w),$$

$$\varepsilon_{eff} = \varepsilon_{eff}(0).$$

Эта формула правильная только в том случае, если длина λ намного больше чем размеры линий. Если это не так, тогда ε_{eff} будет зависеть от частоты:

$$\sqrt{\varepsilon_{eff}(f)} = \sqrt{\varepsilon_{eff}(0)} + \left[\frac{\sqrt{\varepsilon_r} - \sqrt{\varepsilon_{eff}(0)}}{1 + aG^{-b}}\right], \qquad (9)$$

где $G = f/f_{TE}$ – нормированная частота;

$f_{TE} = \dfrac{C}{4h\sqrt{\varepsilon_r - 1}}$ – частота среза;

a, b - константы, учитывающие размеры линий.

Для копланарной линии константы g_1 и g_2 имеют вид:

$$g_1 = \begin{cases} \pi \left\{ \ln \left[2 \dfrac{1+\sqrt{k}}{1-\sqrt{k}} \right] \right\}^{-1} & \text{при } 0 \leq k \leq 0{,}707, \\ \pi^{-1} \ln \left[2 \dfrac{1+\sqrt{k}}{1-\sqrt{k}} \right] & \text{при } 0{,}707 \leq k \leq 1, \end{cases} \qquad (10)$$

$$k = \frac{S}{S+2W},$$

$$g_2 = 17{,}34 \left(\frac{p'}{\pi S} \right) \left(1 + \frac{W}{S} \right) \frac{\dfrac{1{,}25}{\pi} \ln \dfrac{2\pi W}{t} + 1 + \dfrac{1{,}25t}{\pi W}}{\left[1 + \dfrac{2W}{S} + \dfrac{1{,}25}{\pi S} \left(1 + \ln \dfrac{4\pi W}{t} \right) \right]^2}, \qquad (11)$$

где

$$p' = \begin{cases} k \left[\left(1 - \sqrt{1-k^2} \right) (1-k^2)^{\frac{3}{4}} \right]^{-1} g_1^2 & \text{при } 0 \leq k \leq 0{,}707, \\ \left[(1-k) \sqrt{k} \right]^{-1} & \text{при } 0{,}707 \leq k \leq 1. \end{cases}$$

Зависимости постоянного распространения от размеров линий учитывает модальная дисперсия:

$$\beta_{MODAL}(f) = \frac{2\pi f}{C} \sqrt{\varepsilon_{eff}(f)}. \qquad (12)$$

Модальная дисперсия определяет фазовую скорость

$$V_0 = 2\pi f \; / \; \beta_{MODAL}(f).$$

Члены $\alpha(f)$ и $\beta(f)$ учитывают параметры электродов. Эти параметры зависят от импеданса проводящей поверхности:

$$z_s(f) = \sqrt{2\pi j f \mu_0 / \sigma} \; \coth \left(\sqrt{2\pi j f \mu_0 / \sigma} \cdot t \right), \qquad (13)$$

где t - толщина электрода;

σ - проводимость электрода.

$$\alpha_{COND}(f) = Re \left[\frac{z_s}{z_0} \right] g_2 \quad [\text{Дб/един длины}];$$

$$\beta_{COND}(f) = Im \left[\frac{z_s}{z_0} \right] g_2 \quad [\text{Дб/един длины}].$$

Если электроды сверхпроводящие, тогда проводимость комплексная:

$$\sigma = \sigma_1 - j\sigma_2 . \qquad (14)$$

Для расчета величины σ используют теорию Mattis Brudeen. Для низкотемпературной СП теория подтверждена экспериментами, но для ВТСП пока мало известна. По этой теории σ_1 и σ_2 определяются следующим образом:

$$\frac{\sigma_1}{\sigma_2} = \frac{2}{\hbar w}\int_{\Delta}^{\infty} dE\left[f(E) - f(E+\hbar w)\right]\frac{g(E)}{(E^2-\Delta^2)^{1/2}\left[(E+\hbar w)^2-\Delta^2\right]^{1/2}} +$$

$$+ \frac{1}{\hbar w}\int_{\Delta-\hbar w}^{-\Delta} dE\left[1-2f(E+\hbar w)\right]\frac{g(E)}{(E^2-\Delta^2)^{1/2}\left[(E+\hbar w)^2-\Delta^2\right]^{1/2}} . \qquad (15)$$

$$\frac{\sigma_2}{\sigma_1} = \frac{1}{\hbar w}\int_{\Delta-\hbar w;\,-\Delta}^{\Delta} dE\left[1-2f(E+\hbar w)\right]\frac{g(E)}{(\Delta^2-E^2)^{1/2}\left[(E+\hbar w)^2-\Delta^2\right]^{1/2}} , \qquad (16)$$

где $g(E) = E^2 + \Delta^2 + \hbar w E$;

$w = 2\pi f$;

σ_n - проводимость в нормальном состоянии;

$\Delta = \Delta(T)$;

$f(E)$ - функция распределения Ферми-Дирак.

В первом выражении для σ_1 второй интеграл равен нулю при $\hbar w < 2\Delta$.

Во втором выражении для σ_2 интегрируется от Δ, если $\hbar w > 2\Delta$.

Потери в диэлектрике учитывает $\alpha_{DE}(f)$:

$$\alpha_{DE}(f) = 27{,}3\,\frac{\varepsilon_r}{\sqrt{\varepsilon_{eff}(f)}}\cdot\frac{\varepsilon_{eff}(f)-1}{\varepsilon_r-1}\cdot\frac{\tan\delta}{\lambda_0} . \qquad (17)$$

[Дж/единица длины].

λ_0 - длина волны в свободном пространстве.

Для диэлектрика диэлектрическая проницаемость является комплексной функцией частоты:

$$\left.\begin{aligned} \varepsilon_r(f) &= \varepsilon'_r(f) - j\varepsilon''_r(f), \\ \operatorname{tg}\delta &= (2\pi f \varepsilon'' + \sigma_{DE}) / 2\pi f \varepsilon' \end{aligned}\right\} \qquad (18)$$

где σ_{DE} - проводимость подложки.

Для затухания в диэлектрике имеется также формула:

$$\alpha_{DE}(f) = \frac{\pi f}{C} \sqrt{\varepsilon_{eff}(f)}\, \operatorname{tg}\delta . \qquad (19)$$

С помощью известной постоянной распространения $\gamma(f)$ можем найти сигнал в сечении линий z $v(\eta, z)$, если во входе был сигнал $v(\eta, 0)$:

$$v(\eta, z) = F^{-1}\left[F\{v(\eta, 0)\} - \exp\{-\gamma(f) z\}\right], \qquad (20)$$

где F - преобразование Фурье.

Константы a и b в разных работах имеют разные значения:

a = 4, B = I,5 [I]
a = 23,5 в = 0,9 [2]
a = 260 в = 0,9 (зависит от геометрии).

Кроме вышеприведенной модели требуется учитывать самопрогрев линии передачи/межсоединений. Для полного решения вопроса требуется численное решение теплового уравнения энергии, момента и непрерывности для сжимаемого потока: в литературе встречаются первые такие попытки, как например, [3].

По нашему мнению самым перспективным направлением использования ВТСП является его роль межсоединением или линией передачи в интегральных и гибридных схемах.

Литература

1. Hsiang T.Y., Whitaker J.F., Sobolovski R., Martinet S., Goldi L.P. High-frequency characterization of superconducting transmission structures from picosecond transient measurements. ISEC '89. Tokyo. P. 510-515.

2. Whitaker J.F., Sobolovski R., Dykaar D.R., Hsiang T.Y., Kovrov G.A. Propagandation model for ultrafast signals on superconduc-

tive dispersive striplines // IEEE Trans. Microwave Theory and Techn. 1988. Vol. 36. P. 277-285.

3. T i e n C.L., F l i k M.I., P h e l a n P.E. Mechanisms of local thermal stability in high temperature superconductors // Cryogenics. 1989. Vol. 29. P.602-609.

S. Orro, T. Rang

Kõrgtemperatuurilisel ülijuhtivusel baseeruvate ühendusliinide / ülekandeliinide modelleerimine

Kokkuvõte

Artiklis on käsitletud kõrgtemperatuurilisel ülijuhtivusel baseeruvate ühendusliinide/ülekandeliinide mudeleid. Mudel põhineb lainejuhtivuse teoorial, arvutamine kiirel Fourier' teisendusel.

S. Orro, T. Rang

Modelling the Connection Paths / Transmission Lines on the Basis of Superconductivity

Abstract

A superconductive based model for connection paths / transmission lines has been described. The model is based on wave theory and uses rapid Fourier transformation. The model allows us to determine the influence of technological and geometrical parameters on connection path /transmission line characteristics.

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED

ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

УДК 621.382

Т. Ранг, А. Коэл

МОДЕЛЬ КЛЮЧА НА БАЗЕ ВТСП

Развитие исследований ВТСП ключей подтверждает наше мнение, что в настоящее время ни технология изготовления, ни возможности управления не достигли такого уровня, чтобы можно было ждать в ближайшее время кардинальных изменений в этой области, хотя некоторые успехи имеются.

Короткий обзор моделей ключей приведен в источнике [1]. Повторим только некоторые главные моменты. Электрическая эквивалентная модель ключа приведена на рис. 1.

Источник тока I_0 питает ВТСП ключ с током меньше или равным критическому току (I_c). Нагрузочное сопротивление в порядке нескольких омов и подключено к внутренним контактам.

При B=0 данный ключ имеет сверхпроводимые свойства и ток источника протекает через ключ, а ток в нагрузке равняется нулю. Магнитное поле не менее 10 G, перпендикулярное к течению тока ВТСП, превращает ключ в сопротивление, как показано на рис. 1б. Ток, который может быть переключен, определяется сопротивлениями нагрузки, ВТСП и серебряных контактов соответственно.

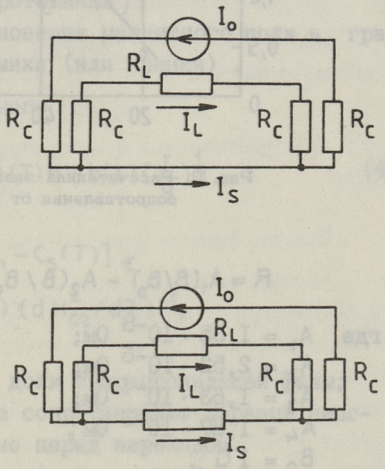


Рис. 1. Электрические включения ВТСП ключа:

а) B = 0T;
б) B ≠ 0T.

Ток источника (не учитывая сопротивление контактов R_c) определяется по закону $I_o = I_s + V_s / R_L$, где R_L - нагрузочное сопротивление.

Интерес представляет определение величины сопротивления ВТСП, т.е. магнитосопротивления. В работе [1] приведенная формула, к сожалению, очень плохо вычисляется из-за нехватки начальных данных. Поэтому на базе данных работы [2] предлагается очень простая формула для определения магнитосопротивления в зависимости от внешней индуктивности (см. рис. 2):

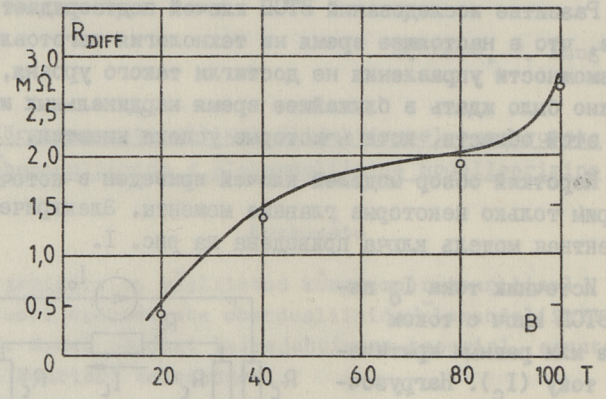


Рис. 2. Рассчитанная зависимость дифференциального сопротивления от магнитного поля.

$$R = A_1(B/B_o)^3 - A_2(B/B_o)^2 + A_3(B/B_o) - A_4, \qquad (I)$$

где $A_1 = 1{,}35 \cdot 10^{-8}$ Ом;
$A_2 = 2{,}52 \cdot 10^{-6}$ Ом;
$A_3 = 1{,}63 \cdot 10^{-4}$ Ом;
$A_4 = 1{,}87 \cdot 10^{-3}$ Ом;
$B_0 = 1$ Гс.

Более подробный подход к объяснению магнитосопротивления следует искать в неоднородности иерархических (пленочных) образцов. Эта неоднородность связана, по-видимому, с вариациями химического состава различных гранул керамики (пленки) (или их внутренних областей) и приводит к неоднородным (по объему керамических/пленочных образцов) распре-

делениям критических температур T_c и полей H_{c2}. В этом случае доля $C_s(H, T)$ сверхпроводящей фазы падает с ростом магнитного поля по закону [3]:

$$C_s(H,T) = C_s(T) - f(T)H/(dH_{c2}/dT), \qquad (2)$$

где $f(T_c)$ – функция распределения гранул по критическим температурам;

dH_{c2}/dT – усредненная производная по угловому распределению гранул;

$H << H_{c2}$ – магнитное поле считается слабым.

Для конечной плотности тока

$$C_s(I,H,T) - C_s(T) = -\frac{1}{2}\left[H + \left(\frac{I}{g}\right)^{\frac{1}{2}}\right] f(T)\,(dH_{c2}^{\perp}/dT), \qquad (3)$$

где $H_{c2}^{\perp}$ – значение верхнего критического поля для ориентации $H_{\perp}$, $g = q\xi^3/\hbar\lambda^2$ (ξ корреляционная длина задачи протекания):

λ – глубина проникновения магнитного поля в гранулы ВТСП керамики (или пленки) .

Исходя из вышеизложенного

$$R(I,H,T) = R(T) + b(T)H + b(T)\left(\frac{I}{g}\right)^{\frac{1}{2}}, \qquad (4)$$

где

$$R(T) = R_n[C_n - C_s(T)],$$

$$b = \left(\frac{1}{2}\right) R_n f(T)\,(dH_{c2}^{\perp}/dT)^{-1},$$

$C_p = 0{,}17$ пороговая доля сверхпроводящей фазы;

R_n – нормальное сопротивление образца непосредственно перед переходом.

По данным работы [4] $dH_{c2}^{\perp}/dT \approx 10^4$ Э/К, $f(T) \sim 1\ К^{-1}$; однако точное выражение неизвестно.

Важным параметром является определение толщины сверхпроводящего слоя в зависимости от магнитного поля. Исходим из уравнения

$$\int_{-d/2}^{d/2}(\Omega_{sH}-\Omega_{nH})dx=\frac{H_{cm}^2}{4\pi}\int_{-d/2}^{d/2}\left[-\psi^2+\frac{\psi^4}{2}+(H-H_0)^2+A\psi^2\right]dx, \qquad (5)$$

где $$H_0^2=\frac{2\psi(1-\psi^2)ch^2(\psi d/2)}{sh(\psi d)/(\psi d)-1}.$$

Для большой толщины $\psi\to 1$ ($\psi=1-\varphi$,

где $\varphi<<1$), получим $$\varphi\approx H_0^2/(2d).$$

Для тонкой пленки

$$H_c=2\sqrt{6}\,H_{cm}\lambda/d, \qquad (6)$$

где $$\lambda=(mc^2/4\pi n_s q^2)^{\frac{1}{2}};$$

n_s – концентрация сверхпроводящих электронов.

Для $T_c-T<<T_c$ глубину проникновения магнитного поля можно заменить следующим соотношением

$$\lambda(T)=\lambda_L(0)[T_c/2(T_c-T)]^{\frac{1}{2}}.$$

Для критической температуры по теории Гинзбурга и Ландау

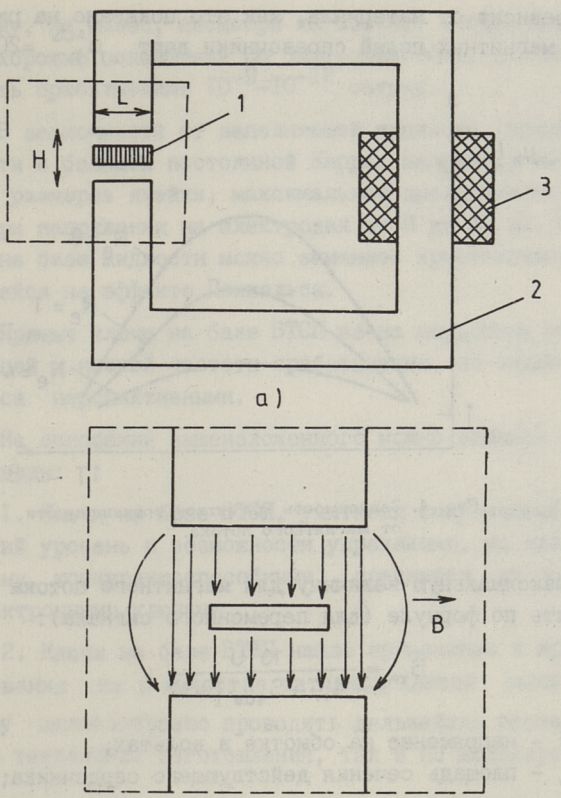
$$d_c=\sqrt{5}\lambda. \qquad (7)$$

Теперь оценим возможности управления ВТСП ключами.

Рассмотрим ситуацию, где СП линия находится в зазоре ферромагнитного материала. Ситуация показана на рис. 3. Если выполняется условие $H<<L$, СП находится в зазоре. Его поверхностные слои являются параллельными к поверхностям зазора, следовательно, направление $\vec{B}$ вектора совпадает с направлением нормвектора поверхности. Следовательно, величины магнитной индукции в зазоре и ферромагнитном материале одинаковые. Поэтому можем написать:

$$\oint H_\ell d\ell = \oint j_n dS. \qquad (8)$$

Напряженность магнитного поля считается равной по всей площади зазора и $H_{Fe}=B/(\mu_0\mu_{Fe})$. В остальной окружности $H_*=B/(\mu_0\mu_*)$. Определим часть контура длиной ℓ_{Fe} в остальной окружности ℓ_*. Циркуляция определяется следующим образом:



а)

б)

Рис. 3. Магнитная управляющая схема:
а) 1 – СП, 2 – ферромагнитный материал,
3 – управляющая намотка;

б) магнитное поле в воздушном зазоре (выделено штриховыми линиями на рисунке а).

$$H_{Fe}\ell_{Fe} + H_{*}\ell_{*} = Ni. \qquad (9)$$

где N – число витков обмотки возбуждения и i ток.

Итак, для постоянного тока:

$$B = \mu_0 i \frac{N}{\frac{\ell_{*}}{\mu_{*}} + \frac{\ell_{Fe}}{\mu_{Fe}}}. \qquad (10)$$

μ_{Fe} зависит от материала, как это показано на рис. 4. Для слабых магнитных полей справочники дают μ_*^{ALG} =200–20000.

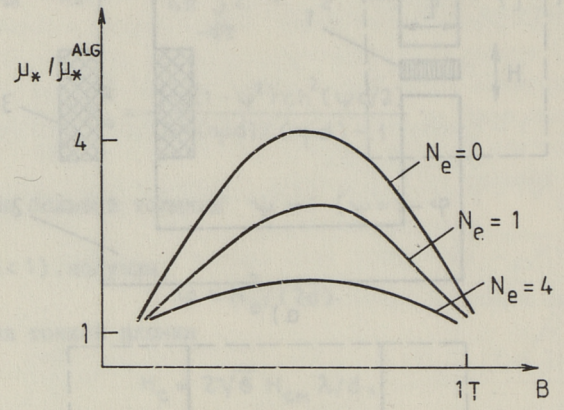


Рис. 4. Зависимость магнитной проницаемости от магнитного потока.

Максимальную величину для магнитного потока можно вычислить по формуле (для переменного сигнала):

$$B_m = \frac{10^4 U}{4{,}44 \cdot S_{эф} \cdot f \cdot w}, \qquad (II)$$

где U – напряжение на обмотке в вольтах;
$S_{эф}$ – площадь сечения действующего сердечника;
f – частота.

Для оценки требуемых витков обмотки и напряжения был сделан расчет на базе ферритного сердечника ПII0 (ШII0П), в которой используется феррит типа 2500 НС или 2500 НМС.

На основании вышесказанного определим напряжение U по отношению к десяти виткам для магнитного потока I Т (I Т = 10^4 Гс) на частоте I КГц. Расчеты показали: U = = 4,4 В/по I0 виткам. Индуктивность обмотки I6,6 мГенри.

Итак, возможность управления СП с помощью магнитного поля является весьма приемлемой, но сама конструкция выполнения управляющей цепи неудобная.

Следующим методом управления ВТСП может быть метод с помощью лазерного луча (тепловое управление) через ячейки Керра. Однако реальной возможности данный метод, по-

видимому, не имеет, несмотря на то, что ячейка Керра имеет очень хорошие показатели по быстродействию, как например, скорость срабатывания 10^{-9}-10^{-12} секунд.

В зависимости от заполняющей жидкости (применяются жидкости с большой постоянной Керра, например, нитробензол) и размеров ячейки, максимальная прозрачность достигается при напряжении на электродах от 3 до 30 кВ. Ячейку Керра на базе жидкости можно заменить кристаллической, базирующейся на эффекте Поккельса.

Прямые ключи на базе ВТСП из-за неудобных управляющих цепей и низкой частоты срабатывания, по-видимому, не являются перспективными.

На основании вышеизложенного можно сделать следующие выводы:

I. Ключи на базе ВТСП, учитывая современный технологический уровень и возможности управления, не являются, по-видимому, конкурентоспособными, сравнивая их с известными электронными ключами.

2. Ключи на базе ВТСП нашли применение и круг использования их в качестве датчиков/ключей расширяется. Поэтому целесообразно проводить дальнейшие исследования как по технологии изготовления, так и по моделированию.

Литература

I. Ранг Т., Коэл А., Орро С. Дополнение к методике ВТСП. Обзор по соединениям и ключам. Этап I. Техническая информация. Таллинн, 1989.

2. Tzeng Y., Cutshaw C., Roppel T., Wu C., Tanger C.W., Belsen M., Williams R., Czekala L. High-temperature superconductor opening switch // Appl. Phys. Lett. 1989. Vol. 54, N 10. P. 949-950.

3. Глазнам Л.И., Кошелев А.Е., Лебедь А.Г. Резистивный переход и критические поля сверхпроводящих керамик//ЖЭТФ. Т. 94. 1988. С. 259-269.

4. Аронзон Б.А., Гершанов Ю.В., Мейлихов Е.З., Шапиро В.Г. Влияние магнитного поля на вольт-амперную характеристику резистивного состояния керамики $YBa_2Cu_3O_{6,9}$ вблизи перехода. Сверхпроводимость: физика, химия, техника. Т. 2. 1989. С. 83-88.

T. Rang, A. Koel

Kõrgtemperatuurilisel ülijuhtivusel baseeruva võtme mudel

Kokkuvõte

Artiklis on kirjeldatud KTÜJ-l baseeruva võtme tööd. On esitatud võtme aseskeemmudel ning välja pakutud matemaatiline mudel võtme karakteristikute arvutamiseks. Töö lõpus on antud hinnang võtme tüürimisvahenditele.

T. Rang, A. Koel

Modelling the Superconductivity Switch

Abstract

The principle circuit and the mathematical model for SC switch have been introduced. The critical meaning of the control circuit has been proposed.

The model could be used in non-linear circuit analysis programs (TRANZ-TRAN) for the analysis of SC circuits.

№ 708

TALLINNA TEHNIKAÜLIKOOLI TOIMETISED
ТРУДЫ ТАЛЛИННСКОГО ТЕХНИЧЕСКОГО УНИВЕРСИТЕТА

ЭЛЕКТРОТЕХНИКА И АВТОМАТИКА XXXIX

УДК 519:713

Построение проверяющего теста для сети автоматов. Кеэваллик А., Каширова Л., Круус М. - Труды Таллиннского технического университета, 1990, № 708, с. 3-19.

В статье исследованы проблемы построения проверяющего теста для дискретного управляющего устройства, реализованного в виде сети взаимодействующих компонентных автоматов на базе программируемых логических матриц. Доказано, что предложенный тест обнаруживает неисправности рассматриваемых классов как на соединениях между компонентами сети, так и внутри самих компонент. Приведена верхняя оценка длины построенного теста. Теоретические выводы иллюстрируются.

Таблиц - 6, рисунков - 5, библ. наименований - 5.

УДК 62.507

О информационных операторах конечного автомата. Кеэваллик А., Лаусмаа Т. - Труды Таллиннского технического университета, 1990, № 708, с. 20-26.

Статья посвящена изучению свойств операторов $\mathfrak{m}$ и M в алгебре пар разбиений. Показано, что основные информационные неравенства, доказанные Ю. Хартманисом и Р. Стирнзом, для конечного автомата действительны и в случае входных слов длиной k.

Библ. наименований - I.

УДК 62.507

Преобразование описаний на языке VHDL от функциональных к структурным. Беркман Б., Судницын А., Удрэ Ю. - Труды Таллиннского технического университета, 1990, № 708, с. 27-44.

В работе предложен подход к получению структурных опи-

саний цифровых систем по абстрактным описаниям их функционирования при помощи разделения на управляющую и операционную части. Приводится пример с описанием специализированного цифрового устройства, вычисляющего наибольший общий делитель двух целых чисел по алгоритму Эвклида.

Рисунков - 10, библ. наименований - 10.

УДК 681.32

Новые методы проектирования тестов для обнаружения неисправностей в дискретных объектах. Аланго В., Коньт Т., Убар Р. - Труды Таллиннского технического университета, 1990, № 708, с. 45-62.

Предлагается новый обобщенный подход к построению и анализу тестов для широкого класса дискретных объектов, основываясь на теории альтернативных графов. Альтернативные графы обеспечивают эффективную и универсальную модель для задания структуры, функций и неисправностей дискретных объектов на различных уровнях их представления, в частности, на уровнях транзисторного, логического, функционального и процедурного представлений. Предложены универсальные методы генерации тестов для этих уровней. Показано единство модели для решения широкого класса задач технической диагностики, что влечет за собой возможность замены множества различных библиотек моделей для этих задач единственной универсальной библиотекой.

Таблиц - 2, рисунков - 8, библ. наименсваний - 9.

УДК 681.32

Система контроля микропроцессорных устройств. Убар Р., Лохуару Т., Мяннисалу М., Пукк П., Ванамельдер Э. - Труды Таллиннского технического университета, 1990, № 708, с. 63-77.

Предлагается новый метод генерирования тестов для микропроцессорных устройств, основывающийся на применении концепции альтернативных графов. Тесты генерируются путем символьного моделирования и организуются компактно в виде символьной программы и массива данных. Предлагается новая архитектура системы контроля, ориентированная на аппаратурное генерирование тест-векторов на основе заданной ком-

пактной тест-программы и массива данных. Введены возможности организации асинхронного обмена данными между тестером и объектом контроля, что дает возможность контролировать объекты, быстродействие которых выше, чем у системы контроля.

Таблиц - I, рисунков - 4, библ. наименований - 6.

УДК 681.32

Система синтеза тестовых программ для дискретных объектов диагностирования (ОД). Заугаров В., Саарепера М., Сторожев С. - Труды Таллиннского технического университета, 1990, № 708, с. 78-89.

Описывается система автоматизированного проектирования тестовых программ для проверки микропроцессорных БИС и устройств, построенных на их основе. Теоретической основой системы является теория альтернативных графов. Приводится классификация исходных данных для этой системы и описываются способы ввода их в ЭВМ. Перечисляются и характеризуются основные этапы работы системы.

Рисунков - 2, библ. наименований - I.

УДК 681.32

Автоматизированное построение программ самодиагностики микропроцессоров. Сторожев С. - Труды Таллиннского технического университета, 1990, № 708, с. 90-97.

Предлагается метод автоматизированного проектирования программ самодиагностики микропроцессорных БИС, а также микроЭВМ. Метод основывается на описании системы команд проверяемого устройства альтернативными графами. На основе альтернативных графов синтезируются абстрактные тест-векторы, из которых затем компилируется тест-программа. Предложена методика создания программного тестера на языке заданного микропроцессора. Программный тестер и сгенерированная тест-программа в совокупности образуют программу самодиагностики.

Рисунков - 2, библ. наименований - 3.

УДК 681.32

Декомпозиционный метод представления объектов диагностирования моделями обобщенных альтернативных графов. Заугаров В. - Труды Таллиннского технического университета, 1990, № 708, с. 98-102.

В статье предлагается метод описания объектов диагностирования (ОД), который повышает возможность автоматизации построения тестов для широкого класса ОД: от микропроцессорных БИС до структур, содержащих микропроцессорные БИС в качестве компонентов. Метод основывается на представлении ОД совокупностью моделей обобщенных альтернативных графов.

Рисунков - 2, библ. наименований - 3.

УДК 621.382

Модель межсоединений/линии передачи на базе ВТСП. Орро С., Ранг Т. - Труды Таллиннского технического университета. 1990, № 708, с. 103-108.

Описана модель и приведен анализ модели межсоединений линии передачи на базе ВТСП. Модель базируется на теории волнопроводимости и реализуется с помощью быстрого преобразования Фурье.

Рисунков - 1, библ. наименований - 2.

УДК 621.382

Модель ключа на базе ВТСП. Ранг Т., Коэл А. - Труды Таллиннского технического университета, 1990, № 708, с. 109-116.

Описывается принцип действия, эквивалентная схема и математическая модель ключа на базе ВТСП. Дается оценка управляющим цепям для управления ВТСП ключами.

Рисунков - 4, библ. наименований - 4.

Цена 3 руб.