УДК 681.518.54
В.Г. Новоселов, канд. физ.-мат. наук, профессор
А.В. Борисевич, студент
Севастопольский национальный технический университет
Студгородок, г. Севастополь, Украина, 99053

ПОСТРОЕНИЕ ПРОВЕРЯЮЩЕГО ТЕСТА ДЛЯ СОВРЕМЕННЫХ ПЛМ

Рассматривается решение задачи синтеза полного проверяющего теста для синхронных цифровых схем, реализованных на сложных программируемых логических матрицах (CPLD). Даны алгоритмы построения тестов для комбинационной части схемы и нахождения установочной последовательности. Приведены результаты тестирования алгоритмов для типовых схем, реализованных на CPLD семейства Altera Acex1K.

Введение. Применение программируемых логических интегральных схем (ПЛИС) позволяет создавать многофункциональные цифровые устройства, к которым предъявляются требования высокой надежности, малых габаритов и большого быстродействия. По этой причине ПЛИС в последнее время стали такими же стандартными компонентами схем, как микроконтроллеры. Наибольшее распространение получили ПЛИС от фирм Altera, XilinX и Actel. Построению тестов для CPLD посвящено ряд работ зарубежных авторов, в которых используется модели неисправностей на уровне межрегистровых передач (RTL) и построение тестовых наборов базируется на моделировании неисправностей.

Целью данной работы является алгоритмическое обоснование и проработка вопросов реализации метода построения проверяющего теста для ПЛИС с памятью, который оперирует со схемой на структурно-функциональном уровне и строит тест аналитически, используя операцию вычитания интервалов.

Структура CPLD. Классические ПЛМ (PLD), описанные в [1], в данный момент полностью вытеснены более совершенными приборами, в которых содержится несколько логических блоков, аналогичных по структуре PLD, связанных внутри ПЛИС с помощью отдельной матрицы, которая также коммутирует входы и выходы блоков с внешними выводами ПЛМ. Самыми простыми и распространенными структурами данного типа являются сложные программируемые логические матрицы – CPLD.

Рассмотрим структуру ПЛИС типа CPLD на примере ПЛИС семейства XC9500 фирмы XilinX, описанной в [2]. CPLD других семейств и производителей (в первую очередь фирмы Altera), отличаются лишь в деталях от описанной ниже.

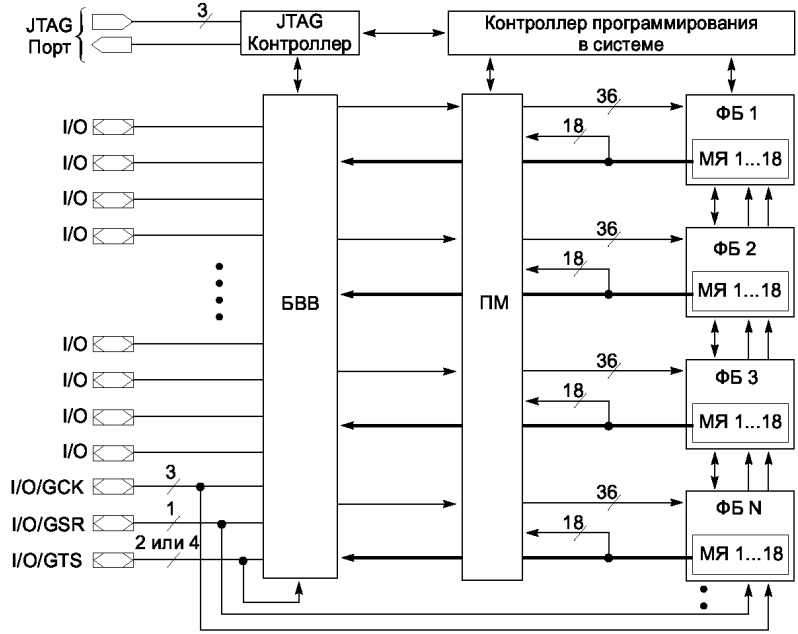

Рисунок 1 – Общая структура CPLD

Каждая CPLD представляет собой систему, состоящую из множества функциональных блоков (ФБ) и блоков ввода-вывода (БВВ), соединенных переключающей матрицей (ПМ) (рисунок 1). БВВ обеспечивают буферизацию всех входов и выходов CPLD. Каждый ФБ содержит матрицу И, позволяющую формировать произведения *Nx* сигналов, поступающих от ПМ, (для примера $Nx = 36$) или их инверсий. Также ФБ содержит *Nc* макроячеек (для примера $Nc = 18$), каждая из которых реализует функцию ИЛИ либо ИЛИ-НЕ, а также Исключающие ИЛИ от произведений из матрицы И. Выходной сигнал макроячейки может быть подан на матрицу ПМ либо непосредственно или через триггер. Следует заметить, что в CPLD всегда применяются динамические D-триггера, причем все их входы (включая входы асинхронной установки и тактовый вход) также могут управляться сигналами, формируемыми матрицей И в ФБ.

Модели неисправностей. Схема, реализуемая в CPLD, может быть представлена в виде автомата с вектором переменных состояния $W = \{w_1, \ldots w_q\}$, задающего значения сигналов на выходах триггеров схемы; вектором входов $X = \{x_1, \ldots x_n\}$ и вектором выходов $Y = \{y_1, \ldots y_m\}$. Функционирование схемы задается с помощью вектор-функции переходов

$W(X,W)$, описывающей сигналы на информационных входах триггеров и вектор-функции выходов $Y(X,W)$.

Как видно из структуры CPLD, для построения проверяющего теста модель, рассмотренная в [1], непосредственно неприменима. Это объясняется тем, что CPLD представляет собой многоуровневую схему в элементом базисе НЕ - И - ИЛИ - Исключающие ИЛИ. В работе [3] показано, что если в структуре схемы есть перемножение двух ДНФ $f_{ij} = f_i \wedge f_j$, в результате неисправности в f_{ij} могут появиться конъюнкций, исходно не присутствующие в исправной функции f_{ij}. Для тестирования неисправностей типа появления интервалов будет использован подход, описанный в [3].

Рассмотрев по аналогии с [1] различные физические неисправности в CPLD, можно прийти к выводу, что все они могут быть сведены к следующим логическим изменениям функции $f_i(X,W) \in W \cup Y$, ДНФ которой получена при анализе структуры CPLD.

1. Расширение интервала в f_i по одной или нескольким внутренним переменным. Данная неисправность соответствует исчезновению элемента связи в ПМ или в матрице И, либо появлению элемента связи в данных матрицах, если рассматривается инверсия функции. Также данное изменение f_i имеет место при константных неисправностей типа =1 на инверторах или буферах входов схемы или триггеров.

2. Сокращение интервала в функции схемы по одной или нескольким внутренним переменным. Данная неисправность соответствует появлению элемента связи в ПМ или в матрице И, либо исчезновению элемента связи в данных матрицах, если рассматривается инверсия функции.

3. Появление интервала, если в некоторой части схемы происходит перемножение функций, содержащих такие конъюнкции α_i и α_j, что в одной из них содержится переменная x в прямом значении, а в другом – с инверсией $\bar{x}$. В случае, если $x \equiv 1$ и $\bar{x} \neq 0$ (данная неисправность вполне может иметь место, поскольку схемотехнически x и $\bar{x}$ формируются разными логическими элементами) или $x \neq 0$ и $\bar{x} \neq 0$, то произведение $\alpha_i \alpha_j \neq \emptyset$.

Преобразование булевых функций при обработке структуры CPLD. Для тестирования неисправностей типа появления интервалов требуется расширение машинного представления интервала, данного в [4]. Опишем одно из возможных решений. Назовем интервал, образованный произведением конъюнкций α и β, вида $\gamma' = \alpha_i(x_j) \wedge \beta_k(\overline{x_j})$ фиктивным по переменной x_j. При тестировании необходимо проверить появления двух интервалов в функции: $\alpha_i(\overline{x_j}) \wedge \beta_k(\overline{x_j})$ и $\alpha_i(x_j) \wedge \beta_k(x_j)$.

Пусть интервал γ описывается тройкой векторов $\gamma = (A, B, P)$, где A – вектор значений внешних переменных, B – вектор-маска внешних переменных, P – вектор-маска фиктивных переменных ($P_j = 1$ если интервал образован произведением $\gamma = \alpha_i(x_j) \wedge \beta_k(\overline{x_j})$). Если вектор $P \neq 0$, то это означает, что $\gamma = \emptyset$. Положим, что если $P_i = 1$, то $A_i = 0, B_i = 1$.

Введение фиктивных интервалов в ДНФ требует модификации операции пересечения интервалов $\gamma = \alpha \wedge \beta$, рассмотренной в [4].

Компоненты результата вычисляются по соотношениям:

$$\begin{cases} P_\gamma = (B_\alpha B_\beta \wedge (A_\alpha \oplus B_\beta)) \vee (P_\alpha \vee P_\beta); \\ A_\gamma = (A_\alpha \vee A_\beta)\overline{P_\gamma}; \\ B_\gamma = (A_\alpha \vee A_\beta) \vee P_\gamma. \end{cases}$$

При проверке k-кратных неисправностей, если $|P_\gamma| > k$, то $\gamma = \emptyset$.

Вычитание интервалов $\gamma = \alpha \# \beta$ также требует корректировки. Рассмотрим частные результаты этой операции:

1. Если $\alpha \subseteq \beta$ или $P_\alpha \neq 0$, то $\alpha \# \beta = Ж$.

2. Если $\alpha \wedge \beta = \emptyset$ или $P_\beta \neq 0$, то $\alpha \# \beta = \alpha$.

3. В противном случае $\alpha \# \beta = \gamma$, где γ – вычисляется по алгоритму, изложенному в [4], без учета значений вектора P.

Операции проверки на поглощение и равенство интервалов, а также операции симметрирования и расширения интервалов реализуются согласно [4] без учета вектора P.

Рассмотрим операции, с помощью которых можно получить ДНФ тестируемых функций из описания структуры CPLD.

1. Дизъюнкция A и B. Результатом C является дизъюнкция операндов $C = A \vee B$.

2. Конъюнкция A и B. Компоненты результата C формируются из интервалов: $C = \bigvee_i \alpha_i \beta_i$, где произведение интервалов $\alpha_i\beta_i$ вычисляется по вышеописанному правилу, учитывающему появление фиктивных интервалов.

3. Инверсия A. Можно показать, что появление интервалов в функции A всегда приводит к немаскируемому сокращению интервалов результата инверсии C. Таким образом, для получения инверсии функции A необходимо удалить из A все фиктивные интервалы и получить ДНФ инверсии, как результат вычитания $\overline{A} = 1 \# \bigvee_i \alpha_i$.

4. Исключающие ИЛИ A и B. Данная операция эквивалентна функции $C = A\overline{B} \vee \overline{A}B$, с помощью которой обработка элемента Исключающие ИЛИ сводится к соответствующему использованию преобразований 1, 2 и 3.

5. Элемент с тристабильным выходом, в котором функция A задает входной сигнал, а функция B – сигнал разрешения (по $B = 0$ выход элемента переходит в Z-состояние). В первом приближении, такой элемент эквивалентен конъюнкции своих операндов: $C = A \wedge B$. Можно допустить, что при $B=0$ значение тристабильного выхода $C=0$, поскольку в процессе тестирования (снятия сигналов), выход элемента не будет нагружен ни на одну из шин питания, а при этом вход контроллера тестирования (например, выполненного согласно спецификации JTAG) в большинстве случаев рассматривается как нагрузка на нейтральный полюс источника питания.

Построение теста для комбинационной части. Как и в [1], рассматриваются неисправности кратности не более $k=3$. Для каждой функции F при синтезе теста строится список $Z = \bigcup_{j=0,k-1} Z_j$ допустимых изменений интервалов. Каждое подмножество Z_i представляет собой список несущественных неисправностей кратности i и содержит все несущественные сокращения и расширения интервалов списка Z_{i-1}. Список Z_0 – интервалы, приписанные проверяемой функции (без учета фиктивных интервалов). При построении списка Z_i в него не включаются все интервалы, содержащиеся в списках $Z_0 \dots Z_{i-1}$.

При построении всех несущественных расширений элемента списка Z_{ik} рассматривается интервал Z_{ik}^q, полученный из элемента списка Z_{ik} расширением по его внешней переменной q. Если $Z_{ik}^q \# \bigvee_j F_j = \emptyset$, то расширение допустимо и Z_{ik}^q заносится в список Z_{i+1}. Для построения всех несущественных расширений элемента списка Z_{ik} вычисляется $Z_{ik}^\circ = МПК(Z_{ik} \# \bigvee_j F_j)$. Все допустимые сокращения Z_{ik} – это Z_{ik}°, а также всевозможные его расширения по множеству внешних переменных, которые являются внутренними для Z_{ik}.

Тестовые наборы, проверяющие интервалы функции на расширение определяются как $T_{расш} = Z_{ik}^q \# \bigvee_j F_j$, где Z_{ik}^q – интервал, полученный из элемента списка Z_{ik} расширением по его внешней переменной q. Для определения множества тестовых наборов $T_{сокр}$, каждый проверяемый на сокращение интервал F_i представляется в виде: $F_i = F_i^* \vee F_i'$ где F_i^* – область,

на которой невозможно маскирование ($F_i^* = F_i \# (\bigvee_l Z_l \setminus \{F_{ik}\})$, где F_{ik} – допустимые расширения интервала F_i) и $F_i' = F_i \# F_i^*$ – область, которая маскируется. Тестовый набор состоит из объединения $T_{сокр} = T_i^* \vee T_i'$, где T_i^* удовлетворяет соотношению $МПК(T_i^*) = МПК(F_i^*)$, где $МПК(F)$ – минимальный покрывающий интервал для интервалов функции F; а T_i' состоит из точек, каждая из которых принадлежит одному интервалу из F_i' . В список Z_i также включаются допустимые появления α_j и $\hat{\alpha}_j$ фиктивных интервалов α'_j, для которых $|P_{\alpha'_i}| = i$ и $\alpha_j \# \bigvee_i F_i = \emptyset$ или $\hat{\alpha}_j \# \bigvee_i F_i = \emptyset$, где α_j – интервал, полученный из α'_j заменой всех фиктивных переменных x_j на их прямые значения, $\hat{\alpha}_j$ – на инверсные.

Появления фиктивных интервалов α'_j для которых $|P_{\alpha'_i}| = l$ тестируются при рассмотрении ошибок кратности $l=k$, поскольку $\alpha'_{j_j} = 0$ для $l<k$. Множество интервалов, тестирующих появление α'_j получается как $T_{появл} = (\alpha_j \vee \hat{\alpha}_j) \# \bigvee_i F$.

Важной подзадачей является нахождение такого множества наборов T_i^* минимальной мощности, для которого $МПК(T_i^*) = МПК(F_i^*)$. Ее решение производится в два этапа: сначала осуществляется вычисление T_i^* по F_i^* , если искомый T_i^* содержит не более 2-х точек (это выполняется в 95 % случаев). Если решение на первом этапе не получено, производится граничный перебор.

Вычисление набора T_i^* выполняется путем перебора интервалов в F_i^* . Обозначим за $П$ множество внутренних переменных $МПК(F_i^*)$. Возьмем два различных интервала $\alpha_k, \alpha_j \in F_i^*$. Если в α_k и α_j есть общие внешние переменные с одинаковым значением и входящие в $П$, то берем другую пару интервалов. Иначе, определяем решение $t_1 = \alpha_k$ и $t_2 = \alpha_j$. Из $П$ исключаются все переменные, которые являются внешними для t_1 и t_2, и имеют различное значение. Далее доопределяем каждую переменную $x_h \in П$ в векторах t_1 и t_2 таким образом, чтобы выполнялось $МПК(T_i^*) = МПК(F_i^*)$.

В случае перебора берутся три различные точки $T_i^* = \{t_1, t_2, t_3 \in F_i^*\}$. Для всех точек из T_i^* осуществляется перебор их всевозможных различных координат внутри F_i^*. Если перебор закончен и требуемый МПК не построен, то в набор включается еще одна точка и процесс продолжается снова.

Для минимизации длины теста используются следующий простой метод: перед тем, как включить в тест T набор из некоторого множества интервалов Q, каждый интервал Q_j проверяется на пересечение с интервалами теста T_i. Если пересечение найдено, то набор T_i в тесте заменяется не результат пересечения $T_i \wedge Q_j$, в противном случае в тестовый набор выбирается интервал Q_k максимального ранга.

Построение теста автомата. Далее модифицируем методы поиска неисправностей в автоматах, описанные в [1], учтя, что входы установки в 0 и 1, а также тактовый вход каждого триггера обладают асинхронными свойствами.

С каждым триггером i схемы (переменной состояния w_i) связана совокупность функций (R_i, S_i, D_i, C_i), где $R_i = r_i(X, W)$ – состояние входа установки в 0, $S_i = s_i(X, W)$ – состояние входа установки в 1, $D_i = d_i(X, W)$ – состояние информационного входа триггера и $C_i = c_i(X, W)$ – состояние тактового входа.

Пусть w_i' – неисправная функция переходов, y_j' – неисправная выходная функция. Соответственно, W' – вектор-функция переходов, содержащая функцию с ошибкой, и Y' – выходная вектор-функция, с введенной неисправностью.

Перед генерацией установочной последовательности все триггеры схемы должны быть установлены в начальное состояние, независимое от предыдущего. Если в схеме существует хотя бы один триггер, для которого невозможна непосредственная установка начального состояния (т.е. триггер, для которого ни функция R_i, ни функция S_i не могут быть представлены как функции, не зависящие от W), то данная схема считается нетестируемой.

Определить начальное состояние схемы W_R и входной вектор сброса X_R можно, используя следующий алгоритм.

1. Для каждого триггера i определяется функция $U_i(X)$ следующим образом: $U_i = R_i$, если $R_i \neq \text{const}$ и R_i не зависит от W, в противном случае $U_i = S_i$.

2. Вектор W_R формируется следующим образом: если $U_i = R_i$, то $W_{R_i} = 0$, иначе $W_{R_i} = 1$.

3. Вектор сброса X_R определяется как корень максимального ранга уравнения $\bigwedge_i U_i(X) = 1$.

Результатом синтеза теста для комбинационной части схемы является множество элементарных тестов $\{E_j\}$, каждый из которых определяется тройкой вида: $E_j = (f, \sigma_{испр}, T)$, где f – метка функции, к которой относится данная неисправность, $\sigma_{испр}$ – значение на выходе исправной функции и тестовая ситуация $T = \{(X_k^*, W_k^*)\}$ – множество пар (X_k^*, W_k^*) минимально определенных наборов входных и внутренних переменных, тестирующих появление данной неисправности.

Цель построения установочной последовательности – привести исправный и неисправный автомат в состояния, для которых отличаются значения выходных функций y_j. Рассмотрим два случая:

1. Если рассматриваемая неисправность находится в выходной функции, то для ее обнаружения достаточно перевести автомат из состояния W_R в состояние W_t, такое, что $W_t \subseteq W_k^*$, где W_k^* – некоторый вектор состояний в тестовой ситуации неисправности. Для этого требуется построить последовательность входных векторов $\Xi = (X_1, \dots X_t)$, $|\Xi| \to \min$, где каждый вектор X_i удовлетворяет условию $X_i \wedge X_R = 0$.

После того, как автомат будет переведен в состояние W_t, согласно определению тестовой ситуации: $Y(X_k^*, W_k^*) \neq Y'(X_k^*, W_k^*)$. Окончательным тестом для неисправности является последовательность $(X_1, \dots X_t, X_k^*)$.

2. Построение тестовой последовательности для тестирования функций переходов осуществляется аналогично предыдущему случаю.

После того, как автомат будет переведен в состояние W_t, следующее состояние, полученное под воздействием X_k^*, будет ошибочным: $W(X_k^*, W_k^*) \neq W'(X_k^*, W_k^*)$ и далее, в общем случае, функционирование исправного и неисправного автомата различаются. Различие в функционировании требуется передать на выход схемы *Y*. Для этого требуется достроить такую последовательность $\Xi^\circ = (X_1^\circ, \dots X_q^\circ)$ $|\Xi^\circ| \to \min$, для которой $Y(X_q^\circ, W_q^\circ) \neq Y'(X_q^\circ, W_q^\circ)$. Окончательным тестом для неисправности является последовательность $(X_1, \dots X_t, X_k^*, X_1^\circ, \dots X_q^\circ)$.

Переход между состояниями осуществляется на основе итерационного вычисления состояния триггеров, что необходимо для учета асинхронных свойств входов C, R и S. Введем два вектора состояния выходов триг-

геров: текущее состояние $W(t)$ и новое состояние $W(t+1)$. Для моделирования работы тактовых входов триггеров также введем вектор их предыдущего состояния $C(t)$.

Алгоритм вычисления следующий:

0. $k = 1$

1. Для каждого триггера i схемы выполнять:

1.2. Если $r_i(X(t),W(t)) = 1$, то $W_i(t+1) = 1$.

1.3. Если $s_i(X(t),W(t)) = 1$, то $W_i(t+1) = 0$.

1.4. Если $C_i(t) = 0$, и $c_i(X(t),W(t)) = 1$, то $W_i(t+1) = d_i(X(t),W(t))$, иначе $W_i(t+1) = W_i(t)$.

2. Если $W(t+1) = W(t)$, то закончить моделирование, иначе выполнять:

2.1. Если $k > K$, то данный вектор $X(t)$ вызывает генерацию в схеме и состояние $W(t+1)$ – считается неопределенным и не рассматривается, иначе $k = k+1$ и переход к п. 1.

Число K – максимальное число итераций моделирования.

Поскольку бит W_j в векторе меняется в том случае, когда $C_j(t) = 0$ и $C_j(t+1) = 1$, то следующее состояние $W(t+1)$ зависит не только от предыдущего состояния $W(t)$, но и от предыдущего состояния тактовых входов $C(t)$. Пусть N_C – число различных функций $c_i(X(t),W(t))$, и $q = |W|$ – число триггеров в схеме. В таком случае, автомат в данной модели содержит 2^{N_C+q} состояний. Код состояния – определяется как совокупность векторов (W,C).

Алгоритм поиска установочной последовательности использует волновой поиск в графе (поиск в ширину):

0. В очередь заносится начальное состояние $S_0 = (W_R, C_R)$.

1. Из очереди выбирается текущее состояние $S_i = (W(t), C(t))$.

1.1. Если $W(t)$ – требуемое конечное состояние W_k (или $W(t) \subseteq W_k$), то процесс поиска заканчивается и восстанавливается обратный путь.

1.2. Перебираются всевозможные входные наборы X_j и определяются переходы из текущего состояния S_i в последующие состояния $S_k = (W(t+1), C(t+1))$, где значения $W(t+1)$ и $C(t+1)$ определяются на основе моделирования состояния триггеров по входному вектору X_j и $W(t)$.

1.2.1. Если полученное состояние S_k не рассматривалась, оно помещается в очередь и пара (S_i, S_k) сохраняется вместе с X_j для восстановления обратного пути.

1.3. Переход к п. 1.

Для хранения информации о переходах между состояниями автомата, полученной при поиске пути, эффективно использовать хеш-таблицу. Ключом к таблице является состояние, в которое осуществляется переход S_k. Элемент таблицы – тройка (S_k, S_i, X_j), где S_i – состояние, из которого осуществляется переход, X_j – условие перехода.

Практические результаты. Нами была выполнена программная реализация алгоритма построения полного проверяющего теста для CPLD. Разработанная программа предназначена для автоматической генерации тестов для схем средней сложности, реализованных на CPLD фирмы Altera.

Проектирование схем на CPLD данной фирмы производится в специализированной САПР – Quartus II. С помощью данного пакета можно осуществить полный синтез схемы на ПЛИС из описания на языках VHDL или Verilog, а также из схем на логических элементах серии 74 (К155). САПР Quartus II в процессе компиляции, наряду с файлами кодов конфигурации CPLD, загружаемыми непосредственно в программатор, генерирует также текстовое описание структуры результата компиляции в виде EQN-файла.

EQN-файл содержит описание функции каждой макроячейки в виде суперпозиции сигналов матрицы ПМ. Функция макроячейки задается в виде операторов, которые могут описывать дизъюнкцию (<A> # <B>), конъюнкцию (<A> & <B>), инверсию (!<A>), Исключающие ИЛИ (<A> $ <B>). Также существуют синтаксические элементы для описания триггеров и тристабильных элементов. Особенностью формата EQN-файла является отсутствие деклараций используемых сигналов. Данное свойство требует многократного транслирования входного файла до тех пор, пока все выходные и переходные функции y_j и w_j не будут выражены через входные переменные x_i и w_i.

Реализация описанных алгоритмов выполнена на языке C++ и содержит около 5000 строк кода. В программу входят:

– транслятор с языка EQN-описаний, построенный на основе метода рекурсивного спуска;

– модуль построения теста для комбинационной части модели CPLD;

– модуль для генерации установочных последовательностей.

Тестирование программы проводилось на тесте ITC'99, который содержит схемы в виде описаний на языке VHDL, представляющие собой наиболее типовые фрагменты систем, синтезируемых на CPLD [5]. Рассматривались схемы, для которых $|W| + |X| \leq 64$, что связано с ограничениями текущей программной реализации. Схемы теста ITC'99 компилировались в Quartus II v. 5.0 под ПЛИС семейства Acex1K. Некоторые показатели полученных тестов даны в таблице 1.

Таблица 1 – Статистика результатов синтеза теста для некоторых схем библиотеки ITC99, реализованных в CPLD семейства Altera Acex1K

Схема	Число функций	Общее число входов и триггеров	Общее число интервалов	Среднее число внешних переменных в функциях	Число ошибок	Общая длина теста	Средняя длина набора установки	Время работы, сек.
b01	7	9	65	3,73	201	89	2,47	<1
b02	5	7	13	3,55	50	55	3,20	<1
b03	34	36	3914	8,75	11147	520	5,36	72
b05	70	37	5138	12,73	24614	28707	6,44	295
b06	14	12	36	2,69	130	92	3,53	<1
b07	49	44	247	9,62	1429	326	3,54	<1
b08	25	32	175	6,73	1180	281	4,96	2
b09	29	31	497	12,70	5981	7137	17,97	38
b10	23	30	122	6,53	685	866	3,73	1
b11	37	40	1871	14,80	13895	20209	5,65	60

Выводы. В данной работе разработаны высокоэффективные алгоритмы построения проверяющего теста для ПЛИС, основанные на подходах изложенных в [1]. Полученные практические результаты подтверждают высокое быстродействие метода и применимость к схемам средней размерности.

Дальнейшая работа направлена на увеличение числа входов обрабатываемых функций путем применения методов декомпозиции. Также проводится работа по оптимизации реализаций алгоритмов по быстродействию, что составит предмет дальнейших исследований в данном направлении.

Библиографический список

1. Бутаков Е.А. Диагностика программируемых логических матриц / Е.А. Бутаков, М.Б. Волынский, В.Г. Новоселов. — М.: Радио и связь, 1991 — 160 с.
2. Перепрограммируемые в системе ПЛИС CPLD семейства XC9500 (пер. с англ.) — http://www.plis.ru/pic/pict/File/9500_rus.pdf.
3. Новоселов В.Г. Синтез полных проверяющих тестов И–НЕ схем произвольной структуры на базе модели расширения и исчезновения интервалов / В.Г. Новоселов // Тр. четвертой междунар. конф. «CADD'2001» сент. 2001, г. Минск. — Минск: Ин-т техн. кибернетики НАН Беларуси, 2001. — Т. 3. — С. 96–103.
4. Новоселов В.Г. Дискретная математика в задачах и примерах. Ч. 2 / Учеб. пособие / В.Г. Новоселов, Т.В. Пластун, А.В. Скатков. — Севастополь: Изд-во СевНТУ, 2002. — 121 с.
5. Corno F. RT-Level ITC 99 Benchmarks and First ATPG Results / F. Corno, M. Sonza Reorda, G. Squillero // IEEE Design & Test of Computers. — 2000. — July-August. — P. 45–53.

Поступила в редакцию 21.12.2005 г.