ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1994
Seria: ELEKTRONIKA z. 3 Nr kol. 1254

Krzysztof GUCWA
Instytut Elektroniki
Politechnika Śląska

WYKORZYSTANIE METODY SYMULACJI DO OCENY POKRYCIA USZKODZEŃ W UKŁADACH CYFROWYCH

Streszczenie. W artykule przedstawiono niektóre problemy związane z wykorzystaniem metod symulacyjnych do oceny skuteczności pokrycia uszkodzeń przez zbiory pseudoprzypadkowych wektorów testowych. Przedstawiono krótki przegląd klasycznych algorytmow symulacji oraz opisano symulator stworzony w celu oceny skuteczności pokrycia uszkodzeń przez zbiory wektorów testowych wygenerowanych za pomocą rejestrów ze sprzężeniem liniowym (LFSR) o różnej strukturze wewnętrznej, ze szczególnym uwzględnieniem rejestrów ze sprzężeniami liniowymi zbudowanymi z przerzutników D oraz T. Przedstawiono również przykładowe wyniki uzyskane za pomocą tego symulatora.

Using a simulation method for the evaluation of fault coverage in digital circuits

Summary. Some problems connected with using a simulation method for the evaluation of average fault coverage in a test circuitry with a pseudo-random test vector set applied to the inputs are presented. A brief review of simulation algorithms is given. The article describes a simulator created for evaluating fault coverage in test circuitry stimulated by test vector set generated by Linear Feedback Shift Registers (LFSR) with different structure. Particular stress was put on LFSR using D and T flip-flops. Exemplary results obtained with this simulator are given.

Применение метода моделирования для оценки качества источников тест-векторов для цифровых схем

Резюме. В предлагаемой статьи представлены некоторые вопросы связанные с применением метод моделирования для оценки удельного частного обнаруживаемых повреждений по сравнению с общим количеством помех в схеме. Изложен краткий обзор классических алгоритмов моделирования, а также описана моделирующяя программа разработанна с целью оценки величины этого частного, когда испытываемая схема возбуждается тест-векторами, источником которых является сдвиговый регистр с линейной обратной связью (анг. LFSR - Linear Feedback Shift Register) с различной внутренной структурой. Особое внимание уделено регистрам на Д-триггерах и Т-триггерах. Представлены также образцовые резыльтаты полученные с помощью моделирующей программы.

1. WSTĘP

Miarą skuteczności działania generatora wektorów testowych (ang. Test Pattern Generator) jest pokrycie możliwie największej ilości uszkodzeń układu przy możliwie najkrótszej sekwencji wektorów testowych. Pokrycie definiuje się jako stosunek liczby uszkodzeń wykrywanych przez dany zbiór wektorów testowych do liczby wszystkich uszkodzeń w testowanym układzie. Miarą skuteczności działania kompresora testów jest prawdopodobieństwo niewykrycia błędnej odpowiedzi (ang. aliasing). Ocenę skuteczności działania tych układów przeprowadza się wykorzystując metody probabilistyczne. Jednak takie szacowanie skuteczności działania generatorów musi opierać się na pewnym przyjętym modelu błędów. Najczęściej przyjmuje się model błędów jednakowo prawdopodobnych, lecz, jak pokazano w [4] i [8], takie założenie daje wyniki odbiegające od rezultatów doświadczalnych. Lepszym modelem błędów jest model asymetryczny, jednak i on nie zawsze daje poprawne rezultaty. W zasadzie chcąc uzyskać idealne rozwiązanie generatora lub kompresora należałoby dobrać go do konkretnej struktury układu testowanego wykorzystując metody symulacyjne. Można również wykorzystać metody symulacyjne do szacowania skuteczności dowolnych generatorów i kompresorów przy założeniu pewnych typowych struktur układów testowanych wraz ze zbiorem uszkodzeń, które mogą wystąpić w takim układzie. Powszechnie stosowanym zbiorem układów do oceny metodami symulacyjnymi różnych technik testowania jest zbiór ISCAS [1]. Jest to zbiór kilkunastu typowych układów, jak np. wielobitowe jednostki arytmetyczno-logiczne, komparatory, kodery itp. Są to układy kombinacyjne. Do przeprowadzenia takich badań symulacyjnych można oczywiście wykorzystać typowe symulatory układów cyfrowych, jednakże symulatory ogólnego przeznaczenia zbudowane są w taki sposób, aby symulować szeroką klasę układów, raczej w celu stwierdzenia poprawności ich działania niż symulacji określonych uszkodzeń w tych układach. Powoduje to, że typowe symulatory są bardzo nieefektywne do prowadzenia badań skuteczności generatorów lub kompresorów testów. Podjęto więc próbę stworzenia symulatora układów bazującego na strukturze układów ISCAS85. Klasa układów została bardzo ograniczona, jednakże pozwoliło to wykorzystać specyficzne cechy układów, zwłaszcza generatorów i kompresorów do przyspieszenia procesu symulacji. Szczególne znaczenie ma to w przypadku kompresorów, gdzie nie interesują nas faktyczne stany na wyjściu kompresora po każdym wektorze testowym, a jedynie sygnatura końcowa. Często jedynym interesującym rezultatem jest odpowiedź na pytanie, czy po wprowadzeniu do kompresora ciągu odpowiedzi układu poprawnego i błędnego sygnatury będą się różnić. W związku z powyższym przy konstrukcji symulatora przyjęto następujące założenia:

- Symulator składać się będzie docelowo z generatora wektorów testowych oraz symulatora układu cyfrowego opisanego za pomocą listy połączeń w formacie ISCAS85.
- Możliwa będzie symulacja dowolnych układów kombinacyjnych opisanych przez zbiór w formacie ISCAS85.
- Wielkość symulowanego układu teoretycznie ograniczona będzie jedynie wielkością pamięci i mocą obliczeniową komputera (rozsądny czas symulacji).

- Symulator zostanie w całości napisany w języku C (C^{++}) bez wykorzystywania jakichkolwiek mechanizmów specyficznych wyłącznie dla systemu DOS lub procesorów 80x86. Pozwoli to docelowo przenieść tak utworzony symulator na maszynę o większej mocy obliczeniowej,np. SUN Sparcstation.
- Ponieważ symulator był wstępnie tworzony na komputerze klasy PC z założeniem późniejszego przeniesienia go do innego środowiska, niemożliwe było wykorzystanie mechanizmów środowiska przyjaznego dla użytkownika, co jest obecnie powszechną tendencją. Wszystkie dane do symulacji (struktura układu, rodzaje sprzężeń generatora, kompresora itp.) są zapisywane w odpowiednich zbiorach, które mogą być tworzone lub poprawiane za pomocą dowolnego edytora tekstów.
- Symulator powinien zapewniać symulację możliwie szerokiej klasy generatorów wektorów testowych o sprzężeniach liniowych, zwłaszcza rejestrów LFSR zbudowanych na bazie przerzutników T i D.
- Należy wykorzystać wszelkie możliwości struktury i algorytmów, aby przyspieszyć proces symulacji.

2. ALGORYTMY SYMULACJI USZKODZEŃ

Znanych jest wiele algorytmów symulacji uszkodzeń. Większość z nich wywodzi się lub jest modyfikacją jednego z poniżej przedstawionych [3].

A. Równoległa symulacja uszkodzeń (ang. Paralel Fault Simulation) jest najstarszym wysokoskutecznym algorytmem do symulacji uszkodzeń i pierwszym algorytmem wykorzystującym słowowo zorientowane instrukcje komputera do równoczesnej symulacji wielu uszkodzeń w układzie lub układów pobudzanych różnymi wektorami testowymi. Składa się on z następujących kroków:

1. Uszkodzenia są dzielone na grupy o rozmiarze n. Uszkodzenia z jednej grupy są symulowane równocześnie.
2. Każdy węzeł ma skojarzone ze sobą n+1 wartości logicznych: jedną wartość odpowiadającą wartości logicznej w poprawnym układzie i n wartości odpowiadających wartościom w układzie z uszkodzeniami. Te n+1 wartości jest składowanych na kolejnych bitach w kolejnych m słowach procesora.
3. Gdy obliczany jest stan na wyjściu kolejnej bramki, logiczne operacje słowowe są wykonywane na jej wartościach wejściowych (za każdym razem na m słowach), to oznacza obliczenie stanu bramki w poprawnym układzie oraz w n układach z uszkodzeniami w "tym samym czasie".

Algorytm równoległy jest bardzo efektywny, jeśli chodzi o zużycie pamięci, jest prosty i stosunkowo łatwy do implementacji. Jednakże gdy ilość symulowanych uszkodzeń jest duża, wymagana jest duża liczba cykli obliczeniowych, czego rezultatem jest duża złożoność obliczeniowa.

B. Symulacja równoczesna (ang. Concurrent Fault Simulation). W symulacji równoczesnej wszystkie układy z uszkodzeniami są symulowane w pojedynczym przejściu razem z układem poprawnym. Aby uniknąć powielania opisu układu, zapamiętywane są tylko różnice pomiędzy określonym uszkodzonym układem a układem dobrym w określonym momencie. Osiągnięte jest to poprzez powiązanie z każdym węzłem w układzie stanu układu dobrego w tym węźle oraz listy efektów uszkodzeń (ang. list of fault effects) (stanów układów zawierających uszkodzenie) dla tych układów, dla których stany te różnią się od stanu układu dobrego w danym węźle. Ta lista jest nazywana listą efektów uszkodzeń.

Symulacja równoczesna opiera się na symulacji sterowanej zdarzeniami, gdzie zmiana wartości logicznej węzła (w dobrym lub uszkodzonym układzie) tworzy zdarzenie i powoduje, że dany węzeł jest umieszczany w "kolejce zdarzeń". Postępowanie procesu symulacji dokonuje się w dyskretnych odcinkach czasu poprzez obsłużenie wszystkich zdarzeń w danym odcinku czasu, a następnie zwiększenia "zegara" symulacji. Symulacja rozpoczyna się podaniem wektora na pierwotne wejścia układu, co powoduje,że pewien podzbiór węzłów zostaje umieszczony w kolejce zdarzeń. Gdy węzeł jest usuwany z kolejki zdarzeń, następuje przetwarzanie jak poniżej:

1. Jeżeli zdarzenie wynikło ze zmiany stanu węzła w dobrym układzie, wówczas wszystkie elementy mające ten węzeł jako wejście są ponownie obliczane.
2. Zdarzenie powodowane przez układ z uszkodzeniem jest obsługiwane podobnie, lecz stan danego węzła jest brany z listy efektów uszkodzeń.
3. W czasie obliczania stanu elementu, który został uaktywniony przez dobre zdarzenie, każdy efekt uszkodzenia jest propagowany do wyjścia, o ile tylko uszkodzenie powoduje, że stan wyścia różni się od jego stanu dla poprawnego układu.
4. Jeżeli stan danego węzła w uszkodzonym układzie staje się identyczny ze stanem węzła w poprawnym układzie, wówczas odpowiednie efekty uszkodzeń są usuwane z listy efektów uszkodzeń dla tego węzła.

Zaletą symulacji równoczesnej jest jej szybkość wynikająca z symulacji tylko tych uszkodzeń, które wywołują efekt w układzie. Jednakże gdy ilość aktywnych uszkodzeń w układzie jest stosunkowo duża, wówczas szybkość zmniejsza się z powodu dodatkowych czynności obliczeniowych wynikających z konieczności obsługi listy efektów uszkodzeń. Innym problemem przy stosowaniu tej metody jest niemożliwość przewidzenia zapotrzebowania na pamięć.

C. Różnicowa i równoległo-różnicowa symulacja uszkodzeń (ang. Differential and Paralel Differential Fault Simulation). W tej metodzie różnice pomiędzy uszkodzonym i sprawnym układem są zapamiętywane jedynie w postaci stanów linii (przerzutników i linii sprzężenia zwrotnego). W ten sposób zapotrzebowanie na pamięć w symulacji różnicowej jest znacznie mniejsze niż w technice równoczesnej. Algorytm PROOFS, przedstawiony w [9], jest ulepszeniem różnicowej metody symulacji. Algorytm PROOFS opiera się na wykonaniu symulacji logicznej dla określonego wektora testowego, aby otrzymać stany dla poprawnego układu. W czasie symulacji logicznej uszkodzenia, które powodują różnice stanów w miejscach umieszczenia tych uszkodzeń, są zaznaczane (ang. flaged) jako aktywne. Uszkodzenia są uznawane za nieaktywne wtedy i tylko wtedy, gdy nie powodują żadnych różnic w miejscu ich umieszczenia ani na żadnej z linii stanów. Po wykonaniu symulacji logicznej uszkodzenia zaznaczone jako aktywne są wprowadzane do układu jako grupa składająca się z w uszkodzeń (zwykle w jest równe długości słowa komputera, który jest wykorzystywany do symulacji) i wykonywana jest

symulacja równoległa, aby obliczyć stany na wyjściach uszkodzonych węzłów. Dla każdego uszkodzenia stany węzłów (przerzutniki i linie sprzężeń), które są różne od stanów w poprawnym układzie, są zapamiętywane w postaci list uszkodzeń. Taka lista uszkodzeń jest związana z określonym uszkodzeniem i jest ponownie używana, gdy to samo uszkodzenie jest później wprowadzane do układu. Proces wprowadzania w uszkodzeń, przeprowadzania symulacji równoległej i modyfikacji list uszkodzeń jest powtarzany tak długo, jak długo nie zostaną wyczerpane wszystkie aktywne uszkodzenia. Należy zauważyć, że wstępnym krokiem dla wszystkich błędów jest ich uporządkowanie i to uporządkowanie jest używane do określania kolejności pobierania aktywnych uszkodzeń.

Zaletą algorytmu PROOFS jest jego szybkość i niewielkie wymagania w stosunku do pamięci, gdyż układy są niewrażliwe na opóźnienia, czyli są to układy synchroniczne i niektóre klasy układów asynchronicznych. Jednakże dla ogólnych układów z opóźnieniami algorytm ten jest podobny do prostej symulacji szeregowej (ang. serial fault simulation).

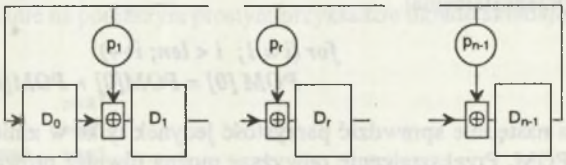
Istnieje możliwość połączenia symulacji równoległej i równoczesnej, opierając się na różnych metodach podziału uszkodzeń.

3. STRUKTURA I DZIAŁANIE PROGRAMU

Moduł generatora wektorów testowych TPG

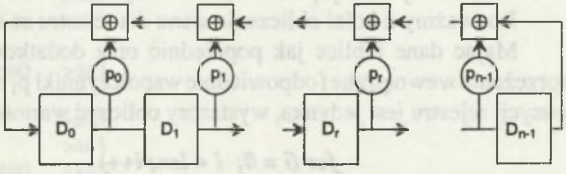
Generatory wykorzystujące w swej strukturze rejestry LFSR mogą być zbudowane jako: układy z wewnętrznym sprzężeniem zwrotnym (ang. Internal Exclusive OR type linear feedback path - IE) - rys.1, układy z zewnętrznym sprzężeniem zwrotnym (ang. External Exclusive OR type linear feedback path - EE) - rys. 2, układy o sprzężeniu mieszanym tzw. wewnętrzno-zewnętrznym lub zewnętrzno-wewnętrznym.

Dodatkowo w strukturze mogą być wykorzystane oprócz przerzutników D również przerzutniki T; zalety takiej struktury opisane są w [7]. Przykładowa struktura takiego generatora wykorzystującego przerzutniki T oraz D i sprzężenie typu IE przedstawiona jest na rys. 3.



Rys. 1. Generator LFSR ze sprzężeniami typu IE
Fig. 1. IE LFSR test pattern generator

Stworzony moduł generatora może być skonfigurowany jako rejestr ze sprzężeniami typu EE, IE oraz mieszanymi typu Top Bottom (TB), zbudowany z przerzutników D lub T.



Rys. 2. Generator LFSR ze sprzężeniami typu EE
Fig. 2. EE LFSR test pattern generator

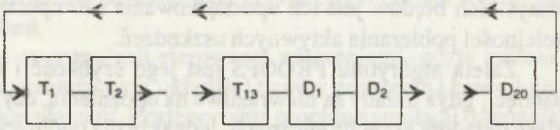
Rozpatrzmy najpierw sposób opisu i obliczania stanu rejestru EE w programie w języku C. Niech będą dane tablice:

longint GEN [len] /* opisująca stan wewnętrzny przerzutników rejestru, typ longint ma taką ilość bitów, jak długie jest słowo procesora (32b dla procesora i386 lub i486); liczba elementów tablicy (len) zależy od maksymalnej długości rejestru, który chcemy symulować, np. dla len = 4 maksymalna długość rejestru wynosi 4 *32 = 128 bitów.

longint EE [len] /* opisująca sprzężenia zewnętrzne rejestru. Jedynka na danej pozycji oznacza, że istnieje sprzężenie z wyjścia danego przerzutnika (odpowiednie współczynniki p_i na rys. 2).

longint POM [len] /* tablica pomocnicza do obliczeń.

W danej chwili w tablicy GEN znajduje się wartość opisująca stan generatora.



Rys. 3. Przykład generatora testów wykorzystującego przerzutniki T oraz D
Fig. 3. An example of test pattern generator using D and T flip_flop

Aby otrzymać kolejny stan dla rejestru wykorzystującego tylko sprzężenia typu EE, należy obliczyć wartość na wejściu rejestru (punkt A rys.1), a następnie przesunąć zawartość rejestru w prawo. Do obliczenia wartości wykorzystywane jest następujące wyrażenie (zapis w języku C):

```
for (i = 0; i < len; i++)
    POM [i] = GEN[i] & EE[i];
```

Po wykonaniu powyższej operacji tablica POM zawiera pewną liczbę jedynek. Z właściwości funkcji Exclusive OR wiadomo, że ma ona wartość 1, gdy ilość jedynek na jej wejściach jest nieparzysta. Jednak bezpośrednie zliczanie liczby jedynek byłoby nieefektywne, a nie istnieje prosta metoda wykorzystania wskaźnika Parity procesora ze względu na przyjęte założenia. Jednak wykorzystując własności liniowości funkcji Exclusive OR można dokonać następującego przekształcenia:

```
for (i = 1; i < len; i++)
    POM [0] = POM[0] ∧ POM[i];
```

a następnie sprawdzić parzystość jedynek tylko w zmiennej POM[0] zamiast w całej tablicy POM. Przekształcenie powyższe można również prowadzić dalej i uzyskać jeszcze mniejszą ilość bitów do zliczania. Następnie tak obliczona wartość wejściowa jest wsuwana na pierwszą pozycję rejestru, a pozostałe są przesuwane w prawo, po wykonaniu powyższych operacji obliczony zostaje kolejny stan rejestru ze sprzężeniami typu EE.

Rozważmy z kolei obliczenie stanu dla rejestru ze sprzężeniami typu IE.

Mając dane tablice jak poprzednio oraz dodatkowo tablicę ***longint IE [len]*** opisującą sprzężenia wewnętrzne (odpowiednie współczynniki p_i na rys. 1), w przypadku gdy na ostatniej pozycji rejestru jest jedynka, wystarczy obliczyć wartość wyrażenia pomocniczego:

```
for (i = 0; i < len; i++)
    POM [i] = GEN[i] ∧ IE[i];
```

a następnie dokonać przesunięcia zawartości rejestru (Tablicy GEN) i obliczenia:

```
for (i = 0; i < len; i++)
  GEN [i] = GEN[i] ∧ POM[i];
```

w wyniku czego otrzymuje się stan następny. Jeżeli na ostatniej pozycji rejestru jest zero, wystarczy dokonać jedynie przesunięcia zawartości rejestru. Jeżeli wykorzystamy dodatkowo wartość obliczoną poprzednio dla rejestru ze sprzężeniami typu EE, to możemy również obliczyć stan dla rejestru o sprzężeniach mieszanych typu TB.

Dysponując dodatkowo tablicą T[len] opisującą, które z przerzutników rejestru są przerzutnikami typu T oraz dokonując obliczeń podobnych do opisanych powyżej, można otrzymać stan następny dla generatora zbudowanego na bazie przerzutników typu D i T.

Należy zauważyć, że większość operacji wykonywanych wielokrotnie wewnątrz pętli to pojedyncze instrukcje procesora, co więcej krotność wykonywania tych operacji jest niewielka, jeżeli używamy komputera o dostatecznie długim słowie maszynowym. Sprawia to, że przedstawione metody są efektywne czasowo.

4. MODUŁ SYMULATORA UKŁADÓW ISCAS85

Format listy połączeń ISCAS85

Format listy połączeń ISCAS85 nie był nigdy formalnie opisany. Pierwotnie [1] był rozpowszechniany z translatorem napisanym w języku FORTRAN, który umożliwiał tłumaczenie na kilka innych formalnie udokumentowanych formatów list połączeń. Zbiór w formacie ISCAS85 zawiera dane o strukturze układu, jak również o możliwych do wystąpienia błędach. Część danych jest podana nadmiarowo, tzn. można by je uzyskać z pozostałych danych zawartych w tym zbiorze. Jednak ta nadmiarowość sprawia, że bardzo łatwo przekształcić te dane do formatu dogodnego do symulacji.

Format ISCAS85 objaśniony zostanie na poniższym prostym przykładzie układu składającego się z 6 bramek NAND.

```
    1     1 gat  inpt  1   0              >sa1
    2     2 gat  inpt  1   0              >sa1
    3     3 gat  inpt  2   0  >sa0        >sa1
    8     8 fan  from    3 gat            >sa1
    9     9 fan  from    3 gat            >sa1
    6     6 gat  inpt  1   0              >sa1
    7     7 gat  inpt  1   0              >sa1
   10    10 gat  nand  1   2              >sa1
        1    8
   11    11 gat  nand  2   2  >sa0        >sa1
        9    6
   14    14 fan  from  11 gat            >sa1
   15    15 fan  from  11 gat            >sa1
   16    16 gat  nand  2   2  >sa0        >sa1
```

Pierwsza linia zawiera następujące pola:

Pierwsze pole w linii:	adres węzła	1
Drugie pole w linii:	nazwa węzła	1gat
Trzecie pole w linii:	typ	inpt
Czwarte pole w linii:	obc.wyjścia	1
Piąte pole w linii:	ilość wejść	0
Ostatnie pole w linii:	uszkodzenie(a)	>sa1

Znaczenia poszczególnych pól są następujące:

adres węzła – unikalna liczba, która odróżnia węzeł od wszystkich pozostałych w układzie;
nazwa węzła – dowolny napis dostarczający dodatkową informację dla czytającego dotyczącą funkcji węzła;
typ – funkcja realizowana przez bramkę sterującą tym węzłem.

Dopuszczalne są następujące typy:

inpt pierwotne wejście układu;
and nand or nor xor xnor odpowiednie bramki logiczne;
buff nie odwracający wzmacniacz (bufor);
from rozdzielenie wyjścia na kolejne wejście;

obc.wyjścia – liczba wejść bramek sterowanych przez dany węzeł (0) w przypadku wyjść pierwotnych układu;
ilość wejść – liczba wejść bramki (0) w przypadku wejść pierwotnych;
uszkodzenie(a) – uszkodzenia typu sklejenie węzła z zerem (ang Stuck at 0) Sa0 lub sklejenie z 1 Sa1.

Taki typ linii jak opisany powyżej, zwany linią węzła dostarcza podstawowych informacji o każdym węźle w układzie. Są również dwa inne typy linii,które mogą być powiązane z linią pierwszego rodzaju. Pierwszym z nich jest linia wejść. Linia ta dostarcza listę adresów węzłów, które sterują danym węzłem. Linia taka zawsze pojawia się bezpośrednio po linii węzła, z którą jest powiązana. Po liniach opisujących węzły, które są wejściami pierwotnymi, nie występują linie wejść, gdyż wejście pierwotne nie jest sterowane przez żaden z wewnętrznych węzłów układu. Liczba adresów występujących w linii wejść musi być zgodna z liczbą wejść podaną w linii węzła.

Przykładowo w węźle o adresie 10 ilość wejść wynosi 2 i odpowiednio w kolejnej linii podane są dwa adresy 1 i 8. Węzeł o adresie 1 jest wejściem pierwotnym.

Węzeł o adresie 8 jest przykładem linii trzeciego rodzaju, linii rozgałęzienia wyjścia. Linia ta podobnie jak linia węzła posiada adres, nazwę oraz typ (zawsze jest to typ *from*). Linia rozgałęzienia wyjścia musi pojawić się bezpośrednio po linii węzła, którego jest rozgałęzieniem. Ponieważ węzeł rozgałęzienia ma zawsze jedno wejście i jego wyjście przyłączone jest do jednego wejścia, informacje te nie są podawane, zamiast nich podana jest nazwa węzła sterującego. Informacja ta jest informacją nadmiarową, niemniej jednak stanowi ona część formatu ISCAS85. Z tego, że węzeł rozgałęzienia from ma obciążenie wyjścia równe 1, wynika, że nie

może on być pierwotnym wyjściem układu, gdyż wyjście pierwotne z definicji ma obciążenie równe 0.

Przykładowo węzeł o adresie 16, mający nazwę 16gat, jest bramką typu NAND, steruje dwoma wejściami i sam ma 2 wejścia. Bezpośrednio za tą linią następuje linia wejść, która podaje adresy węzłów 2 i 14, z których to są sterowane wejścia węzła 16 (węzeł 2 jest wejściem pierwotnym układu, a węzeł 14 jest węzłem rozgałęzienia z innej bramki NAND). Za linią wejść następują dwie linie rozgałęzienia, dalej następują kolejne węzły układu.

Moduł symulatora pełni kilka funkcji:

1. Wczytania pliku w formacie ISCAS85 i przetworzenie go w pewien format pośredni dogodny do symulacji.
2. Przyporządkowanie stanów generatora wektorów testowych do odpowiednich wejść pierwotnych układu symulowanego.
3. Właściwa symulacja układu.

4a. Przyporządkowanie wyjść do odpowiedniego układu kompresji.

4b. Porównanie odpowiedzi układu poprawnego z odpowiedziami błędnymi i przeprowadzenie odpowiednich obliczeń.

Aby przyspieszyć symulację, wykorzystano metodę symulacji równoległej dla pełnej długości słowa procesora, tzn. operacje logiczne są wykonywane na całym słowie procesora (32b), a każdy bit odpowiada układowi z innym uszkodzeniem. Zastosowanie tej metody w połączeniu z odpowiednim doborem struktur danych do symulacji umożliwiło osiągnięcie znacznej szybkości symulacji. Ponieważ układy ISCAS85 są wyłącznie układami kombinacyjnymi, sam proces symulacji jest dość prosty. Jedynym problemem było to, że w ogólnym przypadku nie znana jest kolejność ani hierarchia bramek w układzie, dlatego dla obliczenia stanu w dowolnym punkcie zastosowano metodę iteracyjną. Pozwala to obliczyć stan dowolnego punktu układu zaczynając obliczenia od tego punktu i w miarę potrzeby postępując wstecz. Układy ISCAS85 z założenia są kombinacyjne, jednak aby w przypadku błędu w pliku wejściowym nie doprowadzić do "zapętlenia się programu", podczas kolejnych iteracji jest sprawdzane, czy w strukturze układu nie powstała pętla sprzężenia zwrotnego, która prowadziłaby do powstania układu sekwencyjnego.

Użytkowanie programu

Jak już wspomniano, ze względu na punkt 4 założeń program nie został wyposażony w mechanizmy środowiska przyjaznego dla użytkownika. Wszystkie dane niezbędne do symulacji należy przed rozpoczęciem symulacji umieścić w kilku plikach. Są to pliki typu ASCII, można je więc tworzyć i modyfikować za pomocą dowolnego edytora tekstów. Plik konfiguracji zawiera informacje niezbędne do przeprowadzenia symulacji. Są to nazwa pliku opisującego układ symulowany w formacie ISCAS85, nazwa pliku, w którym należy umieścić wyniki symulacji, struktura, i wartości początkowe generatora testów.

Ponieważ proces symulacji przy dużej ilości powtórzeń (a tak jest w przypadku badań pokrycia uszkodzeń) jest procesem czasochłonnym, wyniki częściowe są zapisywane w plikach wyjściowych.

W wyniku działania programu powstaje plik wyjściowy zawierający wyniki symulacji. Jest to plik typu ASCII, a więc można go również przetwarzać za pomocą edytora tekstów podobnie jak pliki konfiguracyjne. Plik zawiera tyle linii, ile wektorów testowych zostało zasymulowanych. W każdej linii znajduje się liczba określająca, ile uszkodzeń pozostało jeszcze nie pokrytych przez zasymulowane już wektory testowe. Pierwsza linia zawiera więc całkowitą ilość symulowanych uszkodzeń. Ilość ta jest określona przez dane zawarte w pliku ISCAS85 opisującym symulowany układ cyfrowy. Taki format danych wyjściowych pozwala łatwo wykorzystać je do wykreślenia krzywej pokrycia błędów. W celu uzyskania tego wykresu stworzono program umożliwiający wykreślenie go. Do wykreślenia wykresu można również wykorzystać jeden z ogólnie dostępnych do tego celu programów.

5. PRZYKŁADOWE WYNIKI

Poniżej przedstawiono przykładowe wyniki symulacji przeprowadzonych za pomocą stworzonego symulatora. Te wstępne badania zostały przeprowadzone przede wszystkim w celu weryfikacji poprawności działania symulatora i oceny jego możliwości, a zwłaszcza szybkości działania.

Zestawione poniżej wyniki i czasy symulacji zostały uzyskane przy wykorzystaniu komputera klasy PC 386DX 33 MHz, a więc powszechnie dostępnego. Próby przeniesienia symulatora na maszynę typu workstation nie zostały jeszcze podjęte.

Tabela 1 zawiera zestawione czasy symulacji kilku przykładowych układów ze zbioru ISCAS85.

C432 - jest to dekoder priorytetowy składający się z 153 bramek logicznych, posiada on 432 połączenia wewnętrzne, 36 wejść pierwotnych i 7 wyjść. Układ był pobudzany rejestrem ze sprzężeniem liniowym, o długości 36 bitów (tyle ile pierwotnych wejść układu) opisanym wielomianem pierwotnym, o strukturze D25T11.

C1908 - jest to układ składający się z 855 bramek logicznych, 1908 połączeń wewnętrznych, posiada 33 wejścia pierwotne i 25 wyjść. Układ był pobudzany z wyjść rejestru ze sprzężeniem IE, o długości 33 bitów opisanym wielomianem pierwotnym. Krzywą pokrycia dla tego układu przedstawia rys. 4.

C6288 - jest to układ mnożący 16 * 16 bitów, zawierający 2384 bramki logiczne, 6288 połączeń wewnętrznych, układ ten posiada 32 wejścia i 32 wyjścia. Układ był pobudzany z wyjść rejestru ze sprzężeniem liniowym o długości 32 bitów.

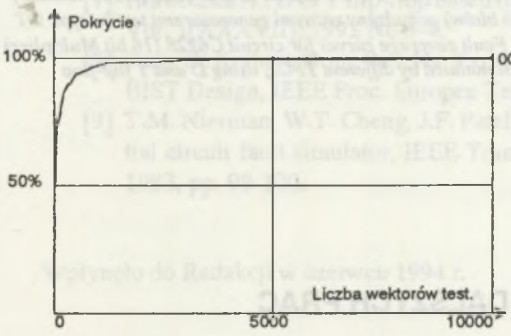
Do symulacji pokrycia uszkodzeń przyjęto następującą metodologię: w pierwszym kroku (pierwszy wektor uzyskany z generatora) symulowane są wszystkie uszkodzenia układu zgodnie z opisem w formacie ISCAS85. Uszkodzenia, które zostały wykryte w danym kroku, są zapamiętywane i w kolejnym kroku (dla kolejnego wektora testowego) symulowane są już tylko te uszkodzenia, które nie zostały wykryte w poprzednich krokach. W tym przypadku symulacja równoległa jest wykorzystywana w ten sposób, że każdy bit słowa procesora symuluje inne

Tabela 1

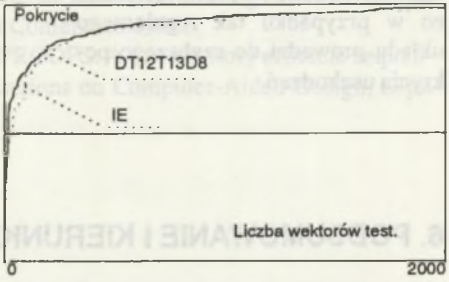
Rodzaj układu	Liczba bramek	Liczba sym. uszkodzeń	Liczba wektorów test.	Czas symulacji
C432	153	524	10000	7' 35"
C432	153	524	20000	14' 30"
C1908	855	1879	1000	14' 12"
C1908	855	1879	2000	25' 02"
C1908	855	1879	5000	34' 14"
C1908	855	1879	1000	48' 15"
C1908	855	1879	20000	1h 15' 30"
C6288	2384	7744	1000	34' 00"
C6288	2384	7744	5000	1h 20' 00"
C6288	2384	7744	10000	2h 20' 00"

pojedyncze uszkodzenie układu pobudzonego tym samym wektorem testowym. Po czym sprawdza się, czy odpowiedź układu z pojedynczym błędem różni się od odpowiedzi układu poprawnego. Symulowanie w każdym kroku tylko tych uszkodzeń, które nie zostały wykryte w poprzednich krokach, przyczynia się dodatkowo do znacznego skrócenia czasu symulacji.

Tak znaczną szybkość symulacji tłumaczy typowa krzywa pokrycia błędów pokazana na rys. 4. Jak widać, pierwszych 500 - 1000 wektorów z reguły powoduje pokrycie ponad 90% uszkodzeń, tak więc w pozostałych krokach symulacji symuluje się pozostałe 10% uszkodzeń. Krzywa dąży do poziomu 100% bardzo powoli, więc wymagane jest wygenerowanie dużej ilości testów, ale liczba symulowanych uszkodzeń jest coraz mniejsza. Przykładowe krzywe pokrycia



Rys. 4. Przykładowa krzywa pokrycia błędów dla układu C1908
Fig. 4. Typical fault coverage curve for C1908 circuit



Rys. 5. Krzywe pokrycia układu C1908 dla dwóch różnych generatorów wektorów testowych opisanych tym samym wielomianem pierwotnym: $p(x) = 1 + x^{20} + x^{21} + x^{24} + x^{25} + x^{28} + x^{29} + x^{32} + x^{33}$
Fig. 5. Fault coverage curves for C1908 circuit stimulated by two different TPG, described by the same primitive polynomial p(x)

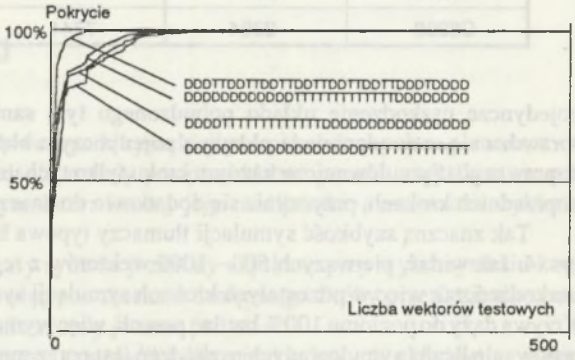
błędów przedstawione na rys. 5 uzasadniają celowość stosowania metod symulacyjnych do oceny skuteczności działania generatorów testów. Wszystkie krzywe zostały uzyskane w wyniku symulacji pokrycia uszkodzeń w układzie C1908 opisanym poprzednio. Do pobudzania wejść użyto natomiast różnych generatorów wektorów testowych. Każdy z tych generatorów opisany był tym samym wielomianem pierwotnym, ale posiadał różną realizację.

Był to wielomian:

$$p(x) = 1 + x^{20} + x^{21} + x^{24} + x^{25} + x^{28} + x^{29} + x^{32} + x^{33}$$

Pierwszy z generatorów (linia przerywana) był typowym generatorem o sprzężeniach wewnętrznych IE, drugi (linia ciągła) był zrealizowany za pomocą przerzutników T oraz D. Konfiguracja tego generatora jest opisana skrótowo D12T13D8. Jak widać, w tym konkretnym przypadku drugi z generatorów daje znacząco lepsze pokrycie uszkodzeń w testowanym układzie.

Rys. 6 przedstawia krzywe pokrycia uszkodzeń dla układu ISCAS 6288 pobudzanego generatorami zbudowanymi z przerzutników D oraz T o różnej konfiguracji. Nie trudno zauważyć, że zmiana konfiguracji ma znaczny wpływ na szybkość pokrycia uszkodzeń w pierwszym okresie. Zjawisko to można by wytłumaczyć, wiedząc, że układ ISCAS 6288 jest układem mnożącym 16*16 bitów, a więc układem bardzo regularnym. Umieszczenie w rejestrze generatora testów na przemian przerzutników D i T powoduje częstsze zmiany na jego wyjściach, co w przypadku tak regularnego układu prowadzi do szybszego pokrycia uszkodzeń.



Rys. 6. Krzywa pokrycia dla układu ISCAS 6228 (układ mnożący 16×16 bitów) pobudzany różnymi generatorami testów typu DT
Fig. 6. Fault coverage curves for circuit C6228 (16 bit Multiplier) stimulated by different TPGs, using D and T flip-flop

6. PODSUMOWANIE I KIERUNKI DALSZYCH PRAC

Przedstawione powyżej przykłady potwierdzają celowość użycia metody symulacji do oceny skuteczności testowania za pomocą generatorów wykorzystujących sprzężenia liniowe. Uzyskane czasy symulacji dają nadzieję na wykorzystanie stworzonego symulatora do symulacji dość złożonych układów cyfrowych nawet przy użyciu komputera klasy PC.

W ramach dalszych prac należałoby rozszerzyć możliwości symulatora o symulacje różnych struktur kompresorów (jednowejściowych, wielowejściowych MISR) oraz różnych sposobów połączenia generatora i kompresora z układem testowanym, np. technika scan path circullar path itp. Należałoby również przeprowadzić porównanie szybkości działania symulatora na komputerze o większej mocy obliczeniowej i ewentualnie wprowadzić zmiany optymalizujące czas wykonania programu.

Stworzony program symulatora jest jedynie narzędziem, należałoby więc opracować metodologię użycia tego narzędzia, np. do wyboru optymalnych struktur generatorów i kompresorów do określonych zastosowań.

LITERATURA

[1] Brglez F., Fujiwara H.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran, Proc. IEEE Int. Symposium on Circuits and Systems; Special Session on ATPG and Fault Simulation, June 1985.

[2] Dihraj K. P.: Fault-tolerant Computing, Theory and Techniques, Prentice-Hall 1986.

[3] Daniel G. S.: Parallel-Concurrent Fault Simulation, IEEE Transactions on VLSI vol 1 No.3 SEPT 1993.

[4] Xavier D, A. Aitken A. D., Ivanov A.: V.K. Agraval: Experiments on Aliasing in Signature Analysis Registers, Proc. International Test Conference 1989.

[5] Lambidoniz D., Ivanov A., Agraval V. K.: Fast Signature Computation for Linear Compactors. Proc. International Test Conference 1991.

[6] Kewal K. Saluja, Chin-Foo: See An Efficent Signature Computation Method, IEEE Design & Test of Computers, December 1992.

[7] Hławiczka A.: D or T flip-flop based linear registers, Archives of Control Sciences, Vol. 1(XXXVII), 1992 No 3-4.

[8] Rajiv D. Kothari, Dong Sam Ha: Experimental Results on aliasing Errors in circular BIST Design, IEEE Proc. Europen Test Conference 1993.

[9] T.M. Nierman, W.T. Cheng, J.F. Patel: PROOFS: A fast memory efficient sequential circuit fault simulator, IEEE Transactions on Computer-Aided Design, Sept. 1983, pp. 99-100.

Wpłynęło do Redakcji w czerwcu 1994 r.

Abstract

In the evaluation of fault coverage in digital circuits, analysis based on independent or asymetric error models is very often used. However this approach not always leads to correct results. An alternative to this is using a simulation method for the evaluation of fault coverage.

The simulation method requires some exemplary circuits that make possible to compare obtained results. A set of such exemplary circuits is ISCAS which is briefly described in this article. Requirements which should be met by the simulator used are discussed. Because these requirements are different from general purpose simulator requirements, the special simulator for evaluating fault coverage has been designed. This simulator has been used for evaluating fault coverage in circuits stimulated by LFSR registers with various structures. Particular attention was given to LFSR based on D and T flip-flops. A brief description of a created simulator is given. In the article a review of simulation algorithms is given too. Some examples are given which show that some structures of LFSR described by the same polynomial are better, i.e. less test vectors give better fault coverage.