

INSTYTUT CYBERNETYKI TECHNICZNEJ
POLITECHNIKI WROCŁAWSKIEJ
Raport serii: PREPRINTY nr 27/2003

Synteza sekwencyjnych
układów samosprawdzalnych
z użyciem kodów
nieuporządkowanych
(rozprawa doktorska)

Jerzy W. Greblicki

PROMOTOR:

dr hab. inż. Stanisław J. Piestrak, prof. PWr

Słowa kluczowe:

- synteza układów cyfrowych
- układ sekwencyjny
- układ samosprawdzalny
- komputerowo wspomagane
projektowanie układów cyfrowych
- tolerowanie uszkodzeń w systemach cyfrowych
- testowanie układów cyfrowych

Wrocław
maj 2003

Podziękowania

Składam serdeczne podziękowania prof. Stanisławowi J. Piestrakowi za opiekę naukową i pomoc okazaną w czasie mojej pracy nad rozprawą doktorską. Słowa podziękowania należą się również prof. Januszowi Biernatowi, Kierownikowi Zakładu Architektury Komputerów, z którym mam przyjemność pracować. Dziękuję także Koleżankom i Kolegom z Zakładu Architektury Komputerów i Instytutu Cybernetyki Technicznej za liczne dyskusje i wszelką okazaną mi pomoc.

Spis treści

Spis rysunków	4
Spis tablic	5
Wykaz ważniejszych oznaczeń	6
Wykaz ważniejszych skrótów	7
1 Wprowadzenie	8
1.1 Motywacja	8
1.2 Samosprawdzalne układy sekwencyjne	9
1.3 Przegląd metod syntezy TSC układów sekwencyjnych	9
1.3.1 TSC układy sekwencyjne wykorzystujące EDC	10
1.3.2 TSC SM wykorzystujące układy kontrolujące	12
1.4 CAD	13
1.5 Mankamenty znanych metod syntezy TSC SM	14
1.6 Cel i teza pracy	17
2 Pojęcia podstawowe	19
2.1 Modele układów sekwencyjnych	19
2.2 Uszkodzenia i błędy w układach cyfrowych	20
2.3 Równoważność i dominacja uszkodzeń	21
2.4 Kody nieuporządkowane wykrywające błędy	22
2.5 Własności układów wykorzystujących kody nieuporządkowane	27
2.6 Układy samosprawdzalne	29
2.6.1 Podstawowe definicje	29
2.6.2 Założenia dotyczące układów samosprawdzalnych	32
2.6.3 Podstawowe struktury systemów samosprawdzalnych	32
3 Synteza TSC układów kombinacyjnych	35
3.1 Synteza TSC/CD układów kombinacyjnych	35
3.1.1 Weryfikacja kodowej rozłączności (CD)	35
3.1.2 Kombinacyjne układy ST o strukturze 2-poziomowej AND-OR	40
3.1.3 Kombinacyjne układy ST o strukturze 3-poziomowej AND-AND-OR	43

3.1.4	Podsumowanie	52
3.2	Synteza układów kombinacyjnych TSC/STC	53
3.2.1	Minimalizacja funkcji układu kombinacyjnego	53
3.2.2	Algorytm minimalizacji funkcji pozytywnie gładkich	53
3.2.3	Weryfikacja własności ST zminimalizowanych układów	55
3.2.4	Podsumowanie	56
4	Automatyczna synteza TSC układów sekwencyjnych	57
4.1	Analiza specyfikacji układów	58
4.2	Stany obojętne	58
4.2.1	Wejściowe stany obojętne	58
4.2.2	Wyjściowe stany obojętne	59
4.3	Gwarancja istnienia STC dla wejść i stanów wewnętrznych poprzez dobór kodów	60
4.3.1	Kodowanie wejść	60
4.3.2	Kodowanie stanów wewnętrznych	60
4.4	Algorytm kodowania wyjść gwarantujący istnienie STC	61
4.5	System automatycznej syntezy TSC układów sekwencyjnych Mealy'ego	65
4.6	Przykład syntezy TSC układu sekwencyjnego	69
5	Wyniki syntezy sekwencyjnych układów wzorcowych	74
5.1	Metoda badań	74
5.2	Analiza struktury układów i minimalizacja liczby stanów wewnętrznych	76
5.3	Kodowanie wejść oraz stanów wewnętrznych	76
5.4	Konstrukcja kodu wyjściowego oraz STC	76
5.5	Układy o strukturze TSC/CD	79
5.6	Układy o strukturze TSC/STC	81
5.7	Podsumowanie	87
6	Zakończenie	88
	Otwarte problemy badawcze	90
	Bibliografia	92

Spis rysunków

2.1	Schemat układu Mealy'ego	20
2.2	Fragment układu AND-OR z uszkodzeniami.	22
2.3	Kombinacyjny układ TSC: Struktura I TSC/STC	33
2.4	Kombinacyjny układ TSC: Struktura II TSC/CD	33
2.5	Schemat SM TSC/CD	34
2.6	Schemat SM TSC/STC	34
3.1	Schemat układu H realizującego funkcję przejść z przykładu 3.2	44
3.2	Schemat układu H z przykładowym uszkodzeniem wewnętrznym linii x_1	45
3.3	Fragment układu o strukturze AND-AND-OR	47
3.4	Schemat ST układu H o strukturze AND-AND-OR z przykładu 3.4.	51
4.1	Schemat syntezy TSC SM	66
4.2	Schemat blokowy systemu TSCSM/CAD syntezy TSC SM	67
4.3	Opis układu mc w formacie KISS	70

Spis tablic

2.1	Charakterystyka STC dla wykorzystywanych kodów $2/n$.	25
2.2	Charakterystyka STC dla wykorzystywanych kodów $1/n$.	26
2.3	Charakterystyka STC wykorzystywanych kodów Bergera.	26
4.1	Lista słów grupy G_p w układzie kirkman uwzględniająca stany obojętne	65
4.2	Lista słów grupy G_p w układzie kirkman (stany obojętne równe 0)	65
4.3	Kompletny zbiór słów kodu wyjściowego dla układu kirkman	66
4.4	Lista modułów pakietu oprogramowania TSC/SM CAD	69
4.5	Tablica przejść układu mc	71
4.6	Tablica przejść układu mc	72
4.7	Kod wejściowy układu mc	72
4.8	Kodowanie stanów wewnętrznych układu mc	73
4.9	Kod wyjściowy układu mc	73
5.1	Charakterystyka układów wzorcowych ISCAS'89	75
5.2	Zestawienie kodów zastosowanych do kodowania wejść	77
5.3	Zestawienie kodów zastosowanych do kodowania stanów wewnętrznych	78
5.4	Zestawienie kodów wyjściowych dla układów wzorcowych	79
5.5	Wyniki analizy kodowej rozłączności dla TSC/CD SM	81
5.6	Liczba nietestowalnych uszkodzeń w układach TSC/CD SM	82
5.7	Złożoności układów TSC/CD	83
5.8	Liczba nietestowalnych uszkodzeń w układach TSC/ES	84
5.9	Złożoności układów TSC/ES	85
5.10	Złożoności układów TSC/STC	86

Wykaz ważniejszych oznaczeń

C_{IN}^{e-} – zbiór wszystkich wejściowych słów niekodowych, które powstają na skutek pojedynczych błędów $1 \rightarrow 0$

C_{IN}^{e+} – zbiór wszystkich wejściowych słów niekodowych, które powstają na skutek pojedynczych błędów $0 \rightarrow 1$

C_{IN}^e – zbiór wszystkich wejściowych słów niekodowych powstałych na skutek wystąpienia błędów pojedynczych

C_{IN} – zbiór słów kodu wejściowego

C_{OUT} – zbiór słów kodu wyjściowego

C_{STATE} – zbiór słów kodu stanu

f_0 – uszkodzenie $s/0$

f_1 – uszkodzenie $s/1$

F – zbiór pojedynczych uszkodzeń typu $s/1$ oraz $s/0$

$\mathbf{H}$ – układ kombinacyjny

M^{e-} – zbiór wszystkich implikantów, które odpowiadają wszystkim niekodowym słowom $X_i^{e-} \in$

C_{IN}^{e-}

M^e – zbiór wszystkich implikantów, które odpowiadają wszystkim niekodowym słowom $X_i^e \in C_{IN}^e$

M^{e+} – zbiór wszystkich implikantów, które odpowiadają wszystkim niekodowym słowom $X_i^{e+} \in$

C_{IN}^{e+}

m_i – implikant odpowiadający X_i , $m_i \leftrightarrow X_i$

m_i^e – implikant odpowiadający X_i^e , $m_i^e \leftrightarrow X_i^e$

$m_i(x^*)$ – zbiór wszystkich zmiennych x należących do implikantu m_i

$m^*(p_k)$ – zbiór wszystkich implikantów m zawierających podimplikant p_k

M – zbiór implikantów odpowiadających słowom ze zbioru C_{IN}

X_i^{e+} – wejściowe słowo niekodowe, które powstało na skutek pojedynczego błędu $0 \rightarrow 1$ w X_i

X_i^{e-} – wejściowe słowo niekodowe, które powstało na skutek pojedynczego błędu $1 \rightarrow 0$ w X_i

X_i^e – wejściowe słowo niekodowe, które powstało na skutek pojedynczego błędu w X_i

$X = (x_{n-1}, \dots, x_0)$ – wektor wejściowy

$Y = (y_{n-1}, \dots, y_0)$ – wektor słowa stanu

$Z = (z_{n-1}, \dots, z_0)$ – wektor wyjściowy

$z^*(m_i)$ – zbiór wszystkich zmiennych wyjściowych z zawierających implikant m_i

$z^*(p_k)$ – zbiór wszystkich zmiennych wyjściowych z zawierających podimplikant p_k

Wykaz ważniejszych skrótów

- CD – kodowa rozłączność (ang. *code-disjoint*)
- ES – odporność na błędy (ang. *error-secure*)
- FS – odporność na uszkodzenia (ang. *fault-secure*)
- SC – samosprawdzalność (ang. *self-checking*)
- SM – układ sekwencyjny (ang. *sequential machine*)
- STC – samotestowalny układ kontrolny (ang. *self-testing checker*)
- ST – samotestowalność (ang. *self-testing*)
- TSC – całkowita samosprawdzalność (ang. *totaly self-checking*)

Rozdział 1

Wprowadzenie

1.1 Motywacja

O STATNIE lata przyniosły szybki rozwój różnorodnych zastosowań systemów cyfrowych. Z rosnącym zapotrzebowaniem na systemy cyfrowe wiążą się coraz większe wymagania co do ich poprawności, ciągłości i bezpieczeństwa funkcjonowania. Jednak uszkodzenia systemów cyfrowych, pomimo rozwoju technologii i metod projektowania, są zjawiskiem nieuchronnym i w przypadku wystąpienia mogą mieć fatalne skutki — od zwykłej niewygody użytkowników systemu [32], poprzez znaczne straty finansowe i materialne, do zagrożenia zdrowia i życia ludzkiego [49], [50]. Informacje dostarczane przez uznanych producentów sprzętu komputerowego, takich jak Tandem Computers i Sun Microsystems [32] oraz IBM [104], [52], [94], jednoznacznie wskazują, że poleganie wyłącznie na wysokiej niezawodności układów VLSI oraz programowych środkach kontroli integralności danych staje się niewystarczające w systemach mikroprocesorowych ogólnego przeznaczenia. Z drugiej strony, biorąc pod uwagę, że ponad 90% uszkodzeń ma charakter przemijający [32], [88] oraz że uszkodzenia te są bardzo trudne do wykrycia, problemu tego nie można bagatelizować.

Od szeregu lat obserwowano zbyt dużą podatność gęsto upakowanych układów VLSI na niewykrywalne uszkodzenia przemijające, spowodowane naładowanymi cząsteczkami występującymi w promieniowaniu kosmicznym — głównie w układach pracujących w samolotach i satelitach — oraz rozkładem śladowych ilości uranu i toru, występujących w materiałach używanych do produkcji układów scalonych [104]. Jednakże, pod koniec lat 90. stwierdzono zwiększoną wrażliwość układów — pracujących bynajmniej nie w kosmosie, o czym wiadano od dawna, lecz na powierzchni Ziemi — na neutrony pochodzenia kosmicznego, które mają wystarczającą energię aby zmieniać stan przerzutników [41].

Konieczne staje się zatem projektowanie i realizacja systemów cyfrowych mających zdolność kontynuowania poprawnej pracy, pomimo uszkodzeń wewnętrznych: tolerowanie uszkodzeń staje się niezbędną cechą coraz większej liczby systemów cyfrowych.

Do licznych przykładów stale rozszerzających się zastosowań systemów tolerujących uszkodzenia należą: systemy sterowania procesami przemysłowymi, systemy stosowane w medycynie, serwery sieci komputerowych, systemy obsługi transakcji (systemy bankowe i rezerwacji miejsc), systemy telekomunikacyjne, systemy wojskowe, lotnicze i satelitarne. Szereg zastosowań wymaga zapewnienia bezbłędnego funkcjonowania systemu cyfrowego, tym samym konieczne jest wykrywanie uszkodzeń wewnętrznych na bieżąco, tj. podczas normalnego funkcjonowania (np: systemy dla zastosowań ko-

smicznych [58], stosowane w samochodach systemy ABS firmy Bosch [26]).

W praktyce, podejście to polega na użyciu kodów wykrywających błędy do kodowania stanów układów cyfrowych i ich realizacji jako układów samosprawdzalnych (SC) (ang. *self-checking*) [11]. Układy takie umożliwiają wykrywanie uszkodzeń zarówno trwałych jak i przemijających na bieżąco (CED) (ang. *concurrent error detection, on-line testing*), tzn. w czasie normalnej pracy systemu. W pracy [11] wprowadzono podstawowe własności takich układów: samotestowalność (ang. *self testing* ST) i odporność na uszkodzenia (FS) (ang. *fault secure*). Formalne definicje wymienionych własności zostały podane przez Andersona [1] rozszerzone później o ważną klasę układów samosprawdzalnych, układy całkowicie samosprawdzalne (ang. *totally self checking* TSC) przez Andersona i Metze [2]. Smith i Metze zdefiniowali także klasę układów silnie odpornych na uszkodzenia (SFS) (ang. *strongly fault secure*) [91], które charakteryzują większą grupę układów osiągających cel TSC.

1.2 Samosprawdzalne układy sekwencyjne

Podstawowe prace z zakresu układów samosprawdzalnych dotyczyły zagadnień związanych z syntezą samosprawdzalnych układów kombinacyjnych. W latach 60. rozważano także możliwość syntezy samosprawdzalnych układów sekwencyjnych (SM). Prace dotyczące tego zagadnienia można podzielić na dwie grupy dotyczące różnych klas układów: układów bezpiecznych oraz układów całkowicie samosprawdzalnych.

Syntezę bezpiecznych (ang. *fail safe*) SM rozważano w [13], [16], [24], [29], [46], [96], [97]. Należy tutaj wymienić dwa podstawowe podejścia, które miały duży wpływ na rozwój metod syntezy układów TSC. W pracy [24], rozszerzającej podejście z [97], Diaz podaje definicję oraz metodę syntezy bezpiecznych SM o wejściach oraz wyjściach zakodowanych kodami m/n . Zagadnienia syntezy układów bezpiecznych, wykorzystujących kody m/n , zostały rozwinięte później w pracach [23], [22] dotyczących syntezy układów TSC.

Odmienne podejście zaprezentowali Maki [46] oraz Sawin [16], podając metody syntezy bezpiecznych SM polegające na analizie grafu przejść oraz doborze kodowania stanów. Metody te znalazły swoje rozwinięcie w pracach dotyczących syntezy układów TSC SM [6], [18], [27], [61], [62], [63], [65].

1.3 Przegląd metod syntezy TSC układów sekwencyjnych

Najbardziej restrykcyjne, a zarazem najtrudniejsze do osiągnięcia, są założenia TSC [91]. Definicje układów ST, FS i TSC z [1], [2], dotyczące tylko układów kombinacyjnych, zostały sformułowane dla SM przez Diaza [21], [22] oraz Davida [19]. Analiza literatury pokazuje całkowity brak zgodności autorów dotyczący definicji TSC SM. Rozważania Diaza dotyczące tylko układów Moore'a zostały rozszerzone przez Davida [19], który podaje definicje ST i FS SM Moore'a i Mealy'ego oraz definiuje pojęcie normalnej pracy układu. Podstawowym problemem syntezy samosprawdzalnych SM jest weryfikacja, czy sekwencja słów wejściowych odpowiada poprawnej sekwencji słów wyjściowych. Ponieważ uszkodzenia mogą wpłynąć zarówno na układ realizujący funkcję przejść, jak i funkcję wyjść, wymagana jest poprawność obydwu tych funkcji. Definicje podane w [22] oraz [19] są trudne do weryfikacji, dlatego szereg autorów przyjmuje łatwiejsze do weryfikacji założenia TSC SM. Warunkiem TSC SM

jest zabezpieczenie wejść, wyjść oraz stanów wewnętrznych kodami detekcyjnymi oraz synteza funkcji przejść/wyjść jako ST oraz CD.

Należy także zaznaczyć, że w odróżnieniu od typowo przyjmowanych założeń syntezy TSC układów kombinacyjnych, zbiór rozpatrywanych uszkodzeń został ograniczony. Podstawowym założeniem przyjętym w literaturze jest brak uszkodzeń na liniach zegarowych (wyjątek stanowią prace Kia [34], [35]). W większości metod syntezy TSC SM autorzy zakładają brak uszkodzeń w liniach wejściowych układu (wyjątek [28]). Szereg metod syntezy TSC SM jest adaptacją metod syntezy TSC układów kombinacyjnych, tylko nieliczne prace są rozwiązaniami dotyczącymi tylko SM. Prace dotyczące TSC SM można podzielić na dwie podstawowe grupy: układy TSC wykorzystujące kody detekcyjne oraz układy TSC wykorzystujące układ kontrolujący.

1.3.1 TSC układy sekwencyjne wykorzystujące EDC

Wiele znanych dziś metod syntezy TSC SM wykorzystuje kody detekcyjne. W podejściu tym wejścia, stany wewnętrzne oraz wyjścia układu zakodowane są kodami detekcyjnymi. Synteza TSC SM sprowadza się do syntezy TSC układu kombinacyjnego, realizującego funkcję przejść/wyjść. Prace dotyczące tej klasy układów można podzielić na dwie podstawowe grupy: 1) wykorzystujące kody parzystości, 2) wykorzystujące kody nieuporządkowane.

Rozwiązania podane w [92] wywodzą się ze znanych metod z zakresu syntezy SC układów kombinacyjnych, wykorzystujących wbudowane układy parzystości (ang. *build-in parity protection*). Dla zdefiniowanego zbioru uszkodzeń F , układ jest symulowany dla każdego uszkodzenia, aby określić wpływ uszkodzeń na realizowaną funkcję. Na tej podstawie generowana jest tablica niepoprawnych funkcji (ang. *faulty functions*). Niestety metody te cechuje duża złożoność obliczeniowa, co czyni je nieopłacalnymi szczególnie dla dużych układów. W pracach [20], [98], [103] zastosowano znane metody syntezy układów kombinacyjnych, podejście polegające na zakodowaniu wejść, stanów wewnętrznych oraz wyjść i syntezie całego układu jako w pełni rozgałęzionego. W celu redukcji złożoności, w pracach [98], [103] zaproponowano podział słów wyjściowych oraz słów stanu na grupy kodowane kodem parzystości oraz syntezy całego układu w taki sposób, aby nie było wspólnych podukładów w ramach jednej grupy kodowej, współdzielenie układów jest natomiast możliwe pomiędzy różnymi grupami. Pojedyncze uszkodzenie będzie zawsze wykryte, ponieważ może spowodować jeden wykrywalny błąd w ramach jednej grupy kodowej.

Większość metod syntezy układów TSC znanych z literatury wykorzystuje kody nieuporządkowane [9], [10], [19] [21], [22], [28], [33], [39], [40], [61]. Wybór kodów nieuporządkowanych nie jest przypadkowy, zważywszy na ich własności (podrozdział 2.5). Podejście to polega na zastosowaniu kodów nieuporządkowanych oraz na syntezie układów jako bezinwerterowych. Analiza istniejących metod syntezy TSC SM pokazuje, że kluczowe znaczenie mają następujące zagadnienia: dobór kodu detekcyjnego, istnienie samotestowalnych układów kontrolnych (STC) dla zastosowanych kodów oraz synteza TSC układu realizującego funkcję przejść-wyjść.

Autorzy metod syntezy TSC SM proponują szereg metod doboru kodu. W większości prac przyjęto, że wejście układu zakodowane jest kodem z inwersyjnym powtórzeniem, uzyskanym poprzez zanegowanie wejść pierwotnych. Założenie to powoduje, że układ nie jest zabezpieczony przed błędami na wejściu oraz przed uszkodzeniami wejściowych linii pierwotnych, wyjątek stanowi praca [28].

W literaturze podano szereg metod kodowania stanów wewnętrznych oraz wyjść układów. Metody zaproponowane przez Diaza [21], [22], bazują na zastosowaniu kodu z inwersyjnym powtórzeniem dla kodowania wyjść oraz kodów m/n do kodowania stanów wewnętrznych układów SM. David w [19] zaproponował natomiast zastosowanie kodów $1/n$ do kodowania stanów oraz kodu m/n do kodowania wyjść układu. Podobne podejście zaproponowano w pracach [9], [10], [61], gdzie zweryfikowano własności układów wzorcowych syntezowanych z zastosowaniem kodów m/n do kodowania stanów oraz wyjść układów. W [33] Jha oraz Wang zaproponowali zastosowanie kodów m/n gdzie $n \leq 4$, dla kodowania stanów wewnętrznych oraz kodu Bergera dla kodowania wyjść układu. Ciekawe rozwiązanie polegające na podziale słów kodu wyjściowego w taki sposób, aby stanowiły konkatenację kodów $1/n$ oraz kodu z inwersyjnym powtórzeniem zaproponowali Lai oraz Wey w [39], [40]. Autorzy podają w [39], [40], że stosują także kody $2/n$, jednak analiza wykazała, że w rzeczywistości kody te są konkatenacją kodów $1/n$. Zagadnieniem, które ma ścisły związek z doбором kodów, są stany obojętne (ang. *don't care*). Niestety, zagadnienie kodowania wejść oraz wyjść w sytuacji, gdy występują stany obojętne, jest pomijane przez autorów tych prac. Jak zostanie pokazane (rozdział 4), stany obojętne mogą mieć kluczowe znaczenie w otrzymaniu TSC układu.

Zastosowanie kodów detekcyjnych do kodowania wejść, stanów wewnętrznych oraz wyjść układów pociąga za sobą konieczność użycia układów kontrolnych STC. Autorzy większości prac [5], [9], [10], [19], [21], [22], [33], [59] co prawda stwierdzają konieczność zastosowania układów STC, jednak faktu ich istnienia, oraz ich ST dla kodów niekompletnych nie weryfikują. Przyjęcie takiego założenia często uniemożliwia syntezę całego układu TSC, ponieważ dla osiągnięcia celu TSC wymagana jest ST układów kontrolnych, co nie zawsze jest zagwarantowane w przypadku, gdy pojawiają się kody, dla których układy STC nie istnieją (np.: $1/3$) lub występują kody niekompletne. Zagadnienia te rozważane są częściowo w pracach [33], [39], [40], gdzie autorzy sprawdzają istnienie układów kontrolnych dla stosowanych kodów, jednak nie weryfikują ich samotestowalności.

Zastosowanie kodów nieuporządkowanych dla zakodowania wejść oraz wyjść układu umożliwia syntezę układu funkcjonalnego jako bezinwerterowego o strukturze 2-poziomowej AND-OR. Cel syntezy SM jako TSC można osiągnąć, stosując jedną z dwóch struktur TSC/STC lub TSC/CD (rozważane jedynie w [28]). Dla obu struktur należy wykazać zarówno ST, jak i FS układu kombinacyjnego realizującego funkcję przejść/wyjść. Dla układów TSC/CD należy ponadto zapewnić także CD dla niekodowych pobudzeń. W układach TSC/CD cel ten osiągany jest poprzez syntezę układu kombinacyjnego realizującego funkcję przejść/wyjść jako CD. W przypadku gdy jest to niemożliwe, konieczne jest zastosowanie układów STC monitorujących wejścia oraz stany wewnętrzne. Układ STC dla kodu wyjściowego należy zastosować zarówno dla TSC/CD jak i TSC/STC. Większość metod syntezy samosprawdzalnych SM rozpatruje układy TSC/STC [23], [22], [33], [39].

Kluczowym zagadnieniem syntezy TSC SM jest synteza TSC układu funkcjonalnego realizującego funkcję przejść/wyjść. Zagadnienie syntezy TSC układów, wykorzystujących kody nieuporządkowane, rozważano w pracach Diaza [23], [22] dotyczących SM. Zastosowanie mają także metody opisane w pracach dotyczących tylko układów kombinacyjnych [70], [90]. W [23], [22] Diaz podaje warunki weryfikacji ST układów o rozłącznych iloczynach, o wejściach, stanach oraz wyjściach zakodowanych kodem m/n i pokazuje, że układy te są również FS. Diaz podaje także metodę modyfikacji układów, która prowadzi do ST układu w przypadku, gdy układ tej własności nie posiada. Zaproponowany

algorytm polega na modyfikacji funkcji układu, poprzez eliminację zmiennych, dla których nie jest zachowana ST, ale niestety może to prowadzić do utraty CD. Formalne kryteria podane przez Diaza zostały uogólnione przez Piestraka w [70] dla układów o wspólnych podukładach. Prace [23], [22], [70] dotyczą tylko układów wykorzystujących kody m/n i z inwersyjnym powtórzeniem. Inna metoda modyfikacji układu prowadząca do realizacji ST została podana w [28] (stanowiąca rozwinięcie [70]). Jak pokazano, uzyskanie ST jest możliwe bez modyfikacji funkcji układu poprzez dodanie dodatkowej warstwy bramek AND. Zaproponowane rozwiązanie ma szczególne znaczenie dla układów TSC/CD, gdyż nie modyfikuje funkcji realizowanej przez układ, dzięki czemu układ może być CD dla niekodowych pobudzeń wejściowych.

Autorzy pozostałych prac nie podają, w jaki sposób weryfikują ST, lub podają, że wystarczające dla ST jest zastosowanie kodów nieuporządkowanych oraz syntezy układów jako bezinwerterowych [9], [10]. W pracy Jha i Wanga [33] podano co prawda metodę syntezy układów i zaprezentowano program CAD dla syntezy TSC SM, jednak jak sam autor przyznaje, cel 100% pokrycia uszkodzeń nie jest osiągnięty dla większości przypadków.

Zagadnienie CD SM nie było szeroko badane, ponieważ większość autorów przyjmuje, że wejścia oraz stany wewnętrzne monitorowane są układami STC, co nie wymaga CD układu funkcjonalnego. Problem ten był analizowany jedynie w pracach [55], [70] oraz [36]. Piestrak w [70] podaje formalne warunki weryfikacji CD układów kombinacyjnych o wejściach oraz wyjściach zakodowanych m/n . Nanya w [55] oraz Kia w [36] podają metodę modyfikacji układu, która prowadzi do realizacji CD. Autorzy obu metod podają algorytm dodawania dodatkowych linii kodu z inwersyjnym powtórzeniem, która prowadzi do realizacji CD. Niestety modyfikacja ta może być zastosowana tylko w niektórych przypadkach i niesie za sobą znaczny nadmiar informacyjny i sprzętowy.

1.3.2 TSC SM wykorzystujące układy kontrolujące

Cel TSC SM można także osiągnąć poprzez analizę i modyfikację grafu SM. W szeregu metod w celu syntezy TSC SM, analizie poddawany jest graf przejść (poziom specyfikacji układu), a nie struktura logiczna układu funkcjonalnego. W pracach [62], [65], [64], [81], [87] rozważano metodę syntezy TSC SM polegającą na zastosowaniu dodatkowego układu SM monitorującego funkcjonowanie właściwego SM. W układach tego typu porównywane są algebraiczne relacje pomiędzy wyjściem właściwego SM i układu monitorującego. Układ monitorujący na podstawie wejścia i wyjścia właściwego układu weryfikuje jego poprawność. Wykrycie różnicy oznacza wystąpienie błędu na wyjściu SM lub monitorującego SM. Problemem w realizacjach układów tego typu jest synteza zredukowanego układu SM, lecz niestety cel tej syntezy jest trudny do osiągnięcia.

Do grupy metod analizujących specyfikację układu można też zaliczyć rozwiązania bazujące na odpowiednim doborze kodowania stanów. Rozwiązania te częściowo wykorzystują techniki związane z syntezą układów wykorzystujących kody nieuporządkowane. W pracach [6], [5], [18], [61], [62], [63], zaproponowano metody kodowania stanów wewnętrznych, które mają gwarantować TSC układu. Autorzy proponują taki dobór kodów dla poszczególnych stanów układu, aby odległość Hamminga pomiędzy kodami kolejnych stanów była stała. Niepoprawne przejście może być wykryte, ponieważ w tym przypadku odległość Hamminga pomiędzy kodami kolejnych stanów ulegnie zmianie. W pracach Sapozhnikova [84], [86], [85], oraz Lopuchny [44] podano metodę syntezy TSC SM bazującą na analizie

grafu przejść oraz kodowaniu stanów kodami m/n . Podobne metody zastosował Gössel w [27], przyjmując jednak założenie, że stany wewnętrzne SM są obserwowalne. Niestety metody te nie znalazły zastosowania praktycznego, głównie z powodu trudności związanych z syntezą zredukowanego układu monitorującego. Do metod, które mogą mieć praktyczne znaczenie można zaliczyć pracę [64], której autorzy zaproponowali zastosowanie kodów separowalnych do kodowania wyjść oraz syntezę układu w taki sposób, aby wyjście układu monitorującego stanowiło część kodową kodu, natomiast wyjście układu właściwego część informacyjną kodu.

1.4 CAD

Złożoność współczesnych układów VLSI wymusza konieczność stosowania oprogramowania wspomagającego projektowanie (ang. **Computer Aided Design (CAD)**). Należy tutaj zaznaczyć, że problemy związane z syntezą układów w wersji TSC są dużo bardziej złożone niż związane z syntezą typowych układów cyfrowych z powodu konieczności rozbudowy układu związanej z zapewnieniem TSC oraz analizą wpływu uszkodzeń na syntezowany układ. Weryfikacja i projektowanie złożonych układów TSC jest zbyt złożonym problemem, aby możliwe było jego rozwiązanie bez wykorzystania narzędzi CAD. Niestety pomimo rozwoju narzędzi CAD do syntezy układów cyfrowych, znane są z literatury tylko nieliczne przykłady systemów wspomagających syntezę TSC SM [39], [40], [33], [20].

Lai i Wey [39], [40] zaprojektowali system CAD *SOLiT* wspomagający projektowanie TSC SM. Autorzy, aby osiągnąć cel TSC kodują wejścia, stany wewnętrzne oraz wyjścia kodami nieuporządkowanymi oraz stosują układy STC do ich monitorowania. W celu redukcji złożoności STC dla kodu wyjściowego, autorzy zaproponowali algorytm podziału kodu wyjściowego tak, aby stanowił on konkatencję kodów $1/n$ oraz kodu z inwersyjnym powtórzeniem. Dla kodowania wejść stosują kody Bergera oraz kod z inwersyjnym powtórzeniem. Układy kombinacyjne realizujące funkcję przejść oraz wyjść zrealizowane są jako bezinwerterowe, co gwarantuje, że uszkodzenia pojedyncze powodują tylko błędy jednokierunkowe. Podstawową wadą tego podejścia to brak zapewnienia ST dla wszystkich uszkodzeń pojedynczych układu funkcjonalnego, stąd też pokrycie błędów nie jest znane i trudno mówić o osiągnięciu celu TSC. Ponadto autorzy nie weryfikują ST zastosowanych układów STC dla kodów niekompletnych.

Podobne podejście można znaleźć w pracy Jha i Wang [33], gdzie stosuje się kody nieuporządkowane do kodowania wyjść oraz stanów wewnętrznych. Wejścia układu nie są kodowane, zastosowane są natomiast inwertery na liniach wejść pierwotnych, w wyniku czego układ nie jest zabezpieczony przed błędami na wejściu i uszkodzeniami linii wejściowych. Stany wewnętrzne kodowane są kodami m/n , natomiast wyjścia układu kodowane są kodami Bergera. Zastosowanie kodów Bergera, szczególnie dla układów o dużej liczbie wyjść powoduje, że: 1) układy STC mogą nie istnieć (nie można zapewnić ich ST) lub ich złożoność jest zbyt duża, 2) układy STC nie są ST dla niekompletnego kodu wyjściowego. Zastosowana bezinwerterowa implementacja funkcji przejść oraz wyjść nigdy nie gwarantuje ST układu. Wyniki podawane przez autorów pokazują, że w syntezy układach było od 4% do 7% niewykrywalnych uszkodzeń pojedynczych, stąd też cel TSC całego układu nie został przez nich osiągnięty.

Przykładem rozbudowanego systemu CAD do syntezy TSC SM jest ASYL-SdF zaprojektowany przez Rocheta i Leveugla [82]. System ten oferuje cztery metody syntezy TSC SM oraz trzy me-

tody syntezy układów tolerujących uszkodzenia. Najprostsza metoda syntezy układy TSC polega na powieleniu (zdublowaniu) całego SM oraz porównywaniu wyjść. W rozwiązaniu tym otrzymuje się największy nadmiar sprzętowy (ponad 100%). Złożoności układów tego typu stosowane są jak punkt odniesienia dla innych metod projektowania. Autorzy proponują modyfikację powyższej metody, polegającą na powieleniu układu funkcji przejść i rejestrów stanu, natomiast wyjścia układu pozostają niezabezpieczone, a porównywane są stany wewnętrzne obu zdublowanych układów. System ASYL-SdF oferuje także dwie metody, bazujące na analizie grafu przejść. Obok metod syntezy układów TSC, system umożliwia syntezę SM tolerujących uszkodzenia poprzez potrojenie SM z głosowaniem na wyjściu (TMR), lub potrojenie układu funkcji przejść i rejestrów przy niezabezpieczonych wyjściach (głosowanie odbywa się na wyjściach rejestrów stanu), lub też zastosowanie kodów korygujących (ECC) dla kodowania stanów przy wyjściach niezabezpieczonych. Należy także wspomnieć o rozbudowanym systemie CAD do syntezy kombinacyjnych układów TSC zaprojektowanym przez De [20]. Autor proponuje trzy rozwiązania: powielenie układu, kodowanie wyjść kodem Bergera i realizację układu jako beziwerterowego, oraz kodowanie wyjść kodem parzystości.

Niestety wszystkie z wymienionych pakietów CAD oferują nieznany poziom zabezpieczenia układu. Żadna z przedstawionych metod nie oferuje również zabezpieczenia przed błędami na wejściu układu.

Przykłady praktycznego zastosowania TSC SM (głównie samosprawdzalnych jednostek sterujących) zostały opisane w [14], [15], [23], [25], [30], [31], [36], [42], [45], [53], [56], [60], [66], [89], [93], [95], [100], [101], [102]. W większości prac cel 100% TSC nie został osiągnięty, jednak prace te mają szczególne znaczenie, ponieważ podane są w nich praktyczne przykłady wykorzystania metod syntezy układów TSC. Na tej podstawie można wyciągnąć szereg wniosków dotyczących stosowanych technik i wymagań nakładanych na układy dla zastosowań praktycznych. Analiza prac pokazuje, że szczególnie użyteczne są następujące kody m/n : kody $m/2m$, kody $m/2(m \pm 1)$ i kody $1/n$. Analiza ujawnia także mankamenty znanych metod syntezy układów TSC. Typowo jednostka sterująca to nie jeden SM, lecz kilka współpracujących ze sobą układów. Zagadnienia syntezy tego typu współpracujących układów TSC stanowią w dalszym ciągu otwarty problem.

1.5 Mankamenty znanych metod syntezy TSC SM

Analiza prac, podana w [75], dotyczących syntezy TSC SM pokazuje szereg problemów i zagadnień, które pozostały nie rozwiązane przez autorów lub, co gorsza, pominięte w rozważaniach.

Pierwszym podstawowym problemem jest brak zgodności autorów prac co do definicji TSC SM. Przypomnijmy, że układ kombinacyjny TSC dla zbioru uszkodzeń wewnętrznych ma dwie własności: (1) jest odporny na uszkodzenia (ang. *fault-secure (FS)*) oraz (2) jest samotestowalny (ang. *self-testing (ST)*). Diaz [22] i David [19] jako pierwsi zauważyli, że definicje *ST* i *FS* dotyczące układów kombinacyjnych nie mają zastosowania dla SM. Wykazali oni, że w odróżnieniu od układów kombinacyjnych, poleganie tylko na monitorowaniu wejść i wyjść nie jest wystarczające dla SM, ponieważ uszkodzenie wewnętrzne może spowodować, że pewne stany wewnętrzne mogą stać się nieosiągalne w czasie normalnej pracy układu. Autorzy późniejszych prac unikają niestety podawania definicji *ST*, *FS* oraz TSC, przez co szereg metod znanych z literatury oferuje nieznany poziom zabezpieczenia przed błędami.

W większości prac, błędy na wejściach pierwotnych nie są rozważane. Autorzy przyjmują, że kod

z inwersyjnym powtórzeniem jest generowany dla wszystkich linii wejściowych z wykorzystaniem inwerterów. Oczywiście powoduje to brak zabezpieczenia układu przed błędami na liniach wejściowych oraz przed uszkodzeniami tych linii. Jest to znaczne ograniczenie, które redukuje zakres zastosowania opracowanych metod. Brak zabezpieczenia linii wejściowych może być akceptowalny w niektórych systemach (np. gdy sygnały wejściowe generowane są przez sensory), jednak jest zupełnie nie do akceptacji w złożonych systemach TSC, w których współpracuje kilka modułów TSC (nawet najprostsze urządzenia sterujące mogą składać się z kilku SM).

Większość metod syntezy TSC SM zapewnia tylko częściowe zabezpieczenie przed błędami spowodowanymi uszkodzeniami wewnętrznymi, dlatego w wyniku pokrycie uszkodzeń jest mniejsze niż 100%. Ponadto, nie ma żadnych metod formalnej weryfikacji i wykrywania nietestowalnych uszkodzeń.

Bardzo często, kod detekcyjny wybrany dla zabezpieczenia wejścia, stanów wewnętrznych oraz wyjścia, zawiera tylko część słów kodowych (kod niekompletny). Większość znanych metod projektowania TSC SM nie gwarantuje, że zastosowane układy kontrolne (STC) zachowują ST dla niekompletnych kodów (występujący zbiór słów wejściowych może nie testować wszystkich uszkodzeń wewnętrznych). Dlatego w metodach syntezy TSC SM należy stosować tylko takie kody, które gwarantują istnienie STC oraz jego ST dla niekompletnego kodu wyjściowego. Autorzy prac, weryfikując metody syntezy TSC SM, używają kodu z inwersyjnym powtórzeniem. W niektórych układach uzyskują w ten sposób niekompletny kod, który zawiera tylko pierwszych $2^K + 1$ słów kodowych. W efekcie tylko dla jednego słowa kodowego najbardziej znaczący bit ma wartość 1. Do niedawna [77] dla kodu takiego nie był znany kombinacyjny STC.

Problemem pomijanym w rozważaniach są także stany obojętne (ang. *don't care*): wejściowe oraz wyjściowe. Wejściowe stany obojętne są konsekwencją uproszczenia specyfikacji SM i w trakcie syntezy układu wymagają rozwinięcia do pełnego opisu, co często wiąże się z dużą złożonością obliczeniową związaną z eksplozją kombinatoryczną. Wyjściowy stan obojętny oznacza, że stan danej linii wyjściowej może być ustalony jako 0 lub 1 w czasie syntezy układu. Własność ta ma szczególne znaczenie w syntezie układów TSC, gdyż od odpowiedniego doboru stanu wyjściowych linii obojętnych może zależeć rozmiar słowa kodu wyjściowego oraz, co ważniejsze, istnienie i złożoność STC dla użytego kodu wyjściowego. Autorzy prac nie podają, w jaki sposób kodują wyjścia układów, w których występują stany obojętne. Jedynie w pracy [39] autor podaje, że dla wyjściowych stanów obojętnych przyjmuje wartość 0. Wydaje się, że taka eliminacja wyjściowych stanów obojętnych ogranicza możliwości minimalizacji funkcji przejść/wyjść i wymaga wprowadzenia nadmiarowej liczby dodatkowych wyjść.

Własności ST, FS oraz CD wydają się być zależne od przyjętego kodowania. Przeprowadzona analiza układów wzorcowych pokazuje przykłady układów, w których od doboru kodowania stanów wewnętrznych zależy realizacja układu, jako TSC. Dlatego porównywanie metod realizacji SM z metodami znanymi z literatury jest niemożliwe lub utrudnione, ponieważ autorzy nie podają metody doboru słów kodowych do kodowania poszczególnych stanów wewnętrznych.

Metody syntezy TSC SM są testowane i weryfikowane na zbiorze układów wzorcowych (ISCAS'85 [8], ISCAS'89 [7]). Pierwotnie, zbiory te były przygotowane do weryfikacji metod CAD lub automatycznego generowania zbiorów testów. Problemy spotykane w zagadnieniach syntezy SM są odmienne od tych z zakresu syntezy TSC SM, przez co szereg układów wzorcowych nie nadaje się do weryfi-

kacji metod syntezy układów TSC. Przykładowo, część układów zawiera linie o stałym stanie (np. modulo 12 stałe wyjście 1), w innych układach linie mają identyczne wartości. W takich przypadkach niemożliwe jest zapewnienie ST układu. Dla szeregu układów wykazano, że część stanów wewnętrznych jest nieosiągalna (np. **ex2**). W literaturze, autorzy albo nie rozważają tych przypadków lub, co gorsza podają, że synteza układu TSC jest możliwa (np.: [33] układ **ex2**). Ważnym problemem jest również ocena i porównywanie złożoności uzyskanych układów. W znanych publikacjach autorzy podają liczbę literali, nie podają jednak jaką metodą minimalizowali układ oraz nigdzie nie podają, czy analizując złożoność, uwzględniają układy STC. Wadą wszystkich znanych metod syntezy TSC SM jest ich nieskuteczność w zagadnieniach syntezy współpracujących SM, spowodowana brakiem zabezpieczenia przed błędami na wejściu i uszkodzeniami linii wejściowych.

Podsumowując żadna ze znanych metod syntezy TSC SM nie oferuje 100% lub oferuje nieznany stopień zabezpieczenia układu. Ponadto żadna ze znanych metod nie rozpatruje błędów na wejściu układu co ma szczególne znaczenie w systemach, w którym współpracuje szereg SM. Nie są także znane metody nadające się do automatycznej syntezy TSC SM. Dlatego też konieczne staje się opracowanie nowych metod syntezy TSC SM, nadających się do zastosowania w CAD gwarantujących 100% stopień zabezpieczenia układu lub pozwalających zidentyfikować niewykrywalne uszkodzenia (które mogą być wykrywane deterministycznymi testami off-line).

1.6 Cel i teza pracy

Celem pracy jest opracowanie nowych metod syntezy TSC SM wykorzystujących kody nieuporządkowane:

- nadających się do automatycznej syntezy TSC SM,
- pozwalających uzyskać 100% pokrycie uszkodzeń i błędów,
- pozwalających zidentyfikować niewykrywalne uszkodzenia i ich przyczyny.

Tezą stawianą w tej pracy jest: *istnieje możliwość automatycznej syntezy całkowicie samosprawdzalnych układów sekwencyjnych z wykorzystaniem kodów nieuporządkowanych, która gwarantuje 100% wykrywalność uszkodzeń i błędów.*

Rozdział 2 prezentuje podstawowe własności układów sekwencyjnych oraz kodów wykrywających błędy, ze szczególnym uwzględnieniem kodów nieuporządkowanych. Przedstawiono własności układów kombinacyjnych o wejściach i wyjściach zakodowanych kodami nieuporządkowanymi. W rozdziale tym podano także podstawowe definicje dotyczące układów TSC oraz przedstawiono rozważane w tej pracy struktury TSC układów sekwencyjnych.

Rozdział 3 poświęcony jest dwóm metodom syntezy układów TSC SM o strukturach TSC/CD oraz TSC/STC. W podrozdziale 3.1 zaprezentowano metody syntezy TSC kodowo-rozłącznych SM (TSC/CD) i podano formalne warunki kodowej rozłączności (ang. *code-disjoint (CD)*) oraz samotestowalności (ang. *self-testing (ST)*) układów kombinacyjnych realizujących funkcję przejść i wyjść w TSC SM. Podano także metody modyfikacji wewnętrznej struktury układu realizującego funkcję przejść/wyjść tak, aby był on ST oraz CD. Dla sformułowanych twierdzeń podano algorytmy umożliwiające automatyczną weryfikację własności ST i CD. W podrozdziale 3.2 omówiono metody syntezy TSC układów sekwencyjnych (TSC/STC), wykorzystujących samotestowalne układy kontrolne (STC) dla monitorowania wejść, stanów wewnętrznych oraz wyjść. W podrozdziale tym pokazano metody minimalizacji układów kombinacyjnych realizujących funkcję przejść/wyjść oraz wykazano także, że opracowane algorytmy minimalizacji gwarantują ST tych układów.

Rozdział 4 poświęcony jest zagadnieniom automatycznej syntezy TSC SM. Szczególny nacisk położono na podanie metod doboru kodów gwarantujących istnienie STC. Podano warunki wyboru kodu stanów wewnętrznych zapewniające ST układów STC. Przedstawiono nową metodę konstrukcji kodu wyjściowego oraz syntezy STC, wykorzystującą wyjściowe stany obojętne do zapewnienia ST. W rozdziale tym przedstawiony jest opracowany w ramach rozprawy pakiet oprogramowania TSC-SM CAD (ang. *Totally Self Checking Sequential Machines CAD*).

W rozdziale 5 zawarto wyniki syntezy układów wzorcowych z zastosowaniem opracowanych algorytmów oraz ocenę jakościową zaproponowanych metod syntezy. Syntezowano układy w dwóch rozważanych w pracy strukturach TSC SM: TSC/CD oraz TSC/STC. Należy tutaj podkreślić, że żadna ze znanych z literatury metod syntezy TSC SM nie rozważała układów TSC/CD. Przeprowadzona ocena zaproponowanych metod syntezy TSC/SM pokazuje, że istnieje możliwość syntezy układów TSC zapewniających 100% pokrycie uszkodzeń wewnętrznych i błędów wyjściowych zarówno w wersji TSC/CD jak i TSC/STC.

W rozdziale 6 podsumowano uzyskane wyniki oraz przedstawiono ważne, poznawczo i praktycznie, zagadnienia wymagające dalszych badań.

Rozdział 2

Pojęcia podstawowe

W NINIEJSZYM rozdziale, składającym się z 6 podrozdziałów, zawarte są podstawowe definicje i pojęcia dotyczące SM i układów samosprawdzalnych. W podrozdziale 2.1 podane zostały definicje SM oraz został zdefiniowany model rozpatrywanego SM. Następnie podane są przyjęte założenia dotyczące przyjętego modelu uszkodzeń i spowodowanych nimi błędów. W podrozdziale 2.3 podano definicje dominacji i równoważności uszkodzeń. W podrozdziałach 2.4 i 2.5 scharakteryzowano kody wykrywające błędy ze szczególnym uwzględnieniem kodów nieuporządkowanych oraz podano własności układów kombinacyjnych wykorzystujących kody nieuporządkowane. W podrozdziale 2.6 podane są podstawowe definicje dotyczące układów samosprawdzalnych oraz analizy skutków występujących w nich uszkodzeń i spowodowanych nimi błędów.

2.1 Modele układów sekwencyjnych

Wyróżnia się dwa podstawowe modele SM układy Mealy'ego oraz Moore'a. W pracy tej rozważania ograniczone zostaną do SM Mealy'ego (rys. 2.1).

Definicja 2.1 [51] *Układ sekwencyjny SM Mealy'ego jest to uporządkowana piątka*

$$\text{SM}(\mathcal{X}, \mathcal{S}, \mathcal{Y}, \delta, \omega)$$

gdzie:

$\mathcal{X}$ – zbiór słów wejściowych $X = \{x_1, \dots, x_n\}$,

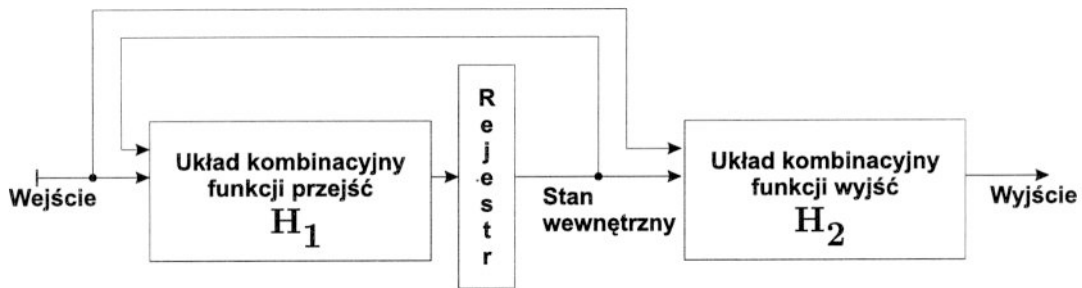
$\mathcal{S}$ – zbiór stanów wewnętrznych $S = \{s_1, \dots, s_k\}$,

$\mathcal{Z}$ – zbiór słów wyjściowych $Y = \{y_1, \dots, y_m\}$,

$\delta(x, s)$ – funkcja przejść zależna od stanu wewnętrznego oraz wejścia układu,

$\omega(x, s)$ – funkcja wyjść zależna od stanu wewnętrznego oraz wejścia układu.

W praktycznych realizacjach SM składa się z układów $\mathbf{H}_1$ oraz $\mathbf{H}_2$ oraz rejestru (pamięci). Układ $\mathbf{H}_1$ jest układem kombinacyjnym, który na podstawie stanu wewnętrznego układu S_i oraz słowa wejściowego X_j generuje kolejny stan SM S_{i+1} . W układach Mealy'ego słowo wyjściowe Z generowane jest przez układ $\mathbf{H}_2$ na podstawie słowa stanu układu i słowa wejściowego (w układach Moore'a tylko na podstawie słowa stanu). W celu redukcji złożoności układu często stosuje się jeden układ kombinacyjny $\mathbf{H}$, wykorzystujący wspólne podukłady układów $\mathbf{H}_1$ oraz $\mathbf{H}_2$, który realizuje funkcję



Rys. 2.1: Schemat układu Mealy'ego

przejść oraz wyjść. Stan SM pamiętany jest w rejestrze (pamięci). Metodą opisu SM stosowaną w tej pracy jest tablica przejść-wyjść.

2.2 Uszkodzenia i błędy w układach cyfrowych

Definicja 2.2 *Uszkodzenie jest to niepoprawny stan linii układu cyfrowego spowodowany defektem jego elementów składowych.*

Najczęściej rozpatrywanym zbiorem uszkodzeń są uszkodzenia typu sklejanie.

Definicja 2.3 *Uszkodzenie typu sklejanie (ang. stuck-at fault) polega na tym, że linia, wejście lub wyjście bramki logicznej przyjmuje stałą wartość 1 (sklejanie z 1, w skrócie s/1) lub 0 (sklejanie z 0, w skrócie s/0), niezależnie od sygnałów podanych na wejście układu.*

Z innych rozpatrywanych podstawowych modeli uszkodzeń należy tutaj wymienić: zmostkowania oraz uszkodzenia opóźnień (ang. delay faults). W pracy tej rozważany jest wyłącznie model uszkodzeń typu pojedyncze sklejanie z 1 lub 0. Wybór tego modelu podyktowany jest z jednej strony dużym zbiorem rzeczywistych defektów dających się modelować jako sklejanie, a z drugiej strony stosunkowo prostą analizą wpływu uszkodzeń na funkcjonowanie układu.

Uszkodzenia występujące w układach cyfrowych możemy podzielić względem czasu trwania na uszkodzenia trwałe oraz przemijające. Uszkodzenia przemijające (nietrwałe), występujące podczas działania układów cyfrowych, powstają wskutek zakłóceń zarówno wewnętrznych jak i zewnętrznych. W przeciwieństwie do uszkodzeń trwałych, które trwale modyfikują funkcję logiczną realizowaną przez układ, uszkodzenia przemijające powodują krótkotrwałe zakłócenia pracy skądinąd sprawnego układu.

Skutkiem uszkodzeń układu cyfrowego są błędy.

Definicja 2.4 *Błąd polega na wystąpieniu niepoprawnego stanu logicznego $\bar{z}$ zamiast, z , co zapisujemy $z \rightarrow \bar{z}$, $z \in \{0, 1\}$, spowodowanego uszkodzeniem układu.*

Układ z uszkodzeniem trwałym wygeneruje te same wyniki dla identycznych warunków początkowych, tym samym uszkodzenia trwałe mogą być wykrywane przez okresowo uruchamiane procedury testujące. Uszkodzenia przemijające są znacznie trudniejsze do wykrycia ze względu na swój nietrwały charakter. Jedną z metod zabezpieczenia układu przed uszkodzeniami przemijającymi jest budowa układu jako samosprawdzalnego.

2.3 Równoważność i dominacja uszkodzeń

Analiza wpływu uszkodzeń na układ może być bardzo złożona obliczeniowo, szczególnie dla układów o bardzo złożonej strukturze. Dla przykładu analiza wpływu uszkodzeń pojedynczych s/z dla układu o n połączeniach wymaga rozpatrzenia $2n$ uszkodzeń, ale w przypadku uszkodzeń wielokrotnych problem ten staje się problemem NP-zupełnym (złożoność 3^n). Liczbę rozpatrywanych uszkodzeń można zredukować wykorzystując własności równoważności i dominacji uszkodzeń.

Definicja 2.5 *Dwa uszkodzenia f oraz g nazywamy równoważnymi, jeżeli dla dowolnego wektora wejściowego X , $H(X, f) = H(X, g)$, co zapisujemy $f \equiv g$, gdzie $H(X, f)$ oznacza funkcję realizowaną przez układ w obecności uszkodzenia f .*

Analizując własność równoważności uszkodzeń dla bramek AND oraz OR otrzymujemy:

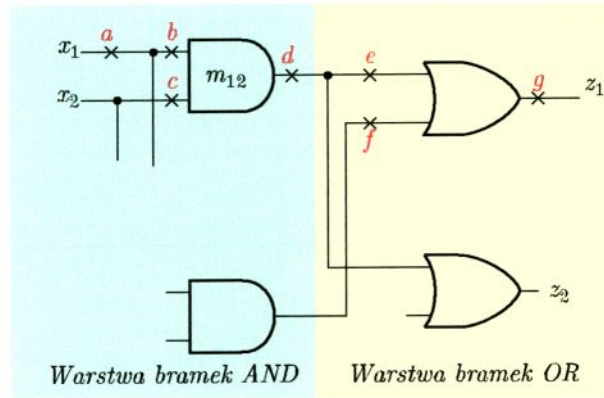
- dla k -wejściowej bramki AND uszkodzenia $s/0$ dowolnej linii wejściowej oraz $s/0$ linii wyjściowej są sobie równoważne (w celu analizy wystarczy rozpatrywać tylko jedno z $k + 1$ uszkodzeń, np.: $s/0$ linii wyjściowej);
- dla k -wejściowej bramki OR uszkodzenia $s/1$ dowolnej linii wejściowej oraz $s/1$ linii wyjściowej są sobie równoważne (w celu analizy wystarczy rozpatrywać tylko jedno z $k + 1$ uszkodzeń, np.: $s/1$ linii wyjściowej).

Definicja 2.6 *Uszkodzenie f jest dominujące względem uszkodzenia g , jeżeli zbiór testów dla uszkodzenia f wykrywa także uszkodzenie g , co możemy zapisać $T_f \supset T_g$, gdzie T_f oznacza zbiór testów dla uszkodzenia f .*

Własność dominacji uszkodzeń również pozwala zredukować zbiór rozpatrywanych uszkodzeń. Dla rozważanych bramek AND oraz OR otrzymamy:

- dla k -wejściowej bramki AND uszkodzenie $s/1$ dowolnej linii wejściowej jest dominujące względem uszkodzenia $s/1$ linii wyjściowej;
- dla k -wejściowej bramki OR uszkodzenie $s/0$ dowolnej linii wejściowej jest dominujące względem uszkodzenia $s/0$ linii wyjściowej.

Własności dominacji i równoważności uszkodzeń pozwalają ograniczyć ich zbiór o połowę (z $2k + 2$ do $k + 1$): dla bramek AND – do uszkodzeń $s/1$ każdej linii wejściowej oraz $s/0$ linii wyjściowej, natomiast dla bramek OR – do uszkodzeń $s/0$ każdej linii wejściowej oraz $s/1$ linii wyjściowej. Zatem, dla układów o strukturze AND-OR (rys. 2.2), należy analizować tylko uszkodzenia $s/0$ oraz $s/1$ we wszystkich punktach oznaczonych od a do g (reprezentatywny zbiór uszkodzeń). Korzystając z własności dominacji i równoważności uszkodzeń problem można uprościć, gdyż wystarczy rozpatrywać jedynie uszkodzenia $s/1$ na wejściach (b, c) oraz $s/0$ na wyjściach (d) bramek AND.



Rys. 2.2: Fragment układu AND-OR z uszkodzeniami.

2.4 Kody nieuporządkowane wykrywające błędy

Kody wykrywające błędy (kody detekcyjne) formuje się przez dodanie nadmiarowych bitów do bitów danych tak, aby nowa reprezentacja danych zawierała nadmiarową informację umożliwiającą detekcję błędów z dobrze zdefiniowanej klasy błędów najbardziej prawdopodobnych (np. wszystkie błędy pojedyncze lub jednokierunkowe). Podstawowe klasy kodów wykrywających błędy to: kody z kontrolą parzystości, kody z podwajaniem i inwersyjnym powtórzeniem, kody arytmetyczne oraz kody nieuporządkowane.

Metody syntezy TSC SM przedstawione w tej pracy, podobnie jak większość znanych z literatury metod syntezy TSC SM (rozdział 1) wykorzystują, z uwagi na ich własności (podrozdział 2.5), kody nieuporządkowane. Dlatego zostanie teraz podany przegląd znanych kodów nieuporządkowanych oraz metod syntezy układów STC dla tych kodów.

Niech X i Y oznaczają dwa wektory n -bitowe.

Definicja 2.7 Relację częściowego uporządkowania wektorów binarnych definiuje się następująco:

$$X \leq Y \text{ wtedy i tylko wtedy, jeżeli } x_i \leq y_i \text{ dla każdego } 1 \leq i \leq n.$$

Np. $(1, 0, 0, 1) \leq (1, 1, 0, 1)$ i $(1, 0, 0, 1) \not\leq (0, 1, 0, 1)$.

Definicja 2.8 Mówimy, że X pokrywa Y wtedy i tylko wtedy, jeżeli X ma 1 na wszystkich bitach, na których Y ma 1, co zapisujemy, $Y \leq X$. Jeżeli ani $Y \leq X$ ani $X \leq Y$, wtedy mówimy, że X i Y są nieuporządkowane, co zapisujemy $X \not\leq Y$.

Definicja 2.9 Zbiór n -bitowych wektorów C nazywamy kodem nieuporządkowanym, jeżeli dla każdego $X, Y \in C$, $X \neq Y$ implikuje $X \not\leq Y$.

Wykazano [90], że kod C wykrywa wszystkie błędy jednokierunkowe wtedy i tylko wtedy, jeżeli jest nieuporządkowany. Należy tutaj zaznaczyć, że kody nieuporządkowane, w naturalny sposób, nadają się do realizacji układów bezinwerterowych, tzn. zrealizowanych bez użycia inwerterów, a więc z użyciem wyłącznie bramek AND i OR (podrozdział 2.5).

Kody z inwersyjnym powtórzeniem

Definicja 2.10 *Kod z inwersyjnym powtórzeniem (ang. double-rail code) opiera się na zasadzie kompletnego zdublowania zanegowanej informacji, w celu sformowania słowa kodowego.*

Kod z inwersyjnym powtórzeniem wprowadza największy nadmiar spośród wszystkich praktycznie stosowanych kodów wykrywających błędy — 100% bitów nadmiarowych. Część kontrolną stanowi $x_{c(i)} = \overline{x_i}$ — powtórzenie z zanegowaniem oryginalnej informacji. Pojemność kodu dana jest wzorem $|C_{podw}(2I)| = 2^I$.

Warto tutaj zaznaczyć, że technologie takie jak ECL czy CMOS DCVS integralnie wykorzystują i generują wszystkie bity zarówno w postaci prostej jak i zanegowanej, które mogą być użyte do wykrywania błędów bez ponoszenia dodatkowych kosztów.

W literaturze podano szereg realizacji układów STC dla kodów z inwersyjnym powtórzeniem. Najszybszym STC dla kodu z inwersyjnym powtórzeniem C_{2K} ($K > 2$) jest 2-poziomowy układ AND-OR — o ile zignorujemy zwiększenie opóźnienia wynikające z dużej liczby wejść i dużej obciążalności bramek oraz ograniczenia technologiczne na liczbę wejść bramek. Układ taki może być użyty tylko pod warunkiem, że kod C_{2K} jest kompletny. STC dla niekompletnego kodu C_{2K} musi być zrealizowany jako układ wielopoziomowy. Najmniej złożony jest układ zbudowany z $K - 1$ modułów STC dla kodu C_4 jako układ kaskadowy (iteracyjny) — o $K - 1$ poziomach modułów, lub w formie drzewa — o $\lceil \log_2 K \rceil$ poziomach modułów, z których każdy wymaga tylko czterech słów kodu C_{2K} jako testów. Uniwersalny STC dla niekompletnego kodu C_{2K} , nadający się np. do budowy STC dla większości kodów systematycznych, został przedstawiony w [47].

Kody ' m z n '

Definicja 2.11 *Kod ' m z n ', oznaczany w skrócie jako kod m/n i nazywany również kodem stałowagowym, jest to taki kod, w którym każde słowo kodowe ma dokładnie m jedynek i $n - m$ zer.*

Zbiór wszystkich słów kodu m/n będziemy oznaczać $C_{m/n}$, a jego pojemność wynosi $|C_{m/n}| = \binom{n}{m} = n! / [(n-m)!m!]$. Kod $\lfloor n/2 \rfloor / n$ jest optymalnym kodem nieuporządkowanym w tym sensie, że dla danej długości słowa kodowego n nie istnieje żaden inny kod nieuporządkowany o większej pojemności. Każdy błąd jednokierunkowy powoduje zmianę wagi słowa kodowego kodu m/n , co można łatwo wykryć sprawdzając wagę otrzymanego słowa. Kody m/n oprócz tego, że wykrywają wszystkie błędy jednokierunkowe, mają zdolność wykrywania wszystkich błędów wielokrotnych różnokierunkowych, które zmieniają wagę wektora.

Szczególnie użyteczne są następujące kody m/n : kody $m/2m$ i kody $m/2m \pm 1$ (optymalne kody nieuporządkowane) stosowane do kodowania stanów wewnętrznych SM oraz kody $1/n$ — które występują w wielu układach cyfrowych, niekoniecznie wyłącznie w celu detekcji błędów, tak jak np. na wyjściu dekodera, wewnątrz pamięci ROM, w zwykłych SM zbudowanych z użyciem programowalnych układów logicznych (PLD) (ang. one-hot code), etc.

W zagadnieniach syntezy TSC SM, m/n mają głównie zastosowanie do kodowania stanów wewnętrznych (kody $1/n$, $2/n$), oraz wyjść układów (kody $1/n$, $2/n$ oraz sporadycznie inne kody m/n). Dlatego przegląd metod syntezy układów STC dla kodów m/n ograniczony zostanie tylko do układów STC dla kodów $1/n$ i $2/n$. Kompletny przegląd metod syntezy układów STC dla kodów m/n można znaleźć w [74] i [12].

Metody syntezy układów STC dla kodów $2/n$, gdzie $n \geq 5$ zostały omówione w pracach [2], [47], [69] oraz [99], przy czym układy o najmniejszej złożoności można zaprojektować zgodnie z metodą z [69]. Zbiór testów dla układów zaprojektowanych metodą z [69] składa się z n słów, zawierających 2 jedynki poprzedzone $n-2$ zerami, przesunięte cyklicznie względem siebie o jeden bit. Szczególnym STC jest układ podany w [77] dla kodu stanowiącego konkatenację kodów $1/2$ oraz $1/3$ (niekompletnego kodu $2/5$). Można go wykorzystać w przypadku, gdy w układzie występuje kod $1/3$ (dla tego kodu nie istnieje kombinacyjny STC). Warunkiem zastosowania jest możliwość konkatenacji kodu $1/3$ z kodem $1/2$ oraz występowanie wszystkich 6 słów kodowych.

W literaturze podano szereg metod syntezy STC dla kodów $1/n$ [2], [38], [48], [54], [68], [78], [79], [80]. Układ o najmniejszej złożoności dla $n = 4$ został przedstawiony w [79], natomiast dla $n = 5$ i $n = 6$ przez Andersona i Metze'a w [2]. Metody syntezy STC o najmniejszej złożoności dla $n > 6$ przedstawiają prace [67], [68]. W zaproponowanym rozwiązaniu najpierw dokonuje się translacji kodu $1/n$ na niekompletny kod $2/n$, a następnie stosuje układ STC dla kodu $2/n$. Zbiór testów układu STC dla kodu $1/n$ składa się z wszystkich słów kodu $1/n$. Przegląd metod syntezy układów STC dla kodów $1/n$ można znaleźć w pracy [74]. W tabelach 2.1 2.2 podano charakterystyki STC dla stosowanych w tej pracy kodów odpowiednio $2/n$ oraz $1/n$ (złożoność układów podana jest w literalach i została obliczona z wykorzystaniem oprogramowania SIS).

Kody Bergera

Kody Bergera są to optymalne systematyczne kody nieuporządkowane — mają minimalną liczbę bitów kontrolnych K dla danej długości części informacyjnej I .

Definicja 2.12 Kod Bergera $C_{(I,K)}$ jest to kod systematyczny, w którym K bitów kontrolnych jest binarną reprezentacją liczby bitów informacyjnych, w których występują zera, $K = \lceil \log_2(I + 1) \rceil$. (Alternatywna definicja równoważnego kodu Bergera definiuje część kontrolną jako negację binarnego zapisu liczby jedynek w części informacyjnej).

Definicja 2.13 Kod Bergera o maksymalnej długości $C_{MLB(K)}$ to taki, w którym $K = \log_2(I + 1)$, tzn. $I = 2^K - 1$.

Kody systematyczne, w których liczba bitów informacyjnych I jest potęgą 2, są szczególnie ważne dla zastosowań praktycznych. Ponieważ, do niedawna [76] nie był znany STC dla kodu Bergera $C_{(2^K-1,K)}$, skonstruowano kod równoważny, dla którego łatwo zaprojektować 2-wyjściowy STC.

Definicja 2.14 Kod C o długości słowa kodowego n równoważny kodowi Bergera $C_{(I,K)}$ jest to kod systematyczny, nieuporządkowany, w którym $n = I + K$ i który ma taką samą liczbę słów kodowych co $C_{(I,K)}$.

Definicja 2.15 Konkatenacja kodów C_1 i C_2 o długości słów kodowych n_1 i n_2 jest to kod $C = C_1 \times C_2$ o długości słowa kodowego $X = (X_1 X_2)$ równej $n = n_1 + n_2$ utworzony przez złożenie wszystkich słów kodowych $X_1 \in C_1$ z każdym słowem kodowym $X_2 \in C_2$; zatem pojemność kodu C jest $|C| = |C_1| \cdot |C_2|$.

Definicja 2.16 Zmodyfikowany kod Bergera $C_{(2^K-1,K)}^*$ definiuje się jako

$$C_{(2^K-1,K)}^* = C_{(2^{K-1}-1,K-1)} \times C_{1/2}, \quad (2.1)$$

Tablica 2.1: Charakterystyka STC dla wykorzystywanych kodów $2/n$.

Kod	$ C_{2/n} $	Liczba testów STC	Złożoność STC
2/3	3	-	*
2/4	6	4	10
2/5	10	5	21
2/6	15	6	22
2/7	21	7	32
2/8	28	8	34
2/9	36	9	46
2/10	45	10	47
2/11	55	11	53
2/12	66	12	57
2/13	78	13	59
2/14	91	14	60
2/15	105	15	66

* - STC istnieje wyłącznie dla konkatencji kodu $1/3$ z innym kodem nieuporządkowanym.

tj. kod $C_{(I,K)}^*$ tworzy się przez konkatencję maksymalnego kodu Bergera $C_{(2^{K-1}-1, K-1)}$ z kodem $1/2$ w taki sposób, że:

1. dwa bity słowa kodu $1/2$ tworzą najbardziej znaczący bit części informacyjnej x_{I-1} i najmniej znaczący bit części kontrolnej x_{c0} kodu $C_{(I,K)}^*$;
2. pozostałe $I + K - 2$ bitów $\{x_{I-1}, \dots, x_2, x_1, x_{c(K-2)}, \dots, x_{c0}\}$ tworzy słowa maksymalnego kodu Bergera $C_{(2^{K-1}-1, K-1)}$.

Metody syntezy układów STC dla kodów Bergera oraz kodów równoważnych zostały przedstawione w [3], [4] [47], [71], [72], [73], [74], [76]. Układy z [4], [47], oraz [72] są zrealizowane jako klasyczne układy kontrolne dla kodów systematycznych. Z drugiej strony prace [3], [71] dotyczą układów kontrolnych dla szczególnych kodów: [3] — $I = 2^K - 1$, [71] — $I = \{2^K - 2, 2^K - 1\}$. Układy STC z [3] oraz [47] wykorzystują nieoptymalne, jak wykazano w [74], liczniki jedynek, zbudowane z FA i HA (pełne sumatory i półsumatory). W [74] podano metodę syntezy układów STC wykorzystujących optymalne liczniki jedynek zbudowane z FA i HA. W pracy [72] zaproponowano inną strukturę STC dla kodów Bergera wykorzystującą liczniki jedynek zbudowane z układów progowych T^i . Trudności związane z zapewnieniem ST układów STC dla najważniejszej (z praktycznego punktu widzenia) klasy kodów Bergera z $I = 2^{K-1}$ są powodem, dla którego stosuje się równoważne im zmodyfikowane kody Bergera. Zaproponowany w [76] STC dla kodów Bergera ma większą złożoność niż STC dla odpowiadającego mu zmodyfikowanego kodu Bergera, dlatego też w praktycznych zastosowaniach wykorzystywany jest zmodyfikowany kod Bergera. W tabeli 2.3 podano charakterystyki stosowanych w tej pracy kodów Bergera oraz zmodyfikowanych kodów Bergera.

Tablica 2.2: Charakterystyka STC dla wykorzystywanych kodów $1/n$.

Kod	Złożoność STC
1/3	*
1/4	16
1/5	22
1/6	24
1/7	39
1/8	43
1/9	47
1/10	51
1/11	54
1/12	58
1/13	62
1/14	66
1/15	70
1/16	82
1/17	86
1/18	90
1/19	94
1/20	98
1/21	102
1/22	106
1/23	110
1/24	114
1/25	118

* - STC istnieje wyłącznie dla konkatenacji kodu 1/3 z innym kodem nieuporządkowanym.

Tablica 2.3: Charakterystyka STC wykorzystywanych kodów Bergera.

Kod	$ C_{(I,K)} $	Liczba testów STC		Złożoność STC	
		[74]	[72]	[74]	[72]
$C_{(3,2)}$	8	8	6	38	29
$C_{(4,3)}^*$	16	8	6	50	41
$C_{(5,3)}$	32	12	8	61	69
$C_{(6,3)}$	64	12	9	71	85
$C_{(7,3)}$	128	8	11	81	101
$C_{(8,4)}^*$	256	8	11	93	113
$C_{(9,4)}$	512	12	12	110	

2.5 Własności układów wykorzystujących kody nieuporządkowane

Rozważmy układ kombinacyjny $\mathbf{H}$, którego wejścia oraz wyjścia zakodowane są kodami nieuporządkowanymi.

Definicja 2.17 *Mówimy, że implikant $m_j = x_{j_1}x_{j_2}...x_{j_m}$ odpowiada słowu X_j , co zapisujemy w postaci $m_j \leftrightarrow X_j$, jeżeli na wszystkich pozycjach $j_l, l \in \{1, m\}$ słowa X_j występują jedynki, a na pozostałych zera.*

Dla przykładu słowu $X_k = (110100)$ kodu 3/6 odpowiada implikant $m_k = x_1x_2x_4$.

Funkcja z_j układu kombinacyjnego $\mathbf{H}$ o wejściach oraz wyjściach zakodowanych kodami nieuporządkowanymi może być wyrażona [97] jako

$$z_j = \sum_{m_i \in M_j} m_i, \quad j = \{1, 2, \dots, s\}, \quad (2.2)$$

gdzie:

$\sum$ – logiczna operacja OR,

M_j – zbiór implikantów m_i , które odpowiadają wektorom wejściowym $X_i \in C_{IN}$ dla, których $z_j = 1$,

s – liczba wyjść układu.

Niech $m_i(x^*)$ oznacza zbiór bitów x_i , które występują w m_i . Dla danego $m_i = x_{i_1}x_{i_2}...x_{i_a}$ oraz $m_j = x_{j_1}x_{j_2}...x_{j_b}$, zachodzi relacja $m_i \subset m_j$ wtedy i tylko wtedy gdy

$$(\forall x_{i_r} \subset m_i(x^*))(\exists x_{j_r} \subset m_j(x^*) || x_{i_r} = x_{j_r}). \quad (2.3)$$

Przykładowo dla implikantów $m_i = x_4x_5$, $m_j = x_2x_3x_4x_5x_6$ oraz $m_k = x_3x_5$ otrzymujemy $m_i \subset m_j$ oraz $m_i \not\subset m_k$.

Definicja 2.18 *Dzielnikiem implikantu m_i ze względu na zmienną x_k jest $m_j(x_k)$, który otrzymujemy poprzez podstawienie $x_k = 1$ w implikancie m_j .*

Przykładowo dla $m_j = x_2x_3x_4x_5x_6$ otrzymujemy $m_j(x_3) = x_2x_4x_5x_6$.

Definicja 2.19 *Funkcja boolowska $H(X)$ jest monotonicznie rosnąca wtedy i tylko wtedy, jeżeli jest spełniony warunek*

$$(X_1 \leq X_2) \Rightarrow (H(X_1) \leq H(X_2)),$$

w szczególności dla poszczególnych z_j spełniony jest warunek

$$(X_1 \leq X_2) \Rightarrow (z_j(X_1) \leq z_j(X_2)).$$

Szczególnym przypadkiem funkcji monotonicznie rosnącej jest funkcja pozytywnie gładka [17].

Definicja 2.20 *Funkcja pozytywnie gładka jest to funkcja, w której w postaci normalnej występują tylko zmienne proste.*

Definicja 2.21 Układ bezinwerterowy jest to układ, który składa się tylko z bramek AND oraz OR.

Układ kombinacyjny $\mathbf{H}$, którego wejścia oraz wyjścia zakodowane są kodami nieuporządkowanymi, odpowiednio C_{IN} oraz C_{OUT} , charakteryzuje się następującymi własnościami:

- (W1) Funkcja realizowana przez układ $\mathbf{H}$ jest monotoniczna.
- (W2) Układ $\mathbf{H}$ zawsze można zrealizować jako układ bezinwerterowy.
- (W3) Jeżeli układ $\mathbf{H}$ realizuje funkcję pozytywnie gładką, to da się zrealizować jako układ bezinwerterowy [37].
- (W4) Dla słów niekodowych X^e (X^e powstaje z X na skutek błędów jednokierunkowych) prawdziwe są następujące zależności [90]:

$$\begin{aligned} (\forall X^e \leq X)(H(X^e) \leq H(X)) \\ (\forall X^e \geq X)(H(X^e) \geq H(X)). \end{aligned}$$

- (W5) Jeżeli układ $\mathbf{H}$ jest bezinwerterowy, to dla pojedynczych uszkodzeń wewnętrznych $f_z \in F$ (F oznacza zbiór pojedynczych uszkodzeń f typu s/z , $z \in \{0, 1\}$), prawdziwe są następujące zależności [90]:

$$\begin{aligned} (\forall X \in C_{IN})(\forall f_0 \in F)(H(X, f_0) \leq H(X)) \\ (\forall X \in C_{IN})(\forall f_1 \in F)(H(X, f_1) \geq H(X)). \end{aligned}$$

- (W6) Jeżeli układ jest bezinwerterowy, to jest odporny na wszystkie uszkodzenia pojedyncze typu s/z , $z \in \{0, 1\}$ [90].
- (W7) Jeżeli układ jest ST dla uszkodzeń pojedynczych ze zbioru F , to jest także ST dla jednokierunkowych uszkodzeń wielokrotnych [90].
- (W8) Jeżeli układ jest CD dla wszystkich błędów pojedynczych, to jest także CD dla wszystkich wielokrotnych błędów jednokierunkowych [90].

Z własności (W5) wynika

$$(\forall X \in C_{IN})(\forall f_z \in F, z \in \{0, 1\})(H(X, f_0) \leq H(X) \leq H(X, f_1)).$$

Definicja 2.22 Wielowejsciowy układ kombinacyjny $\mathbf{H}$ jest układem o rozłącznych podukładach (ang. bit-sliced circuit), jeżeli każdy podukład realizujący funkcję z_i ma wspólne z innymi podukładami tylko linie wejściowe pierwotne. W przeciwnym razie układ jest układem o wspólnych podukładach (ang. shared logic).

Jedną z podstawowych trudności syntezy układów samosprawdzalnych jest możliwość wystąpienia symetrycznych błędów wielokrotnych na wyjściu układu, spowodowanych uszkodzeniami pojedynczymi. Bez wprowadzenia ograniczeń na strukturę wewnętrzną w układach o wspólnych podukładach

błąd taki może wystąpić, jeżeli liczba inwersji modulo 2 w ścieżkach propagacji błędu na różne wyjścia jest różna. Błędy tego typu mogą nie być wykrywane przez stosowane kody detekcyjne (kody nieuporządkowane, kody parzystości). Rozwiązaniem tego problemu może być zastosowanie układów o rozłącznych podukładach, w których dowolne uszkodzenie pojedyncze może spowodować błąd na jednym i tylko jednym wyjściu. Podejście takie znalazło zastosowanie w metodach syntezy układów samosprawdzalnych, wykorzystujących kody parzystości. Zastosowanie układów o rozłącznych podukładach często wiąże się ze wzrostem złożoności struktury układów, co powoduje nieopłacalność ich stosowania.

Własności układów kombinacyjnych wykorzystujących kody nieuporządkowane umożliwiają ich syntezę jako układów o wspólnych podukładach. Implementacja układów jako bezinwerterowych zapewnia, że pojedyncze uszkodzenie może spowodować na wyjściu układu błąd pojedynczy lub wielokrotny błąd jednokierunkowy, wykrywany przez kody nieuporządkowane. Zastosowanie kodów nieuporządkowanych i układów o wspólnych podukładach niesie ze sobą także szereg innych korzyści:

- łatwość testowania układów wykorzystujących kody nieuporządkowane [2], [90],
- łatwość podania funkcji układu w postaci częściowo zminimalizowanej, co ma szczególne znaczenie dla zagadnień syntezy i analizy własności układów o dużej złożoności.

Z powyższych powodów, w pracy tej rozważane będą tylko układy o wejściach oraz wyjściach zakodowanych kodami nieuporządkowanymi, wykorzystujące wspólne podukłady. W większości technologii CMOS układy te są implementowane z wykorzystaniem elementów NAND oraz NOR. Zasady transformacji układów beziwerterowych na implementację NAND (NOR), zachowujące ST podał Smith w [90].

2.6 Układy samosprawdzalne

2.6.1 Podstawowe definicje

Układ **H** z uszkodzeniem wewnętrznym f może generować trzy typy wyjść:

1. $H(X, f) = H(X)$ —poprawne wyjście jest generowane dla X , niezależnie od obecności uszkodzenia f ; mówimy, że uszkodzenie f jest *maskowane dla wejścia X* .
2. $H(X, f) \neq H(X)$ i $H(X, f) \notin C_{OUT}$ —niepoprawne wykrywalne wyjście (wyjściowe słowo niekodowe) jest generowane dla X wskutek uszkodzenia f ; mówimy, że uszkodzenie f jest *wykrywane przez X* .
3. $H(X, f) \neq H(X)$ i $H(X, f) \in C_{OUT}$ —niepoprawne wyjściowe słowo kodowe jest generowane dla X wskutek uszkodzenia f ; mówimy, że uszkodzenie f *powoduje niewykrywalny błąd na wyjściu* w tym sensie, że nie można wykryć tego błędu przez zwykłe sprawdzenie przynależności $H(X, f)$ do C_{OUT} .

Układ **H** jest TSC wyłącznie wtedy, gdy pierwsze dwa przypadki występują dla każdego uszkodzenia z pewnego dobrze zdefiniowanego zbioru uszkodzeń F . W pracy tej przyjęto, że F jest to zbiór

wszystkich uszkodzeń typu s/z , $z \in \{0, 1\}$. Jednakże istnieje szereg klas układów SC, które pozwalają rozszerzyć zbiór F o różne klasy uszkodzeń wielokrotnych.

Podamy teraz formalne definicje szeregu klas układów, które pozwalają osiągnąć cel TSC, w zależności od zachowania uszkodzonego układu.

Definicja 2.23 [2] *Układ H jest odporny na uszkodzenia (ang. fault-secure (FS)) ze zbioru uszkodzeń F , jeżeli dla każdego uszkodzenia f ze zbioru F , układ H , albo mimo uszkodzenia, generuje poprawne wyjściowe słowo kodowe albo, wskutek uszkodzenia generuje wyjściowe słowo niekodowe, które jest wykrywalne, co zapisujemy formalnie*

$$(\forall f \in F) (\forall X \in C_{IN}) (H(X, f) = H(X)) \text{ lub } (H(X, f) \notin C_{OUT}).$$

Odporność na uszkodzenia gwarantuje jedynie, że dla żadnego uszkodzenia f ze zbioru F układ FS, na którego wejście podano słowo kodowe, nie generuje niewykrywalnego błędu na wyjściu. Zatem, aby niewykryte uszkodzenia nie gromadziły się wewnątrz układu, co mogłoby doprowadzić do utraty odporności na uszkodzenia wskutek obecności uszkodzeń wielokrotnych, które mogą być spoza zbioru F , układ ten powinien mieć następującą własność.

Definicja 2.24 [11] *Układ H jest samotestowalny (ang. self-testing (ST)) dla uszkodzeń ze zbioru F , jeżeli dla każdego uszkodzenia f ze zbioru uszkodzeń F układ H generuje wyjściowe słowo niekodowe dla co najmniej jednego wejściowego słowa kodowego, co zapisujemy formalnie*

$$(\forall f \in F) (\exists X \in C_{IN} \mid H(X, f) \notin C_{OUT}).$$

Własność ST zapewnia, że każde uszkodzenie ze zbioru F zostanie wykryte w czasie normalnej pracy układu H , tzn. bez błędów na wejściu układu H , i wyłącznie przez wejściowe słowa kodowe ze zbioru C_{IN} , odciośnie których zakłada się, że wszystkie występują podczas normalnej pracy układu.

Definicja 2.25 [2] *Układ H całkowicie samosprawdzalny (ang. totally self-checking (TSC)) dla zbioru uszkodzeń F jest to układ, który jest zarówno FS jak i ST dla F .*

Układ TSC ma, zatem, dwie własności:

- pierwszy błąd, który wystąpi na wyjściu układu H , jest słowem niekodowym,
- własność ST zabezpiecza układ przed utratą własności FS wskutek gromadzenia się nie wykrytych uszkodzeń ze zbioru F .

Funkcjonalny układ TSC ma zagwarantowane poprawne działanie w obecności uszkodzeń wewnętrznych ze zbioru F , pod warunkiem, że każde uszkodzenie f zostanie wykryte, zanim wystąpi następne uszkodzenie. Podane definicje układów TSC nie gwarantują jednak poprawnej pracy układu w obecności błędów, które mogą pojawić się na wejściu układu.

Niech X^e oznacza wejściowe słowo niekodowe powstałe w wyniku błędów w słowie kodowym X . Układ H z błędem X^e na wejściu może generować trzy typy wyjść:

1. $H(X^e) = H(X)$ —poprawne wyjście jest generowane dla X , niezależnie od błędu na wejściu; mówimy, że układ $\mathbf{H}$ maskuje błąd na wejściu lub, że błąd na wejściu jest korygowany przez układ $\mathbf{H}$.
2. $H(X^e) \neq H(X)$ i $H(X^e) \notin C_{OUT}$ —niepoprawne wykrywalne wyjście (wyjściowe słowo niekodowe) jest generowane dla X^e ; mówimy, że błąd wejściowy jest wykrywany (propagowany na wyjście) przez układ $\mathbf{H}$.
3. $H(X^e) \neq H(X)$ i $H(X^e) \in C_{OUT}$ —niepoprawne wyjściowe słowo kodowe jest generowane dla X^e ; mówimy, że X^e powoduje niewykrywalny błąd na wyjściu układu $\mathbf{H}$.

Ocena poprawności funkcjonowania układu $\mathbf{H}$ z błędami na wejściu zależy od funkcji realizowanej przez $\mathbf{H}$. Jeżeli $\mathbf{H}$ jest układem funkcjonalnym, to pierwsze dwa przypadki są dopuszczalne, gdyż tylko wtedy układ nie generuje niewykrywalnego błędu. Jednakże, jeżeli $\mathbf{H}$ jest układem kontrolnym, to wtedy tylko przypadek 2 jest dopuszczalny, gdyż $\mathbf{H}$ powinien sygnalizować każdorazowe wystąpienie słowa niekodowego generowanego przez układ funkcjonalny, a pojawiającego się na wejściu $\mathbf{H}$, poprzez wygenerowanie wyjściowego słowa niekodowego. W przeciwnym razie niewykryte uszkodzenia gromadzące się wewnątrz układu funkcjonalnego monitorowanego przez $\mathbf{H}$ mogłyby spowodować utratę samosprawdzalności dla uszkodzeń ze zbioru F .

Podobnie jak dla uszkodzeń wewnętrznych, podamy teraz formalne definicje klas układów, które pozwalają osiągnąć cel TSC, w zależności od zachowania układu z błędami na wejściu.

Definicja 2.26 [2] Układ $\mathbf{H}$ jest kodowo-rozłączny (ang. code-disjoint (CD)), jeżeli odwzorowuje każde wejściowe słowo kodowe w wyjściowe słowo kodowe i każde wejściowe słowo niekodowe w wyjściowe słowo niekodowe, co zapisujemy formalnie

$$(\forall X \in C_{IN}) (H(X) \in C_{OUT}) \text{ i } (\forall X^e \notin C_{IN}) (H(X^e) \notin C_{OUT}).$$

Niech C_{IN}^e oznacza zbiór słów niekodowych, które mogą wystąpić na wejściu układu $\mathbf{H}$ wskutek wystąpienia najbardziej prawdopodobnych uszkodzeń w układach, które dostarczają sygnały wejściowe do $\mathbf{H}$.

Definicja 2.27 [53] Układ $\mathbf{H}$ jest bezpieczny dla błędów wejściowych (ang. error-secure (ES)) ze zbioru C_{IN}^e , jeżeli:

1. $\mathbf{H}$ odwzorowuje każde wejściowe słowo kodowe w wyjściowe słowo kodowe,
2. $\mathbf{H}$ odwzorowuje każde wejściowe słowo niekodowe, albo w wyjściowe słowo niekodowe albo generuje poprawne wyjściowe słowo kodowe (takie, które wystąpiłoby, gdyby nie było błędu na wejściu).

Własność ES jest pożyteczna w układach funkcjonalnych mających oprócz własności TSC również własność CD, gdyż pozwala ona nieco złagodzić wymogi CD i mimo to jednocześnie zapewnić poprawne funkcjonowanie układu. Oczywiście, własność ES jest niedopuszczalna w układach kontrolnych, które straciłyby zdolność sygnalizowania błędów wejściowych — co jest ich zadaniem.

Definicja 2.28 [2] *Samotestowalny układ kontrolny dla kodu C (ang. self-testing checker (STC)) jest to układ kodowo-rozłączny samotestowalny dla zbioru uszkodzeń ze zbioru F .*

Wystąpienie wyjściowego słowa niekodowego stanowi sygnał alarmowy, zarówno w razie wystąpienia wejścia niekodowego, jak i w razie wykrycia uszkodzenia wewnątrz samego STC. Powszechnie przyjmuje się, że STC jest układem 2-wyjściowym takim, że wyjściowe słowa kodu $1/2 \{01, 10\}$ oznaczają poprawną pracę systemu, a $\{00, 11\}$ wykrycie błędu na wejściu STC lub jego uszkodzenie wewnętrzne.

2.6.2 Założenia dotyczące układów samosprawdzalnych

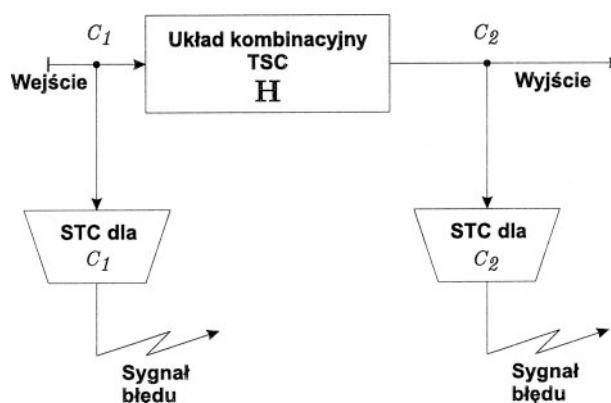
Rozpatrując działanie układów samosprawdzalnych zazwyczaj przyjmuje się następujące założenia, wynikające z konieczności ograniczenia złożoności analizy funkcjonowania układów z uszkodzeniami wewnętrznymi i błędami wejściowymi.

- (A1) Występują tylko uszkodzenia ze zbioru F .
- (A2) Uszkodzenia występują pojedynczo.
- (A3) Po wystąpieniu uszkodzenia w układzie funkcjonalnym lub kontrolnym upływa wystarczająca ilość czasu, aby wystąpiły wejściowe słowa kodowe wystarczające do wykrycia wszystkich uszkodzeń ze zbioru F , zanim wystąpi następne uszkodzenie w układzie.
- (A4) Uszkodzenia i błędy wejściowe nie występują równocześnie.
- (A5) W SM uszkodzenia nie występują na liniach zegarowych.

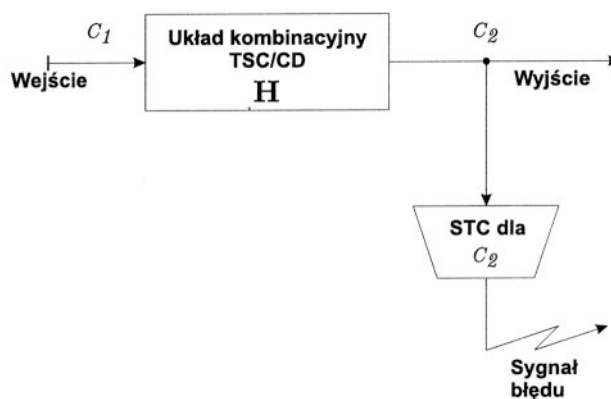
2.6.3 Podstawowe struktury systemów samosprawdzalnych

Podstawowe struktury systemów samosprawdzalnych przedstawiono na rysunkach 2.3 i 2.4. Struktura I (TSC/STC) z rys. 2.3 składa się z jednego TSC bloku funkcjonalnego i dwóch STC — dla kodu wejściowego C_1 i kodu wyjściowego C_2 . Struktura II (TSC/CD) z rys. 2.4 stanowi alternatywę dla struktury I z rys. 2.3 w przypadku, gdy koszt realizacji układu funkcjonalnego, który jest równocześnie TSC i CD, jest mniejszy niż łączny koszt realizacji TSC układu funkcjonalnego oraz STC dla kodu C_1 . Pewną wadą Struktury II jest mniej dokładna informacja o miejscu wystąpienia uszkodzenia, dostarczana przez jedyny wyjściowy STC dla kodu C_2 . Ponadto synteza układu funkcjonalnego, który jest nie tylko TSC, ale i CD, na ogół napotyka na duże trudności.

W pracy tej rozważane są dwie struktury samosprawdzalnych SM TSC/CD (rys. 2.5) oraz TSC/STC (rys. 2.6) (inne znane z literatury modele TSC SM omówiliśmy w rozdziale 1). W celu osiągnięcia celu TSC SM, układ musi być ST i FS dla uszkodzeń ze zbioru F oraz CD dla błędów wejściowych. Istnieją dwie metody spełnienia tego warunku. Pierwsza polega na syntezie układów H_1 i H_2 jako ST, FS oraz CD (rys. 2.5), gdy w efekcie otrzymujemy układ TSC/CD. W szeregu zastosowań można zrezygnować z CD na rzecz ES, w wyniku czego uzyskuje się strukturę TSC/ES, oferującą podobny stopień zabezpieczenia przed błędami wejściowymi (błędy są wykrywane lub korygowane), oraz taki sam stopień zabezpieczenia przed uszkodzeniami wewnętrznymi jak dla struktury TSC/CD. Spełnienie warunków CD to zagadnienie trudne i w literaturze nie są znane metody wystarczająco

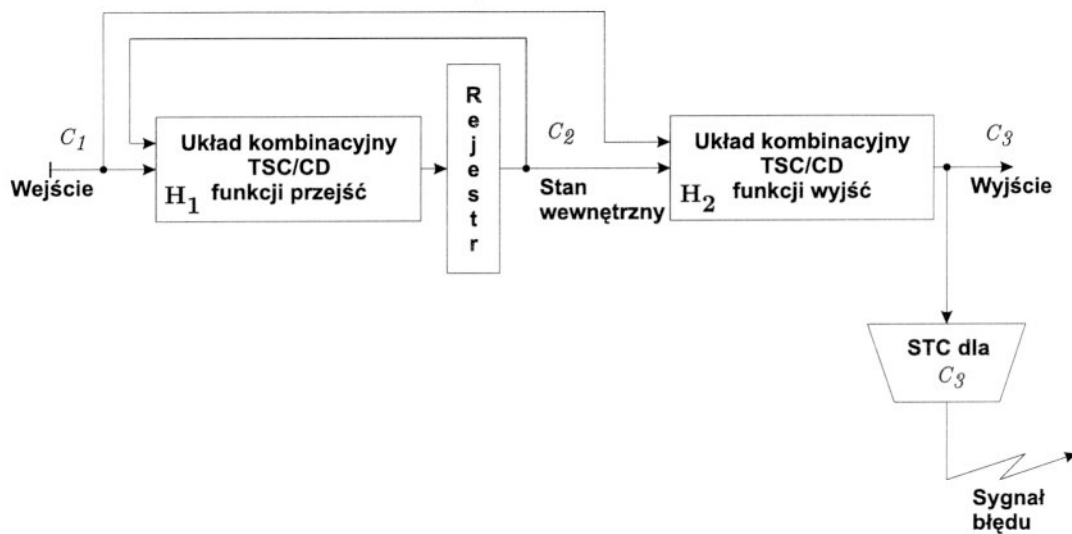


Rys. 2.3: Kombinacyjny układ TSC: Struktura I TSC/STC

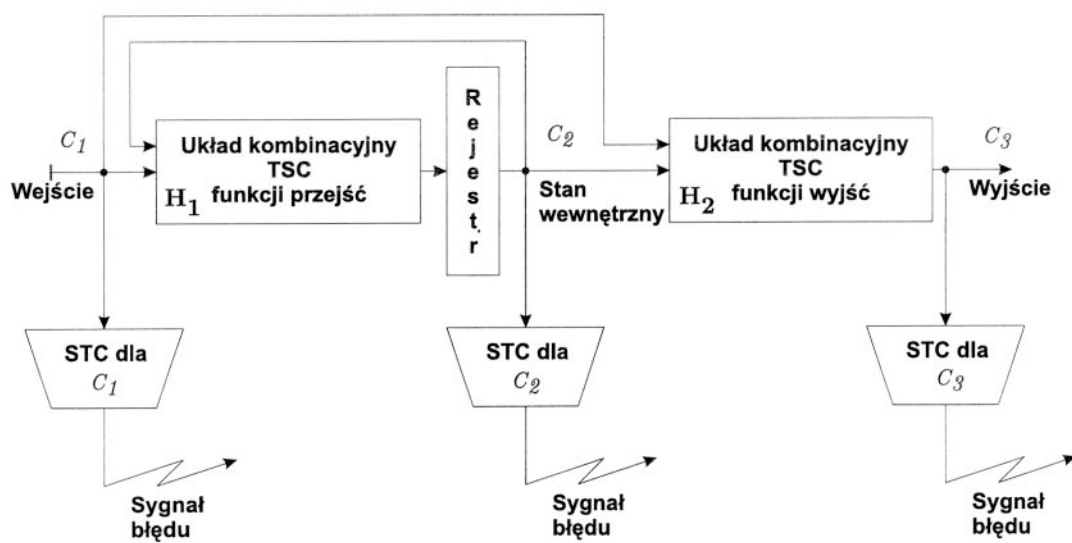


Rys. 2.4: Kombinacyjny układ TSC: Struktura II TSC/CD

ogólne do syntezy szerokiej klasy układów. Stąd też powszechność stosowania drugiego podejścia, polegającego na zastosowaniu układów STC do monitorowania wejść, stanów wewnętrznych oraz wyjść (układ TSC/STC) przy zapewnieniu ST i FS układów kombinacyjnych H_1 i H_2 . Niekodowe pobudzenie lub odpowiedź zostaną wykryte przez układ STC. W rozwiązaniu tym konieczne jest zapewnienie istnienia układów STC dla poszczególnych kodów oraz ich ST w przypadku kodów niekompletnych.



Rys. 2.5: Schemat SM TSC/CD



Rys. 2.6: Schemat SM TSC/STC

Rozdział 3

Synteza TSC układów kombinacyjnych

3.1 Synteza TSC/CD układów kombinacyjnych

W TYM rozdziale zostaną przedstawione metody syntezy klasy układów kombinacyjnych TSC o strukturze TSC/CD, których wejścia i wyjścia są zakodowane kodami nieuporządkowanymi. Podane zostaną warunki kodowej rozłączności (CD) i samotestowalności (ST) układów o strukturze AND-OR oraz przedstawiona zostanie metoda modyfikacji struktury układu AND-OR, która może prowadzić do poprawy ST układów.

3.1.1 Weryfikacja kodowej rozłączności (CD)

W celu syntezy układów TSC o strukturze TSC/CD konieczne jest zapewnienie zarówno FS, ST jak i CD układu, co wymaga przedstawienia warunków weryfikacji własności FS, ST i CD układów kombinacyjnych o wejściach oraz wyjściach zakodowanych kodami nieuporządkowanymi. Jak zostało podane w rozdziale 2.5, układy te są FS dla wszystkich uszkodzeń pojedynczych typu s/z , natomiast warunek ST zostanie omówiony w podrozdziale 3.1.2.

Warunki konieczne i wystarczające CD

Warunki weryfikacji własności CD układów wykorzystujących kody nieuporządkowane zostały podane po raz pierwszy w pracy [70]. Autor ograniczył jednak rozważania do układów wykorzystujących kody m/n . Na tej podstawie opracowano uogólnione warunki weryfikacji własności CD układów wykorzystujących dowolne kody nieuporządkowane.

Twierdzenie 3.1 *Jeżeli dla układu kombinacyjnego H realizującego funkcję (2.2) i wykorzystującego kompletne iloczyny spełniona jest zależność:*

$$(\forall X_k^{e+} \in C_{IN}^{e+}) (\forall m_k^{e+} \leftrightarrow X_k^{e+}) (\exists X_l, X_l \in C_{IN}, X_l \leftrightarrow m_l \mid m_k^{e+} \subset m_l \text{ i } z^*(m_l) \neq z^*(m_k)),$$

to układ H jest kodowo-rozłączny dla błędów $0 \rightarrow 1$.

Dowód: Rozważmy błąd $0 \rightarrow 1$ w bicie x_i wejściowego słowa kodowego $X_k \leftrightarrow m_k$, dla którego $m_k = 0$ oraz $m_l = 0$. Dla niekodowego słowa wejściowego X_k^{e+} zachodzi

$$X_k^{e+} \geq X_l \text{ oraz } X_k^{e+} \geq X_k,$$

skąd otrzymujemy $m_l = 1$ (błąd $0 \rightarrow 1$) i $m_k = 1$. Ponieważ $z^*(m_k) \neq z^*(m_l)$, błąd przepropaguje się na wyjście układu, a zatem układ jest CD. ■

Jeżeli $z^*(m_k) = z^*(m_l)$, to układ jest bezpieczny dla błędów wejściowych (ES), ponieważ maskuje błąd.

Twierdzenie 3.2 *Układ kombinacyjny H , który realizuje funkcję (2.2) i wykorzystuje kompletne iloczyny, jest zawsze kodowo-rozłączny dla błędów $1 \rightarrow 0$.*

Dowód: Rozważmy błąd $1 \rightarrow 0$ w dowolnym wejściowym słowie kodowym X_i . Dla X_i tylko odpowiadający mu implikant m_i ma wartość 1. Dla niekodowego słowa X_i^e , które powstało na skutek błędu $1 \rightarrow 0$ z X_i , zachodzi $X_i^e < X_i$, a zatem iloczyn $m_i = 0$ zamiast 1. Tym samym, błąd $1 \rightarrow 0$ w wejściowym słowie kodowym X_i propaguje się na wyjście układu. ■

Jeżeli nie są spełnione warunki obydwu twierdzeń 3.1 i 3.2, to układ nie jest CD dla błędów na wejściu.

Algorytm weryfikacji CD

Formalne kryteria weryfikacji własności CD pozwalają na opracowanie algorytmu umożliwiającego jej automatyczną weryfikację.

Układ kombinacyjny, opisany w formie tablicy prawdy, zapisany jest jako dwie tablice: **In** zawierającej słowa wejściowe ze zbioru C_{IN} , oraz **Out**— zawierającej opowiadające im słowa wyjściowe ze zbioru C_{OUT} . Dla potrzeb weryfikacji CD konieczne jest zdefiniowanie tablic **InError** i **COut**, reprezentujących odpowiednio zbiór niekodowych słów wejściowych powstałych na skutek błędów pojedynczych oraz kod wyjściowy C_{OUT} .

Algorytm weryfikacji własności CD układu składa się z trzech etapów:

1. Generowanie listy wyjściowych słów kodowych.
2. Generowanie listy niekodowych słów wejściowych.
3. Weryfikacja warunków tw. 3.1 dla niekodowych słów wejściowych.

W pierwszym etapie generowana jest tablica **COut**, w której zapisane są wszystkie kodowe słowa wyjściowe. Następnie (etap 2) generowana jest tablica **InError** zawierająca wszystkie niekodowe słowa wejściowe, które powstają na skutek pojedynczych błędów w słowach kodu C_{IN} . W etapie 3, dla elementów **InError** weryfikowane są warunki tw. 3.1. W wyniku działania algorytmu otrzymujemy tablicę **NotCD** wejściowych słów niekodowych, dla których układ nie jest CD. Jeżeli tablica jest pusta, układ jest CD.

Algorytm 3.1 Algorytm weryfikacji własności CD układów

- 1 Wygeneruj listę **COut** wyjściowych słów kodowych
- 2 Wygeneruj listę **InError** słów niekodowych
- 3 $IEC = \text{SizeOf}(\text{InError})$
- 4 $ICS = \text{SizeOf}(\text{COut})$
- 5 $T = 0$

```

6 for  $i = 1$  to  $IEC$  do
7   begin
8      $T = 0$ 
9     for  $j = 1$  to  $ICS$  do
10      begin
11        if  $Covers(InError[i], In[j])$  then
12           $T = Or(T, Out[j])$ 
13        end
14        if  $T \notin COut$  then
15          Dodaj  $InError[i]$  do tablicy NotCD
16        end
17      end

```

Algorytm modyfikacji układu w celu uzyskania CD

Ponieważ warunki tw. 3.1 mogą nie być spełnione dla dowolnego SM (co wykazała analiza funkcji przejść/wyjść szeregu układów np. ze zbioru ISCAS89), dlatego zaproponowano metodę modyfikacji układu, pozwalającej uzyskać układ CD. Metoda ta stanowi rozwinięcie metody podanej w pracy [36].

Tw. 3.1 może nie być spełnione z dwóch powodów:

1. tylko dla jednego implikantu m_k zachodzi $m_k^{e+} \leq m_k$,
2. istnieją m_l oraz m_k takie, że $z^*(m_k) = z^*(m_l)$.

Wykażemy teraz, że dla szeregu przypadków istnieje możliwość takiej modyfikacji funkcji układu kombinacyjnego $\mathbf{H}$, aby warunki tw. 3.1 były spełnione, czyli układ był CD.

Rozważmy błąd $0 \rightarrow 1$ w bicie x_k słowa wejściowego X_i . Na skutek tego błędu na wejście układu zostanie podane niekodowe słowo $X_i^e \leftrightarrow m_i^e$. Załóżmy, że układ nie jest CD dla powyższego błędu oraz, że istnieją m_l , m_k takie, że $z^*(m_k) = z^*(m_l)$ oraz $m_i^e \subset m_k$ i $m_i^e \subset m_l$. W celu uzyskania CD dla błędu $0 \rightarrow 1$ w x_k , należy rozszerzyć rozmiar słowa wyjściowego w taki sposób, aby słowa wyjściowe odpowiadające m_l oraz m_k były nieuporządkowane względem siebie. W pracach [55], [36] zaproponowano dodawanie linii wyjściowych kodu 1/2 do rozdzielania każdej pary implikantów m_k oraz m_l , co powoduje znaczny nadmiar informacyjny i pośrednio sprzętowy. Nic jednak nie stoi na przeszkodzie, aby w przypadku większej liczby błędów, dla których układ nie zachowuje CD, dodatkowe linie wykorzystać do rozdzielania naraz kilku par m_l oraz m_k , w taki sposób, aby dla każdej pary implikantów m_l oraz m_k zachodziło $z^*(m_k) \neq z^*(m_l)$. Dla pozostałych słów wejściowych, należy ustalić stany dodatkowych linii w taki sposób, aby słowa wyjściowe były nieuporządkowane względem siebie. Jeżeli jest to konieczne, można dodać kolejne linie do rozdzielania innych implikantów.

Zwróćmy jednak uwagę na fundamentalne ograniczenie teoretyczne: jeżeli dla słowa wejściowego X_i^E istnieje tylko jedno m_k (nie jest spełniony warunek 2 tw. 3.1), to powyższa modyfikacja układu nie pozwala na uzyskanie CD.

Algorytm podany poniżej jest rozwinięciem algorytmu 3.1. W kroku 1 jest generowana zgodnie z algorytmem 3.1 tablica NotCD niekodowych słów wejściowych dla, których układ nie jest CD. W krokach 14–22 realizowana jest operacja rozdzielania zbiorów z^* słów wejściowych, dla których układ nie jest CD.

Algorytm 3.2 *Algorytm modyfikacji układów w celu uzyskania CD*

```

1 Wygenerować tablicę NotCD słów niekodowych, dla których układ nie jest CD
2 Dodać 2 do tablicy Out dodatkowe linie wyjściowe  $m$  oraz  $n$ 
3 for  $i = 1$  to  $SizeOf(Out)$  do  $Out[i][m] = Out[i][n] = -$ 
4  $NCDC = SizeOf(NotCD)$ 
5 for  $i = 1$  to  $NCDC$  do
6   begin
7     Wygenerować tablicę  $K$  zawierającą indeksy słów wejściowych pokrywanych przez  $NotCD[i]$ 
9     if  $SizeOf(K) = 1$  then Nie można skonstruować układu CD. goto 28
12     $k_1 = k_2 = -1$ 
13    Wybierać z  $K$  dwa elementy  $k_1$  i  $k_2$  takie, że
14     $Out[k_1][m] = Out[k_1][n] = -$  lub  $Out[k_2][m] = Out[k_2][n] = -$ 
15    if  $k_1 \neq -1 \wedge k_2 \neq -1$  then
16      begin
17        if  $Out[k_1][n] = Out[k_1][m] = - \wedge Out[k_2][n] = Out[k_2][m] = -$  then
18           $Out[k_1][n] = Out[k_2][m] = 1$ ;  $Out[k_1][m] = Out[k_1][n] = 0$ ;
19        if  $Out[k_1][n] = Out[k_1][m] = - \wedge Out[k_2][n] = Out[k_2][m] \neq -$  then
20           $Out[k_1][n] = Out[k_2][m] = 1$ ;  $Out[k_1][m] = Out[k_1][n] = 0$ ;
21        if  $Out[k_1][n] = Out[k_1][m] \neq - \wedge Out[k_2][n] = Out[k_2][m] = -$  then
22           $Out[k_1][n] = Out[k_2][m] = 1$ ;  $Out[k_1][m] = Out[k_1][n] = 0$ ;
23        Usunąć element  $i$  z tablicy NotCD
24      end
25    end
26 Dla pozostałych słów wyjściowych zakodować linie  $m$  oraz  $n$  kodem z inwersyjnym powtórzeniem
27 if  $SizeOf(NotCD) \neq 0$  goto 2
28 Koniec

```

Przykładowo, dla układu, którego tablicę przejść podano poniżej (wejścia zakodowane kodem 1/4, wyjścia niekompletnym kodem z inwersyjnym powtórzeniem), CD nie jest zachowana dla niekodowych słów wejściowych 1100 oraz 0011. Dla słowa 1100 na wyjściu otrzymamy słowo 1001, które należy do kodu wyjściowego (układ maskuje błąd), natomiast dla słowa 0011 otrzymamy słowo 0110, które także należy do kodu wyjściowego (układ maskuje błąd). Słowo 1100 pokrywa słowa 0100 oraz 1000, którym odpowiada wyjście 1001. Można zapewnić CD układu modyfikując funkcję układu tak, aby słowu 0100 odpowiadało wyjście 100110, a słowu 1000 wyjście 100101. Dzięki tej modyfikacji, dla niekodowego słowa wejściowego 0110 na wyjściu otrzymamy słowo 100111, które nie należy do kodu wyjściowego. Dla niekodowego słowa 0011 ustalamy wartości na dodatkowych liniach tak, aby słowu 0001 odpowiadało wyjście 011010, a słowu 0010 – słowo 011001.

Wejście	Wyjście	Wyjście układu CD
0001	0110	001101
0010	0110	001110
0100	1001	100110
1000	1001	100101

Zmodyfikowany w ten sposób układ jest CD dla wszystkich pojedynczych błędów wejściowych.

Przykład 3.1 Przeprowadźmy analizę SM zadanego poniższą tablicą przejść:

	I_1	I_2	I_3
A	E	C	A
B	C	E	A
C	D	B	A
D	B	C	D
E	B	C	B

Układ ten ma 3 słowa wejściowe I_1, I_2, I_3 oraz 5 stanów wewnętrznych A, B, C, D, E . Na wyjściu układu kombinacyjnego realizującego funkcję przejść obserwujemy binarnie zakodowane słowo stanu wewnętrznego. Przyjmijmy następujące kodowanie dla słów wejściowych (kod nieuporządkowany 1 z 3):

Wejście	$x_1x_2x_3$
I_1	100
I_2	010
I_3	001

oraz dla stanów układu (niekompletny kod nieuporządkowany 2/4):

Stan wewnętrzny	$x_4x_5x_6x_7$
A	1100
B	0110
C	0101
D	1001
E	1010

Dla tak przyjętego kodowania, układ realizujący funkcję przejść ma wejścia oraz wyjścia zakodowane kodem nieuporządkowanym. Wektor wejściowy dla układu $\mathbf{H}$ powstaje poprzez konkatencję słowa wejściowego oraz słowa stanu wewnętrznego SM. Bitom $x_1x_2x_3$ odpowiadają kolejne bity słowa wejściowego, natomiast bitom $x_4x_5x_6x_7$ – bity słowa stanu. Funkcja $\mathbf{H}$ układu kombinacyjnego realizującego funkcję przejść przedstawiona jest poniżej:

$$\begin{aligned}
 z_1 &= 145 + 157 + 247 + 256 + 345 + 326 + 356 + 357 \\
 z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 345 + 347 + 356 + 357 \\
 z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\
 z_4 &= 156 + 157 + 245 + 246 + 247 + 347
 \end{aligned}$$

Uwaga: Dla uproszczenia zapis 145 odpowiada iloczynowi $x_1x_4x_5$ itd.

Przeanalizujemy własność CD powyższego układu dla błędu w wejściowym słowie kodowym $X_{357} = 0010111$. Poprawna odpowiedź układu dla tego słowa wynosi $H(X_{357}) = 1100$. Przyjmijmy, że w X_{357} wystąpił błąd $0 \rightarrow 1$ w bicie x_6 — powstało niekodowe słowo wejściowe $X_{357}^E = 0010111$. Analizując własność CD układu otrzymujemy listę implikantów pokrywanych przez $X_{357}^E \subset \{m_{357}, m_{356}\}$. Na tej podstawie obliczamy $H(X_{357}^E) = 110 = H(X_{357})$. Otrzymane słowo należy do kodu C_{IN} , a więc układ nie jest CD dla błędu w bicie x_6 słowa X_{357} (układ maskuje ten błąd). Analiza warunku tw. 3.1 dla pozostałych błędów wejściowych wykazała, że układ nie jest CD dla błędu $0 \rightarrow 1$ w bicie x_7 słowa $X_{146}^E = 1001011$.

W celu zapewnienia własności CD układ można zmodyfikować w taki sposób, aby odpowiedzi układu na słowa niekodowe nie należały do kodu C_{OUT} . Jak zostało pokazane, układ nie jest CD dla niekodowego słowa $X_{357}^E = 0010111$. Słowo to pokrywa implikanty m_{356} i m_{357} , dla których $z^*(m_{356}) = z^*(m_{357}) = \{z_1, z_2\}$. Dodając dwie dodatkowe linie wyjściowe z_5 oraz z_6 , możemy zapewnić własność CD dla błędu $0 \rightarrow 1$ w bicie x_6 słowa X_{357} . Dodatkowe linie wykorzystujemy do rozdzielenia zbiorów $z^*(m_{356})$ i $z^*(m_{357})$. Uzyskamy w ten sposób np. $z^*(m_{356}) = 110001$ oraz $z^*(m_{357}) = 110010$. Modyfikacja taka spowoduje, że układ będzie CD dla niekodowego słowa X_{357}^E . Dodatkowe linie można również wykorzystać do rozdzielenia innych implikantów, dla których układ nie jest CD. Należy przy tym pamiętać, aby uzyskany w wyniku modyfikacji kod C_{OUT} był kodem nieuporządkowanym.

Okazało się, że dodanie tylko 1 pary linii kodu 1/2 wystarczyło do stworzenia układu w pełni CD dla wszystkich błędów $0 \rightarrow 1$ pojedynczych w słowach kodu wejściowego C_{IN} . Zmodyfikowana funkcja układu przedstawiona jest poniżej:

$$\begin{aligned} z_1 &= 145 + 157 + 247 + 256 + 345 + 346 + 356 + 357 \\ z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 345 + 347 + 356 + 357 \\ z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\ z_4 &= 156 + 157 + 245 + 246 + 247 + 347 \\ z_5 &= 146 + 156 + 245 + 247 + 257 + 346 + 356 \\ z_6 &= 145 + 147 + 157 + 246 + 256 + 345 + 347 + 357 \end{aligned}$$

Przypomnijmy, że na podstawie tw. 3.2 układ jest także CD dla błędów $1 \rightarrow 0$.

■

3.1.2 Kombinacyjne układy ST o strukturze 2-poziomowej AND-OR

W niniejszym podrozdziale zostaną przedstawione metody weryfikacji własności samotestowalności (ST) 2-poziomowych (AND-OR) układów kombinacyjnych o wejściach oraz wyjściach zakodowanych kodami nieuporządkowanymi.

Warunki weryfikacji własności ST układów AND-OR

W literaturze podano metody weryfikacji własności ST powyższych układów. Metody podane w [21] ograniczone były do układów o rozłącznych iloczynach, o wejściach i wyjściach zakodowanych ko-

dami m/n . W pracy [70] uogólniono metodę weryfikacji ST dla układów o wspólnych podukładach, ograniczoną jednak tylko do kodów m/n .

Analiza własności ST układu może być uproszczona, przy uwzględnieniu dominacji oraz równoważności uszkodzeń. W układzie o strukturze AND-OR zbiór rozpatrywanych uszkodzeń można ograniczyć do uszkodzeń $s/1$ na wejściach bramek AND oraz $s/0$ na wyjściach bramek AND.

Uszkodzenia $s/0$ na wyjściach bramek AND

Twierdzenie 3.3 *Nieredundantny układ kombinacyjny H realizujący funkcję (2.2), zrealizowany z wykorzystaniem wspólnych bramek AND, jest zawsze ST dla uszkodzenia $s/0$ w linii wyjściowej bramki AND realizującej iloczyn $m_k \leftrightarrow X_k$.*

Dowód: Testem dla uszkodzenia linii m_k jest słowo $X_k \leftrightarrow m_k$. W sprawnym układzie podanie X_k na wejście powoduje, że $m_k = 1$ oraz wszystkie wyjścia $z_l = 1$ ze zbioru $z^*(m_k)$. Natomiast, podanie X_k na wejście układu z uszkodzeniem linii $m_k/0$, powoduje wystąpienie błędu $1 \rightarrow 0$, który przepropaguje się na wszystkie wyjścia z_l ze zbioru $z^*(m_k)$. ■

Uszkodzenia $s/1$ na wejściach bramek AND

Twierdzenie 3.4 *Układ kombinacyjny H realizujący funkcję (2.2), zrealizowany z wykorzystaniem wspólnych bramek AND, jest ST dla wszystkich pojedynczych uszkodzeń $s/1$ wejściowych linii x_k bramek AND, wtedy i tylko wtedy gdy*

$$((\forall m_j \in M) (\forall x_k \in m_j(x^*)) (\exists m_l | z^*(m_j) \neq z^*(m_l) \text{ i } m_j(x_k) \subset m_l)). \quad (3.1)$$

Dowód:

Wystarczalność

Rozważmy uszkodzenie $s/1$ w linii $x_k \in m_j(x^*)$ na wejściu bramki AND realizującej iloczyn $m_j \leftrightarrow X_j$. Przyjmijmy ponadto, że spełnione są warunki tw. 3.4 tzn.

$$(\exists m_l | z^*(m_j) \neq z^*(m_l) \text{ i } m_j(x_k) \subset m_l) \quad (3.2)$$

Testem dla rozważanego uszkodzenia jest wejściowe słowo kodowe $X_l \leftrightarrow m_l$, dla którego nie tylko $m_l = 1$, ale także $m_k = 1$ (błąd $0 \rightarrow 1$). Ponieważ zbiory $z^*(m_l)$ oraz $z^*(m_k)$ są różne, błąd przepropaguje się na wyjście układu H .

Konieczność

Przyjmijmy, że układ jest ST dla uszkodzenia $s/1$ linii x_k w implikancie m_j . Oznacza to, że istnieje takie kodowe słowo wejściowe $X_l \leftrightarrow m_l$, które jest testem dla tego uszkodzenia. Jak łatwo zauważyć, słowo to musi pokrywać podimplikant $m_j(x_k)$ oraz zbiór wyjść $z^*(m_l) \neq z^*(m_j)$ tym samym słowo to spełnia warunki tw. 3.4. ■

Twierdzenia 3.3 i 3.4 podają formalne warunki weryfikacji samotestowalności układu zrealizowanego w strukturze 2-poziomowej AND-OR. Warunki tw. 3.3 są łatwe do spełnienia – wystarczy zagwarantować nieredundantną realizację układu o strukturze AND-OR. Natomiast niespełnienie warunków tw. 3.4 oznacza, że układ nie jest ST dla wszystkich pojedynczych uszkodzeń wewnętrznych s/z , dla których nie zachodzi 3.1.

Algorytm weryfikacji ST 2-poziomowych układów AND-OR

Przedstawione w podrozdziale 3.1.2 formalne warunki weryfikacji własności ST układów kombinacyjnych umożliwiły opracowanie algorytmu automatycznej weryfikacji własności ST.

Funkcja układu **H** zapisana jest w dwóch tablicach **TImpl** oraz **TOut**. W **TImpl** umieszczona jest lista wszystkich implikantów, natomiast w **TOut**— lista odpowiadających im słów wyjściowych (zbiory $z^*(m_i)$). Zmienna **IC** reprezentuje rozmiary obu tablic. Wyniki weryfikacji ST zapisywane są w tablicy **TStat** o rozmiarze równym tablicy **TImpl**. Dla potrzeb algorytmu zdefiniowano funkcję **DivImpl(X,i)**—obliczającą dzielnik X przez zmienną x_i oraz funkcję **Covered(X,Y)** – sprawdzającą relację pokrycia słów X przez Y . Rozmiary słów wejściowego oraz wyjściowego zapamiętane są odpowiednio w zmiennych **IS** oraz **OS**.

Weryfikacja własności ST układu odbywa się w dwóch etapach. W pierwszym, dla każdego implikantu m_k z tablicy **TImpl** oraz dla każdej linii wejściowej x_i generowany jest dzielnik T implikantu przez x_i . Następnie, tablica **TImpl** przeszukiwana jest w poszukiwaniu implikantów m_r , które pokrywają T . Jeżeli dla znalezionej implikantu m_r oraz implikantu m_k zbiory $z^*(m)$ są różne, układ jest ST dla uszkodzenia linii x_i w implikancie m_i . Dla linii testowalnych w tablicy **TStat** zapisywana jest wartość 0, natomiast dla nietestowalnych wartość 1. Algorytm w formie pseudokodu przedstawiony jest poniżej:

Algorytm 3.3 Algorytm weryfikacji ST dla układów o strukturze AND-OR

```

1  for  $i = 1$  to  $IC$  do
2    for  $j = 1$  to  $IS$  do
3      if  $TImpl[i][j] = 1$  then
4        begin
5           $T = DivImpl(TImpl[i], j);$ 
6           $TStat[i][j] = 1;$ 
7          for  $k = 1$  to  $IC$  do
8            if  $Covered(T, TImpl[k])$  then
9              if  $Out[i] \neq Out[k]$  then
10                begin
11                   $TStat[i][j] = 0;$ 
12                next  $j$ 
13              end
14            end
15  end
16  Koniec

```

Przykład 3.2 Należy zweryfikować własność ST funkcji przejść SM z przykładu 3.1. Funkcja **H** układu przedstawiona jest poniżej, a jego schemat o wewnętrznej strukturze AND-OR przedstawiony jest na rys. 3.1.

$$\begin{aligned}
 z_1 &= 145 + 157 + 247 + 256 + 345 + 326 + 356 + 357 \\
 z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 345 + 347 + 356 + 357 \\
 z_3 &= 145 + 146 + 147 + 256 + 257 + 346
 \end{aligned}$$

$$z_4 = 156 + 157 + 245 + 246 + 247 + 347$$

Korzystając z tw. 3.4 zweryfikujemy własność implikantu $m_{145} = x_1x_4x_5$ dla uszkodzenia $x_1/1$ linii wejściowej bramki AND. Dla m_{145} otrzymamy dzielnik $m_{145}(x_1) = x_4x_5$. Oznaczmy przez $M_{45} = \{145, 245, 345\}$ zbiór wszystkich implikantów m_l , dla których zachodzi $m_{145}(x_1) \subset m_l$. Dla tych implikantów należy znaleźć zbiory $z^*(m_{145}) = \{z_1, z_3\}$, $z^*(m_{245}) = \{z_2, z_4\}$ i $z^*(m_{345}) = \{z_1, z_2\}$. Jak można zauważyć, warunki tw. 3.4 spełnione są dla m_{245} i m_{345} , co wystarcza, aby układ był ST dla uszkodzenia $x_1/1$ wejściowej linii bramki m_{145} . Testem dla wymienionego uszkodzenia $x_1/1$ są słowa kodowe X_{245} i X_{345} . Ścieżka propagacji błędu spowodowanego powyższym uszkodzeniem przedstawiona jest na rys. 3.2. Linie w kolorze zielonym oznaczają poprawny stan 1 danej linii, natomiast kolorem czerwonym oznaczają niepoprawny stan 1. Pozostałe linie znajdują się w stanie 0.

Przeprowadźmy podobny test dla implikantu $m_{345} = x_3x_4x_5$ oraz uszkodzenia $s/1$ linii x_4 . Otrzymujemy $m_{345}(x_3) = x_3x_5$. Oznaczmy przez $M_{35} = \{345, 356, 457\}$, zbiór wszystkich implikantów m_l , dla których zachodzi $m_{345}(x_4) \subset m_l$. Ponieważ zbiory $z^*(m_{345}) = \{z_1, z_2\}$, $z^*(m_{356}) = \{z_1, z_2\}$ i $z^*(m_{357}) = \{z_1, z_2\}$ są identyczne, uszkodzenie linii x_4 w implikancie m_{345} nie jest testowane żadnym słowem kodowym.

W wyniku analizy przeprowadzonej dla wszystkich implikantów znajdujemy trzy nietestowane uszkodzenia układu **H** zaznaczone symbolem $\downarrow$

$$\begin{aligned} z_1 &= 145 + 157 + 247 + 256 + 3 \overset{\downarrow}{4} \overset{\downarrow}{5} + 326 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 3 \overset{\downarrow}{4} \overset{\downarrow}{5} + 347 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\ z_4 &= 156 + 157 + 245 + 246 + 247 + 347 \end{aligned}$$

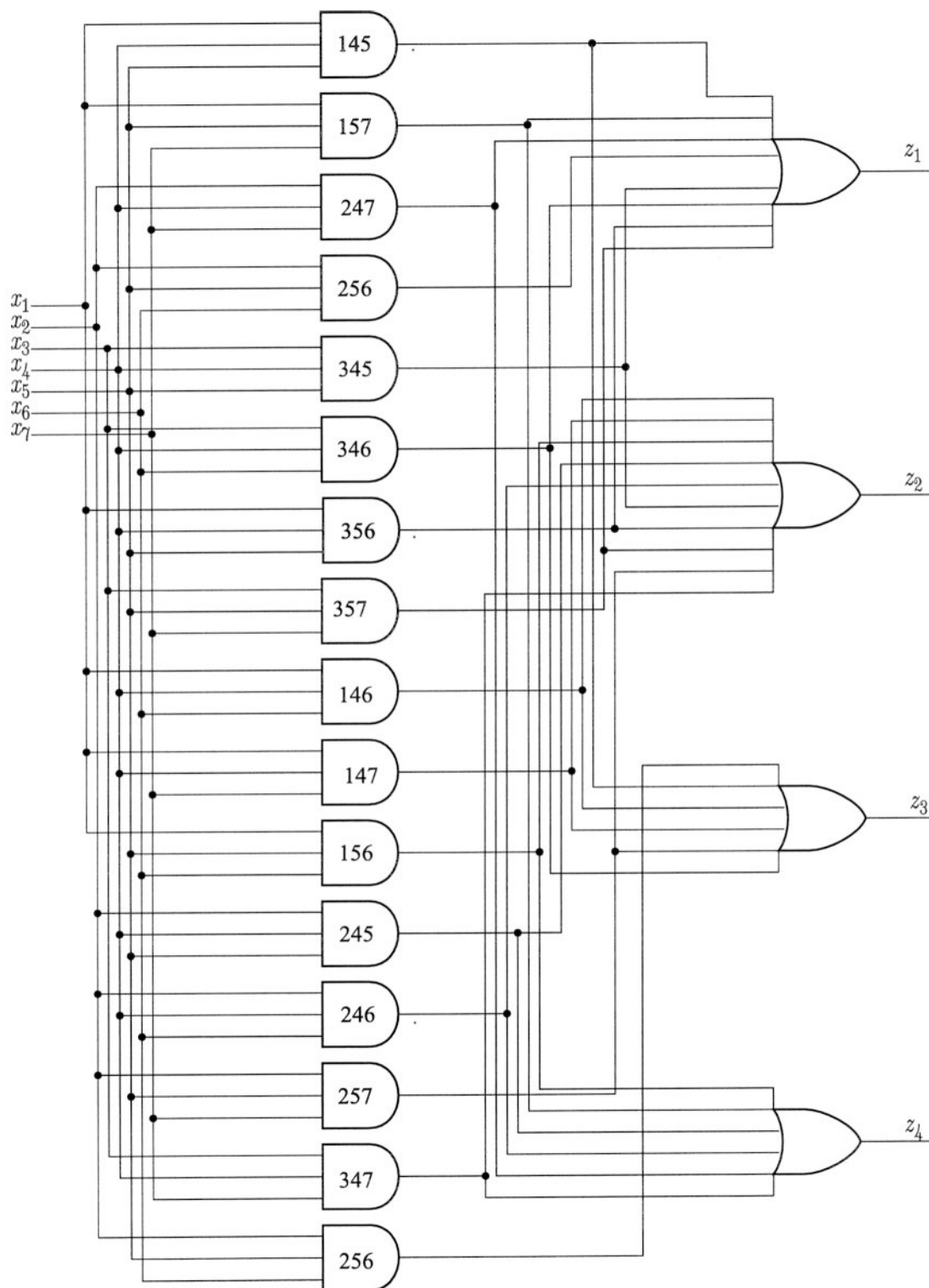
Dla pozostałych uszkodzeń układ jest samotestowalny. ■

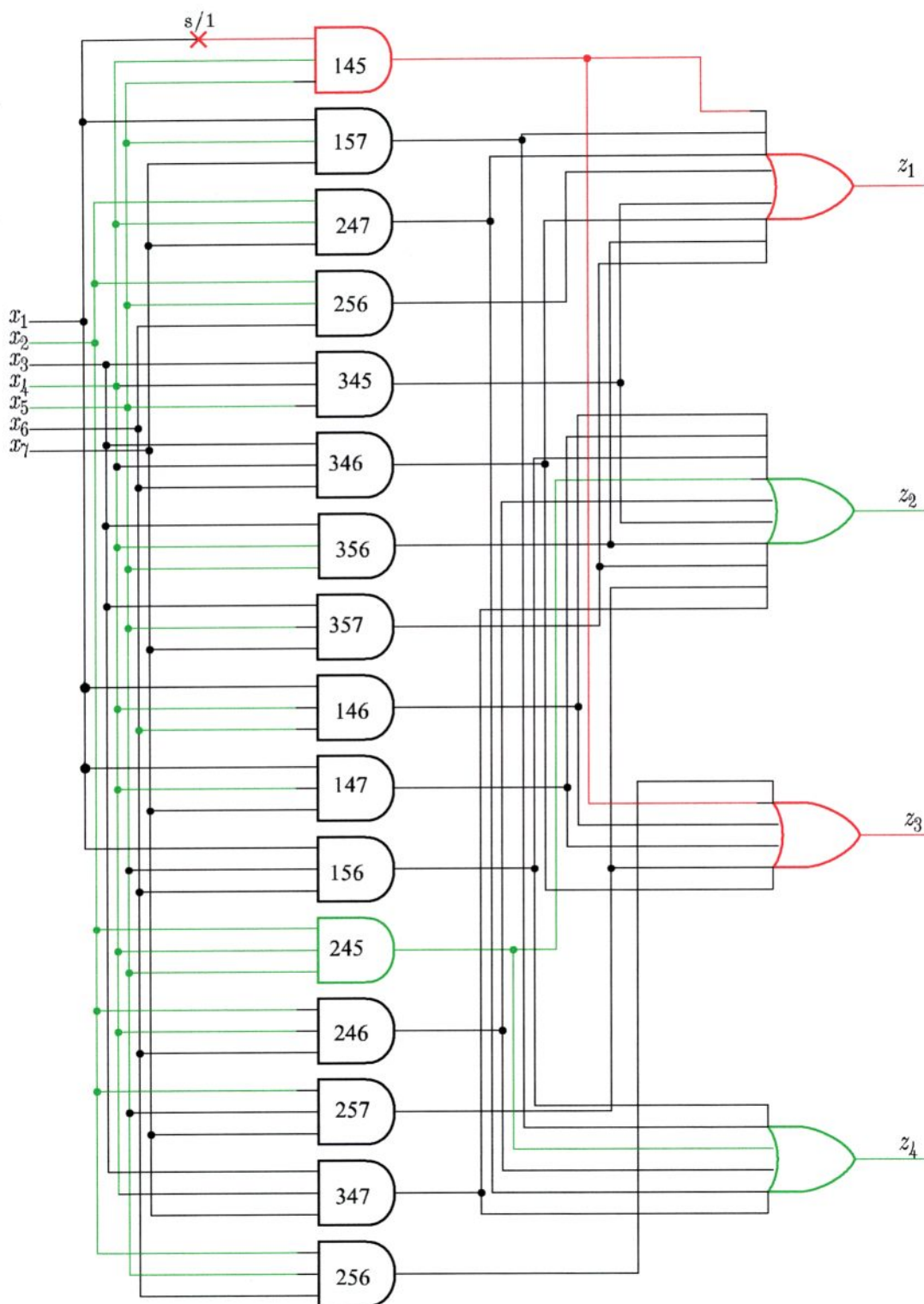
3.1.3 Kombinacyjne układy ST o strukturze 3-poziomowej AND–AND–OR

Warunki tw. 3.4 mogą być trudne do spełnienia, czego potwierdzeniem są wyniki weryfikacji własności ST funkcji przejść wzorcowych SM, wykorzystujących kody nieuporządkowane (rozdział 5). W tym podrozdziale zostanie przedstawiona metoda modyfikacji mająca poprawić ST układu, polegająca na dodaniu warstwy bramek AND w taki sposób, aby iloczyny m_i miały wspólne podimplikanty. Metoda ta stanowi rozwinięcie metody opisanej w [70], gdzie ograniczono się tylko do układów o wejściach i wyjściach zakodowanych kodami m/n .

Warunki weryfikacji ST układów AND–AND–OR

W układach kombinacyjnych o strukturze AND-OR mogą wystąpić uszkodzenia $s/1$ na wejściach bramek AND, dla których układ nie jest ST (nie spełniają warunków tw. 3.4). Eliminacji uszkodzeń nietestowalnych przez słowa kodu C_{IN} można dokonać dodając warstwę bramek AND w następujący sposób.

Rys. 3.1: Schemat układu **H** realizującego funkcję przejść z przykładu 3.2

Rys. 3.2: Schemat układu H z przykładowym uszkodzeniem wewnętrznym linii x_1

Przyjmijmy, że istnieją implikanty mające co najmniej dwie wspólne zmienne $x_{j_1}, \dots, x_{j_a}$ ($a \geq 2$) $m_j = \underbrace{(x_{j_1} \dots x_{j_a})}_{p_u} x_{j_{a+1}} \dots x_{j_c}$ oraz $m_i = \underbrace{(x_{j_1} \dots x_{j_a})}_{p_u} x_{i_{a+1}} \dots x_{i_c}$. Iloczyn p_u może zawierać dowolne zmienne, których uszkodzenie $s/1$ jest nietestowalne albo w m_j albo w m_i . Nietestowalne uszkodzenia w implikancie m_j można spróbować wyeliminować poprzez jego realizację w dwóch poziomach bramek AND. Pierwszy poziom generuje iloczyn p_u , wspólny dla implikantów m_i oraz m_j , drugi natomiast, iloczyny m_i oraz m_j wykorzystujące p_u , co możemy zapisać:

$$\begin{aligned} p_u &= x_{j_1} \dots x_{j_a} \\ m_j &= p_u x_{j_{a+1}} \dots x_{j_c} \\ m_i &= p_u x_{i_{a+1}} \dots x_{i_c} \end{aligned}$$

Dla przykładu rozważmy funkcje z_1 i z_2 z nietestowanym uszkodzeniem $s/1$:

$$\begin{aligned} z_1 &= \dots + x_1 x_2 \overset{\downarrow}{x_3} + \dots \\ z_2 &= \dots + x_2 x_3 x_4 + \dots \end{aligned}$$

Zauważmy, że iloczyn $x_2 x_3$ jest wspólny dla implikantów $x_1 x_2 x_3$ oraz $x_2 x_3 x_4$, skąd otrzymujemy

$$\begin{aligned} p_u &= x_2 x_3 \\ z_1 &= \dots + x_1 p_u + \dots \\ z_2 &= \dots + p_u x_4 + \dots \end{aligned}$$

Nietestowalne uszkodzenie linii x_3 wejściowej bramki m_{123} może być testowane poprzez bramkę m_{234} , przy zachowaniu dodatkowych warunków podanych poniżej w tw. 3.5.

Niech p_u , $u \subset \{1, r\}$, będzie podimplikantem $m_i = (x_{j_1} \dots x_{j_a}) p_1 \dots p_r$. Jeżeli każdy p_u zawiera co najmniej dwie zmienne oraz $p_u \neq m_i$, to istnieje możliwość realizacji układu z współdzieleniem bramki AND realizującej iloczyn p_u .

Niech $m_i(p^*) = \{p_1, \dots, p_r\}$ oznacza zbiór wszystkich podimplikantów w m_i , a P – zbiór wszystkich podimplikantów w układzie kombinacyjnym **H**. Zbiorowi P odpowiada zbiór wszystkich bramek pierwszego poziomu, współdzielonych w układzie **H**. Dzielnik $m_j(p_u)$ implikantu m_i przez p_u ($p_u \subset m_i$) otrzymujemy ustalając $p_u = 1$ w implikancie m_i .

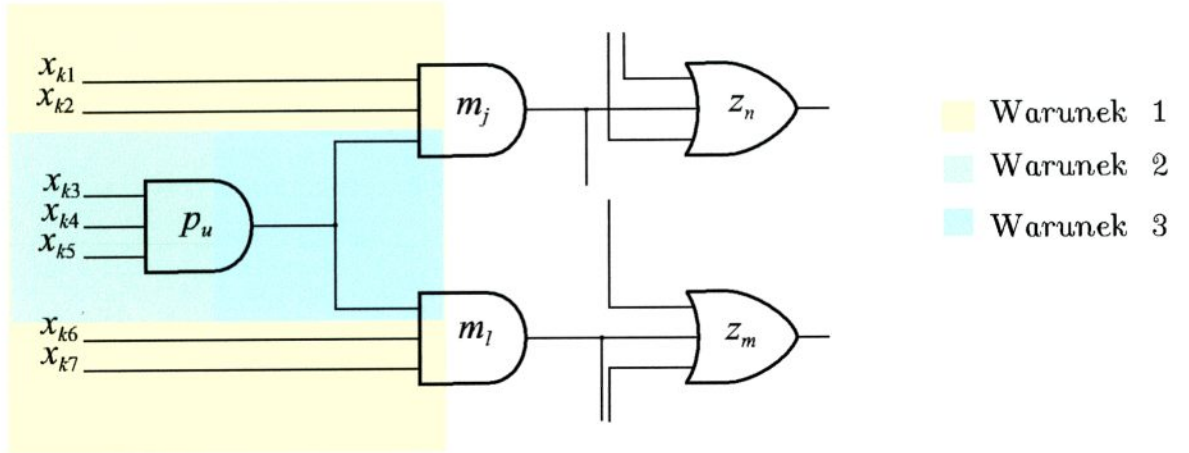
Twierdzenie 3.5 *Jeżeli w 3-poziomowym (AND-AND-OR) kombinacyjnym układzie **H** realizującym funkcję (2.2), wykorzystującym wspólne iloczyny pierwszego poziomu bramek AND, dla każdego $m_j \in M$ spełnione są warunki:*

- 1) $(\forall x_k \in m_j(x^*)) (\exists m_l | z^*(m_j) \neq z^*(m_k) \text{ oraz } m_j(x_k) \subset m_l)$,
- 2) $(\forall p_u \in m_j(p^*)) (\exists m_l | z^*(m_j) \neq z^*(m_k) \text{ oraz } m_j(p_u) \subset m_l)$,

oraz dla każdego $p_u \in P$ spełniony jest warunek

- 3) $(\forall x_k \in p_u(x^*)) (\exists m_r, m_l, m_r \in M, m_l \in M | p_u \in m_l(p^*)$
 $i \ m_r \subset m_l(x_k) \ i \ z^*(m_l) \neq z^*(m_r))$,

to układ jest samotestowalny dla dowolnego pojedynczego uszkodzenia $s/1$.



Rys. 3.3: Fragment układu o strukturze AND-AND-OR

Dowód: Rozważmy fragment przykładowej struktury z rys. 3.3. Warunek 1 odpowiada za testowalność linii wejściowych x_k podłączonych bezpośrednio do wejść bramek AND drugiego poziomu (realizujących m_j i m_l). Warunek 2 odpowiada za testowalność uszkodzeń w liniach łączących wyjścia bramek AND pierwszego poziomu z wejściami bramek AND drugiego poziomu. Warunek 3 dotyczy testowalności linii wejściowych pierwszego poziomu bramek AND (realizującej p_u).

Dowód warunku 1 jest analogiczny do dowodu tw. 3.4.

Rozpatrzmy warunek 2 odpowiadający za testowanie uszkodzeń w liniach p_u łączących bramki AND pierwszego poziomu z bramkami AND drugiego poziomu. Potencjalnym testem jest każde słowo $X_l \leftrightarrow m_l$, które spowoduje, że błąd wygenerowany uszkodzeniem $s/1$ linii p_u przepropaguje się na wyjście. Warunek ten spełnia słowo $X_l \leftrightarrow m_l$ takie, że $m_j(p_u) \subset m_l$ oraz $z^*(m_j) \neq z^*(m_k)$.

Przy braku uszkodzenia mamy:

$$\begin{aligned} p_u &= 0 \\ m_j(p_u) &\subset m_l \\ m_j(p_u) &= 1 \\ m_j &= m_j(p_u)p_u \\ m_j &= 0 \end{aligned}$$

Natomiast w obecności uszkodzenia $p_u/1$ i podaniu na wejście układu słowa X_l , na wyjściu bramki m_j otrzymamy 1 zamiast 0, ponieważ:

$$\begin{aligned} p_u &= 1 \\ m_j(p_u) &\subset m_l \\ m_j(p_u) &= 1 \\ m_j &= m_j(p_u)p_u \\ m_j &= 1 \end{aligned}$$

Ponieważ zbiory $z^*(m_j)$ i $z^*(m_k)$ są różne, błąd $0 \rightarrow 1$ w linii p_u przepropaguje się na wyjście układu.

Dla wykazania warunku 3, rozpatrzmy uszkodzenia linii pierwotnych podłączonych do wejść bramek AND pierwszego poziomu. Przyjmijmy, że uszkodzenie $s/1$ wystąpiło w linii x_k na wejściu bramki realizującej iloczyn p_u . Testem dla tego uszkodzenia jest słowo $X_r \leftrightarrow m_r$, podanie którego na wejście układu spowoduje, że implikant $m_r = 1$. Zauważmy, że ponieważ $m_r \subset m_l(x_k)$ oraz na skutek uszkodzenia $x_k = 1$, także iloczyn $m_l = 1$. Ponieważ zbiory $z^*(m_r)$ i $z^*(m_l)$ są różne, błąd $0 \rightarrow 1$ spowodowany uszkodzeniem przepropaguje się na wyjście układu. ■

Przykład 3.3 Rozważmy SM z przykładu 3.1. Analiza własności ST układu kombinacyjnego realizującego funkcję przejść zrealizowanego w wersji AND-OR(przykład 3.2), wykazała obecność 3 nietestowalnych uszkodzeń (oznaczonych symbolem $\downarrow$):

$$\begin{aligned} z_1 &= 145 + 157 + 247 + 256 + 3 \overset{\downarrow}{4} 5 + 326 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 3 \overset{\downarrow}{4} 5 + 347 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\ z_4 &= 156 + 157 + 245 + 246 + 247 + 347 \end{aligned}$$

Ponieważ iloczyn 34 występuje zarówno w implikantach ze zmiennymi nietestowalnymi jak i testowalnymi, zmodyfikujemy funkcję układu do podanej poniżej postaci, który odpowiada 3-poziomowej (AND-AND-OR) realizacji układu:

$$\begin{aligned} p_{34} &= 34 \\ z_1 &= 145 + 157 + 247 + 256 + p_{34}5 + 326 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + p_{34}5 + p_{34}7 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_3 &= 145 + 146 + 147 + 256 + 257 + p_{34}6 \\ z_4 &= 156 + 157 + 245 + 246 + 247 + p_{34}7 \end{aligned}$$

Przeanalizujemy uszkodzenia linii wejściowych x_3, x_4, x_5 w implikancie $m_{345} = x_3 \overset{\downarrow}{x_4} x_5$. W tym celu należy przeprowadzić weryfikację własności ST dla uszkodzeń w następujących miejscach:

1. linia x_5 – wejście bramki AND drugiego poziomu,
2. linia łącząca bramkę p_{34} z wejściami bramek AND drugiego poziomu, oraz
3. linie x_3 oraz x_4 – wejścia bramki AND pierwszego poziomu.

Zweryfikujemy własność ST dla uszkodzenia $x_5/1$. Jest ono testowalne, ponieważ warunek 1 tw. 3.5 jest spełniony: dla m_{345} istnieje m_{145} takie, że $z^*(m_{145}) \neq z^*(m_{345})$. Zatem testem dla tego uszkodzenia jest $X_{145} \leftrightarrow m_{145}$.

Dla uszkodzenia $p_{34}/1$ warunek 2 tw. 3.5 jest również spełniony, ponieważ istnieje $m_{345}(p_{34}) \subset m_{156}$. Zatem powyższe uszkodzenie jest testowalne słowem X_{156} .

Pozostała teraz do zweryfikowania testowalność dla uszkodzeń $s/1$ w liniach wejściowych x_3 oraz x_4 bramki p_{34} . Pierwsze z nich $x_3/1$ było testowalne w układzie zrealizowanym jako układ 2-poziomowy AND-OR, natomiast dla uszkodzenia $x_4/1$ nie istniało słowo kodowe będące testem. W strukturze

3-poziomowej uszkodzenie to jest testowalne, ponieważ w obecności uszkodzenia dla słowa kodowego 356, otrzymujemy $m_{356} = 1$ oraz na skutek uszkodzenia zachodzi $x_4 = 1$, $p_{34} = 1$ i $m_{345} = 1$. Ponieważ $z^*(m_{356}) \neq z^*(m_{345})$, błąd $0 \rightarrow 1$ w linii x_4 propaguje się na wyjście układu. W wyniku podobnej analizy można wykazać, że uszkodzenie linii x_3 jest także testowalne. Dla weryfikacji ST wejściowych linii pierwotnych bramek drugiego poziomu AND zastosowanie ma warunek 1 tw. 3.5 równoważny tw. 3.4. ■

Modyfikacja układów AND-OR w celu poprawy ST

Warunki sformułowane jako tw. 3.4 i tw. 3.5 umożliwiają weryfikację własności ST układów kombinacyjnych wykorzystujących kody nieuporządkowane o strukturze wewnętrznej AND-OR oraz AND-AND-OR.

Opis algorytmu

Punktem wyjścia do syntezy układu AND-AND-OR jest układ o strukturze AND-OR, dla którego najpierw weryfikujemy własność ST stosując algorytm 3.3. W wyniku analizy otrzymujemy listę implikantów, dla których układ nie jest ST. Przyjmijmy, że układ nie jest ST dla uszkodzenia w linii x_{k_n} w implikancie $m_i = x_{i_1} \dots x_{i_{k_n}} \dots x_{k_p} \dots x_{i_n}$. Stąd łatwo możemy znaleźć listę podimplikantów p_u o dwóch zmiennych x_{k_n} oraz x_{k_p} takich, że $x_{k_n} \in p_u(x^*)$ oraz $x_{k_p} \in p_u(x^*)$. Dla każdego elementu z listy podimplikantów szukamy implikantów m_j takich, że $m_j \subset p_u$. Następnie dla każdego m_j weryfikujemy warunki 2 oraz 3 tw. 3.5 przyjmując, że implikanty m_j oraz m_i mają wspólny podimplikant p_u . Dla pierwszego implikantu m_j oraz podimplikantu p_u , dla którego stwierdzimy spełnienie warunków tw. 3.5, uwspólniamy iloczyn p_u dla implikantów m_i oraz m_j przyjmując:

$$p_u = x_{k_n} x_{k_p}$$

$$m_i = m_i(p_u) p_u$$

$$m_j = m_j(p_u) p_u.$$

Algorytm powtarzamy dla każdej linii x_{k_n} , która jest nietestowalna.

Implementacja algorytmu wymaga modyfikacji struktur opisanych w rozdziale 3.1.2. Dla przypomnienia, do opisu układów zbudowanych z dwóch poziomów bramek wykorzystywane są trzy tablice TImpl, TOut i TStat. Rozwinięcie algorytmu wymaga dodatkowych tablic TShare i TShareStat do opisu dodatkowej warstwy bramek AND (realizującej iloczyn p_u). W tablicy TShare zapamiętana jest lista wspólnych iloczynów p_u w układzie, a zatem jej rozmiar powinien być równy liczbie wszystkich iloczynów p_u układzie. Jednak przed syntezą układu nie jest możliwe dokładne określenie tego rozmiaru, tak więc można przyjąć rozmiar tablicy TShare równy liczbie nietestowalnych uszkodzeń w układzie AND-OR, ponieważ liczba iloczynów p_u zawsze będzie mniejsza lub równa liczbie uszkodzeń nietestowalnych. Tablica TShareStat, o rozmiarach takich jak TShare, zawiera informacje o testowalności poszczególnych linii x_i iloczynów p_u . W związku z wprowadzeniem dodatkowej warstwy bramek AND, zwiększeniu uległy także rozmiary tablic TImpl, TOut i TStat, gdyż w tablicach tych będzie umieszczona dodatkowa informacja o zbiorach $m_i(p^*)$ dla poszczególnych implikantów. W zaproponowanym algorytmie iloczyny p_u zawierają tylko dwie zmienne. Założenie to nie powoduje

utruty własności ST układu, wymaga tylko większej liczby uwspólnień. Redukuje natomiast znacznie złożoność obliczeniową, gdyż nie ma potrzeby generowania listy wszystkich możliwych podimplikantów. Algorytm 3.4 zapisany w formie pseudokodu przedstawiony jest poniżej. Przedstawiony fragment kodu jest głównym blokiem algorytmu, który wykorzystuje dwie funkcje **FindP** oraz **FindI**. Funkcja **FindP** znajduje możliwość uwspólnienia bramek oraz weryfikuje warunek 3 tw. 3.5. Dla znalezionej uwspólnienia weryfikowane jest z wykorzystaniem funkcji **FindI** warunek 2 tw. 3.5. Jeżeli znalezione uwspólnienie spełnia oba warunki, struktura układu jest modyfikowana.

Algorytm 3.4 *Algorytm eliminacji nietestowalnych uszkodzeń w układach o strukturze AND-OR*

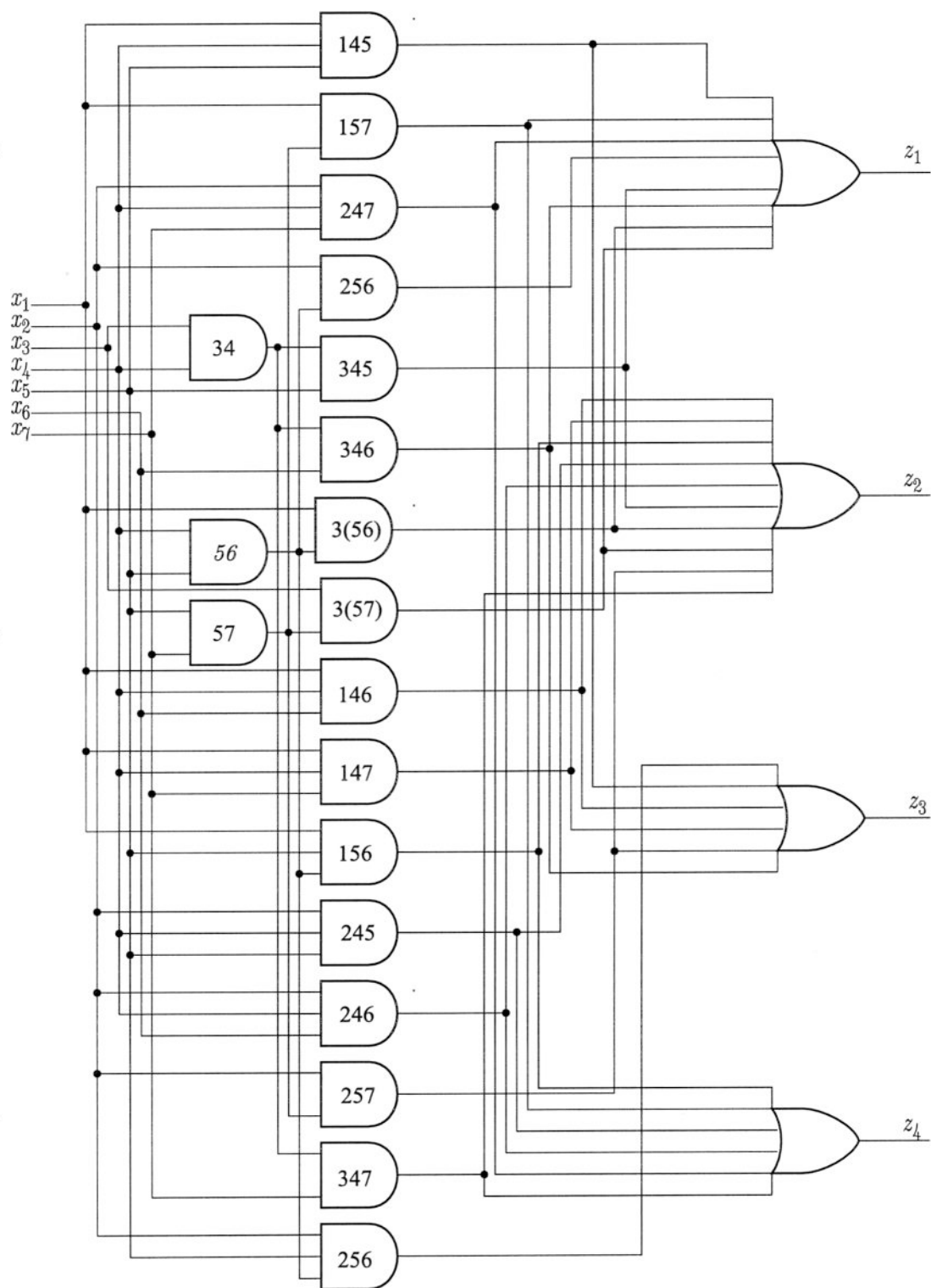
```

1 for Impl = 1 to ImplCount do
2   begin
3     T
4     for i = 1 to InputsCount do
5       begin
6         if TStat[Impl][i] = 1  $\wedge$  TImpl[Impl][i] = '1' then
7           begin
8             for j = 1 to InputsCount do
9               begin
10                if (TImpl[Impl][j] = '1') then
11                  begin
12                    if ((Y = FindP(i, j, Impl))  $\neq$  -1) then
13                      begin
14                        T = TImpl[Impl];
15                        T[j] = T[i] = '0';
16                        if (FindI(T, Impl)  $\neq$  -1) then
17                          begin
18                            T = TImpl[Y];
19                            T[j] = T[i] = '0';
20                            if FindI(T, Impl)  $\neq$  -1 then
21                              begin
22                                if S[Impl][i] = 1 then S[Impl][i] = 2
23                                if S[Impl][j] = 1 then S[Impl][j] = 2
24                              end
25                            end
26                          end
27                        end
28                      end
29                    end
30                  end

```

Przykład 3.4 Przypomnijmy funkcje układu przejść SM opisanego w przykładzie 3.1:

$$z_1 = 145 + 157 + 247 + 256 + 3 \overset{\downarrow}{4} 5 + 326 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7}$$

Rys. 3.4: Schemat ST układu H o strukturze AND-AND-OR z przykładu 3.4.

$$\begin{aligned}
z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 3 \overset{\downarrow}{4} 5 + 347 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\
z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\
z_4 &= 156 + 157 + 245 + 246 + 247 + 347
\end{aligned}$$

Rozważmy nietestowalne uszkodzenie linii x_4 w implikancie $m_{345} = x_3 \overset{\downarrow}{x_4} x_5$. Istnieją dwa podimplikanty p_u w implikancie m_{345} , zawierające zmienną x_4 : $p_{34} = x_3 x_4$ oraz $p_{45} = x_4 x_5$. Dla każdego z tych podimplikantów znajdujemy listę implikantów m_k takich, że p_u jest ich podimplikantem. Otrzymujemy w ten sposób implikanty m_{347} i m_{345} dla p_{34} oraz m_{345} , m_{145} , m_{245} i m_{345} dla p_{45} . Analizując zbiory $m^*(z)$ dla tych implikantów otrzymujemy: $m_{345}^*(z) = \{z_1, z_2\}$, $m_{347}^*(z) = \{z_2, z_4\}$ oraz $m_{145}^*(z) = \{z_1, z_3\}$, $m_{245}^*(z) = \{z_2, z_4\}$ i $m_{345}^*(z) = \{z_1, z_2\}$. Warunki 2 i 3 tw. 3.5 są spełnione przez wszystkie powyższe implikanty, możemy więc wybrać dowolny z nich, np. m_{347} , skąd otrzymamy:

$$p_{34} = x_3 x_4$$

$$m_{345} = p_{34} x_5$$

$$m_{347} = p_{34} x_7$$

Podobnej analizy dokonujemy dla pozostałych nietestowalnych uszkodzeń w implikantach $x_3 x_5 \overset{\downarrow}{x_6}$ oraz $x_3 x_5 \overset{\downarrow}{x_7}$. Wprowadzając trzy dodatkowe bramki AND pierwszego poziomu, otrzymujemy następujące funkcje układu, który jest ST dla wszystkich uszkodzeń s/z :

$$\begin{aligned}
p_{34} &= 34 \\
p_{56} &= 56 \\
p_{57} &= 57 \\
z_1 &= 145 + 1p_{57} + 247 + 2p_{56} + p_{34}5 + 326 + 3p_{56} + 3p_{57} \\
z_2 &= 146 + 147 + 1p_{56} + 245 + 246 + 2p_{57} + p_{34}5 + p_{34}7 + 3p_{56} + 3p_{57} \\
z_3 &= 145 + 146 + 147 + 2p_{56} + 2p_{57} + p_{34}6 \\
z_4 &= 1p_{56} + 1p_{57} + 245 + 246 + 247 + p_{34}7
\end{aligned}$$

a którego schemat został przedstawiony na rys. 3.4.

3.1.4 Podsumowanie

W rozdziale tym podano nową metodę syntezy kodowo-rozłącznych TSC układów kombinacyjnych z użyciem kodów nieuporządkowanych. Metoda ta, w przeciwieństwie do wszystkich znanych metod, uwzględnia możliwość wystąpienia błędów i uszkodzeń na wejściach pierwotnych układu. W rozdziale tym podano również twierdzenia umożliwiające weryfikację własności CD i ST układów o strukturze AND-OR. Zaproponowano metody modyfikacji układu, które umożliwiają uzyskanie CD i ST. Dla układów, które nie są CD, podano także algorytm modyfikacji układu, polegający na dodaniu par linii kodu z inwersyjnym powtórzeniem, umożliwiającą uzyskanie własności CD. Natomiast dla układów, które nie są ST, zaproponowano metodę dodawania dodatkowej warstwy bramek AND w celu uzyskania ST. Należy tutaj zaznaczyć, że w odróżnieniu od znanych z literatury metod [21, 22, 33, 39], metoda ta nie modyfikuje funkcji układu, lecz jedynie strukturę układu. Dzięki temu nie ulega zmianie funkcjonalność układu dla niekodowych pobudzeń, co nie powoduje utraty własności CD układów.

3.2 Synteza układów kombinacyjnych TSC/STC

W podrozdziale tym zostanie przedstawiona metoda syntezy TSC SM o strukturze TSC/STC. W rozwiązaniu takim zakłada się, że układ kombinacyjny H jest samotestowalny (ST), ale nie jest kodowo-rozłączny (CD). Niesie to ze sobą konieczność zastosowania ST układów kontrolnych (STC) monitorujących wejście, stany wewnętrzne oraz wyjście układu. Układy takie w odróżnieniu od układów TSC/CD przedstawionych w podrozdziałach 3.1.1–3.1.3, mogą cechować się mniejszą złożonością, jednakże okupioną koniecznością zastosowania układów STC.

3.2.1 Minimalizacja funkcji układu kombinacyjnego

Przedstawione w podrozdziałach 3.1.1–3.1.3 metody syntezy układów TSC/CD opierają się tylko na zmianach wewnętrznej struktury układu. W metodach tych nie jest konieczne stosowanie STC do monitorowania wejść oraz stanów wewnętrznych układu dzięki zachowaniu własności CD. Natomiast podejście przedstawione w tym podrozdziale, polega na zmianach funkcji układu dla niekodowych pobudeń w celu poprawy ST. Niesie to ze sobą ryzyko zmian w strukturze układu, które spowodują, że uzyskany układ nie będzie CD.

Metoda syntezy układów ST, polegająca na usuwaniu zmiennych, których uszkodzenia są nietestowalne, została przedstawiona w pracy [21], gdzie autor pokazał metodę syntezy układów ST o rozłącznych podukładach. W pracy [70] uogólniono tę metodę dla układów o wspólnych podukładach. Metody podane w pracach [21], [70] były ograniczone do układów wykorzystujących kody m/n .

Rozważmy układ kombinacyjny H , którego wejścia oraz wyjścia są zakodowane kodem nieuporządkowanym. Przyjmijmy, że układ realizuje funkcję (2.2) i ma wewnętrzną strukturę AND-OR. Załóżmy ponadto, że w układzie jest nietestowalne uszkodzenie $s/1$ linii x_k implikantu m_i . W czasie normalnej pracy układu na wejściu pojawiają się tylko słowa kodowe kodu wejściowego. Ponieważ uszkodzenie $s/1$ linii x_k nie jest testowane słowami kodu wejściowego, można więc przyjąć, jak pokazano w [21], $x_k = 1$, a tym samym zredukować liczbę zmiennych w implikancie m_i . Niestety, podejście to modyfikuje funkcję realizowaną przez układ, na niekorzyść CD, gdyż zmiana może ulec funkcjonalność układu dla niekodowych słów wejściowych, a tym samym otrzymany układ może przestać być CD dla błędów typu $1 \rightarrow 0$. W tej sytuacji, aby cały układ był TSC, należy stosować STC, w celu weryfikacji poprawności słów wejściowych.

3.2.2 Algorytm minimalizacji funkcji pozytywnie gładkich

Podana w [23] metoda, ograniczona do eliminacji tylko tych zmiennych, dla których nie jest zapewniona własność ST, jest punktem wyjścia do opracowania ogólniejszej metody minimalizacji funkcji logicznych układów kombinacyjnych wykorzystujących kody nieuporządkowane, gwarantującej ponadto ST otrzymanego układu.

Przypomnijmy funkcję układu kombinacyjnego opisanego w przykładzie 3.1:

$$\begin{aligned} z_1 &= 145 + 157 + 247 + 256 + 3 \overset{\downarrow}{4} 5 + 326 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 3 \overset{\downarrow}{4} 5 + 347 + 35 \overset{\downarrow}{6} + 35 \overset{\downarrow}{7} \\ z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \end{aligned}$$

$$z_4 = 156 + 157 + 245 + 246 + 247 + 347$$

Zauważmy, że podimplikant $p_u = 16$ należy tylko do implikantów m_{146} i m_{156} , suma z_2 zawiera oba wymienione implikanty. Zakładając, że na wejście układu podawane będą tylko słowa kodu C_{IN} wśród których tylko dwa X_{146} oraz X_{156} pokrywają podimplikant $p_u = 16$, można przyjąć $m_{146}(p_{16}) = 1$ oraz $m_{156}(p_{16}) = 1$ w implikantach należących do sumy z_2 . W ten sposób otrzymujemy układ:

$$\begin{aligned} z_1 &= 145 + 157 + 247 + 256 + 345 + 326 + 356 + 357 \\ z_2 &= 16 + 147 + 245 + 246 + 257 + 345 + 347 + 356 + 357 \\ z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\ z_4 &= 145 + 157 + 245 + 246 + 247 + 347 \end{aligned}$$

Twierdzenie 3.6 *Jeżeli dla układu kombinacyjnego H realizującego funkcję (2.2) spełniona jest zależność:*

$$(\forall m_l, m_l \subset m^*(p_u) | z_k \in z^*(m_l)), \quad (3.3)$$

to istnieje możliwość zastąpienia wszystkich implikantów sumy z_k zawierających podimplikant p_u podimplikantem p_u .

Dowód: Przy założeniu, że warunek tw. 3.6 jest spełniony, zbiór $z^*(m_l)$ zawiera wszystkie implikanty m_l , które z kolei zawierają podimplikant p_u . Dla przyjętego założenia otrzymujemy: $z_k = \dots + p_u(m_{l_1}(p_u) + \dots + m_{l_n}(p_u)) + \dots$. Zakładając, że na wejście układu podawane są tylko słowa kodu wejściowego, możemy przyjąć $m_{l_1}(p_u) + \dots + m_{l_n}(p_u) = 1$, skąd otrzymamy $z_k = \dots + p_u + \dots$ ■

Twierdzenie powyższe podaje formalne warunki redukcji liczby zmiennych, tym samym stwarza możliwość minimalizacji funkcji układu przy założeniu, że na wejście układu podawane będą tylko słowa ze zbioru C_{IN} .

Na podstawie tw. 3.6 opracowano algorytm realizujący minimalizację funkcji układu kombinacyjnego H . Jak można zauważyć, w celu uzyskania najmniej złożonego układu należy starać się zredukować implikanty do iloczynów p_u o jak najmniejszej liczbie zmiennych. W opracowanym algorytmie sprawdzamy możliwość eliminacji zmiennych dla wszystkich podimplikantów p_u , które zawierają się we wszystkich implikantach m_k , $m_k \in M$. Zbiór podimplikantów p_u jest posortowany rosnąco względem wagi. Dla każdego elementu tego zbioru weryfikowany jest warunek tw. 3.6. Problemem może być wygenerowanie zbioru wszystkich podimplikantów p_u , dlatego w zaproponowanym rozwiązaniu zbiór podimplikantów p_u jest generowany w kilku etapach. W pierwszej kolejności generowane są podimplikanty o wadze 1 i dla tego zbioru weryfikowany jest warunek tw. 3.6, następnie podimplikanty o wadze 2, itd. Algorytm w formie pseudokodu przedstawiony jest poniżej:

Algorytm 3.5 *Algorytm minimalizacji układów o strukturze AND-OR*

- 1 **var** $w = 1, i, j$
- 2 Wygenerować zbiór INP słów wejściowych
- 3 Wygenerować zbiór Sub podimplikantów słów wejściowych o wadze w
- 4 **for** $i = 1$ **to** $SizeOf(Sub)$ **do**

```

5   for  $j = 1$  to LiczbaWyjść do
6       if Sub[i] spełnia warunki tw. 3.6 dla sumy  $z_j$  then
7           begin
8               Zastąpić podimplikantem Sub[i] implikanty sumy  $z_j$  pokrywające Sub[i]
9               Usunąć z INP implikanty, które zostały zastąpione
10          end
11   $w = w + 1$ 
12  if Istnieją w INP implikanty o wadze większej od  $w$  then goto 3

```

Przedstawiony algorytm weryfikuje możliwość eliminacji zmiennych, zgodnie z tw. 3.6 dla wszystkich podimplikantów.

3.2.3 Weryfikacja własności ST zminimalizowanych układów

Twierdzenie 3.7 *Jeżeli układ jest zminimalizowany zgodnie z algorytmem 3.5, to jest ST.*

Dowód: Przyjmijmy, że w układzie niezminimalizowanym w implikancie m_l należącym do sumy z_m jest nietestowane uszkodzenie linii x_i . Zgodnie z tw. 3.4 oznacza to, że mogły wystąpić dwa przypadki: 1) dzielnik $m_l(x_i)$ jest pokrywany tylko przez podimplikant m_l lub 2) dzielnik $m_l(x_i)$ jest pokrywany przez kilka implikantów, dla których zbiory z^* są równe. W pierwszym przypadku dzielnik $m_l(x_i)$ będący podimplikantem m_l spełnia warunki tw. 3.6, ponieważ suma z_l zawiera wszystkie implikanty (tylko m_l) pokrywające podimplikant $m_l(x_i)$. Natomiast w drugim przypadku, co prawda dzielnik $m_l(x_i)$ jest podimplikantem kilku implikantów m jednak suma z_m zawiera wszystkie te implikanty, ponieważ zbiory z^* dla tych implikantów są równe. Tym samym podimplikant $m_l(x_i)$ implikantu m_l spełnia warunki tw. 3.6. Oznacza to, że w układzie zminimalizowanym, implikant m_l zostanie zastąpiony podimplikantem $m_l(x_i)$, tym samym zostanie usunięta nietestowalna zmienna x_i . ■

Przykład 3.5 Przypomnijmy funkcję układu kombinacyjnego realizującego funkcję przejść dla SM z przykładu 3.1:

$$\begin{aligned}
 z_1 &= 145 + 157 + 247 + 256 + 345 + 326 + 356 + 357 \\
 z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 345 + 347 + 356 + 357 \\
 z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\
 z_4 &= 156 + 157 + 245 + 246 + 247 + 347
 \end{aligned}$$

Zgodnie z algorytmem 3.5, generujemy listę podimplikantów o wadze $w = 1$ (krok 1 i 3) przedstawioną poniżej:

$$Sub_1 = \{p_1 = x_1, p_2 = x_2, p_3 = x_3, p_4 = x_4, p_5 = x_5, p_6 = x_6, p_7 = x_7\}.$$

Następnie dla wszystkich podimplikantów ze zbioru Sub_1 weryfikujemy warunki tw. 3.6. Przykładowo, dla podimplikantu p_3 otrzymujemy:

$$m^*(p_3) = \{m_{345}, m_{326}, m_{356}, m_{357}\}.$$

Analizując warunek tw. 3.6 dla wszystkich wyjść wykazujemy, że jest on spełniony dla z_1 . Usuwamy więc z sumy z_1 wszystkie implikanty zawierające podimplikant p_3 , dodając w ich miejsce implikant p_3 . Zmodyfikowana funkcja przedstawiona jest poniżej:

$$\begin{aligned} z_1 &= 145 + 157 + 247 + 256 + 3 \\ z_2 &= 146 + 147 + 156 + 245 + 246 + 257 + 345 + 347 + 356 + 357 \\ z_3 &= 145 + 146 + 147 + 256 + 257 + 346 \\ z_4 &= 156 + 157 + 245 + 246 + 247 + 347 \end{aligned}$$

Po zakończeniu analizy dla podimplikantów o wadze $w = 1$, generowana jest nowa lista podimplikantów o wadze $w = w + 1$ dla, której weryfikowane są warunki tw. 3.6.

Analizując warunki tw. 3.6 dla pozostałych podimplikantów otrzymamy funkcję układu w postaci:

$$\begin{aligned} z_1 &= 3 + 145 + 256 + 157 \\ z_2 &= 16 + 27 + 24 + 35 + 147 \\ z_3 &= 14 + 346 + 256 + 257 \\ z_4 &= 24 + 347 + 156 + 157 \end{aligned}$$

Złożoność układu przed minimalizacją wynosiła 75 literali, natomiast po minimalizacji 57 uzyskano więc redukcję rozmiaru układu o 25%. Na mocy tw. 3.6 uzyskany układ jest ST.

3.2.4 Podsumowanie

W rozdziale tym podano algorytm minimalizacji układów TSC/STC, stanowiącą rozwinięcie metody z pracy [22]. W porównaniu z [22] nie ograniczono się do eliminacji zmiennych, dla których układ nie był ST lecz zaproponowano algorytm minimalizacji funkcji układu i wykazano, że zminimalizowany układ jest z definicji ST. Ponadto metoda z [22] ograniczała się wyłącznie do kodów m/n , natomiast metoda podana tutaj stosuje się do układów o wejściach oraz wyjściach zakodowanych dowolnymi kodami nieuporządkowanymi.

Rozdział 4

Automatyczna synteza TSC układów sekwencyjnych

W TYM rozdziale przedstawione zostaną zagadnienia automatycznej syntezy TSC SM oraz podana zostanie metoda automatycznej syntezy TSC SM o strukturze TSC/CD i TSC/STC. Omówione zostaną także algorytmy kodowania wejść, stanów wewnętrznych oraz wyjść gwarantujące istnienie układów STC dla zastosowanych kodów. Tematyka ta zostanie zilustrowana na przykładzie syntezy wybranego TSC SM.

Przedstawione w rozdziałach 2 i 3 metody syntezy TSC układów kombinacyjnych nadają się do automatycznej syntezy TSC SM. Rozważane są dwie klasy układów: TSC/CD oraz TSC/STC. W pierwszej klasie, przedstawionej na rys. 2.5, zakłada się, że układ realizujący funkcję przejść-wyjść jest samotestowalny (ST), odporny na uszkodzenia (FS) oraz kodowo rozłączny (CD). W drugiej klasie (rys. 2.6) zakłada się natomiast, że układ realizujący funkcję przejść-wyjść jest ST oraz FS natomiast nie jest CD. W obu klasach układów należy zapewnić istnienie układów STC dla kodu wyjściowego, natomiast dla układów TSC/STC trzeba także zapewnić istnienie układów STC dla kodu wejściowego oraz kodu stanów wewnętrznych.

Automatyczna synteza TSC SM wymaga rozwiązania szeregu problemów:

- analiza i weryfikacja specyfikacji układu,
- uwzględnienie wejściowych i wyjściowych stanów obojętnych,
- dobór kodów wejściowych i wyjściowych,
- dobór kodu stanów wewnętrznych,
- weryfikacja istnienia STC dla stosowanych kodów,
- synteza funkcji przejść i wyjść SM,
- weryfikacja własności ST oraz CD układu.

Ostatnie z wymienionych zagadnień zostało szeroko omówione w poprzednich rozdziałach, pozostałe natomiast wymagają dokładnej analizy, prezentowanej w kolejnych podrozdziałach.

4.1 Analiza specyfikacji układów

W tej pracy przyjętym formatem opisu SM jest format BLIF [83], w którym specyfikacja SM podana jest w formie tablicy przejść-wyjść. Format opisu układu nie gwarantuje jego poprawności, tym samym istnieje możliwość pojawienia się błędów specyfikacji; np.: liczba stanów wewnętrznych układu jest inna niż zadeklarowana w nagłówku. Sytuacja taka może prowadzić do błędów w syntezy układach lub wręcz może uniemożliwić syntezę. Zatem, w celu poprawnej syntezy układu, konieczna jest wstępna weryfikacja poprawności jego opisu. Opracowane algorytmy, które są stosunkowo proste i nie wymagają szerszego omówienia, umożliwiają wykrycie błędów specyfikacji (deklaracja liczby wejść, stanów wewnętrznych, wyjść, przejść jest różna od rzeczywistej liczby). Niektóre wykryte błędy specyfikacji układu są korygowane tak, aby możliwa była jego poprawna synteza (o ile nie prowadzi to do zmiany jego funkcjonalności).

Znacznie poważniejszym problemem są następujące błędy w wewnętrznej strukturze układu, które uniemożliwiają syntezę układów TSC:

- stały stan linii wejściowej lub wyjściowej,
- stan wewnętrzny, z którego nie ma powrotu, lub do którego układ nigdy nie dotrze.

4.2 Stany obojętne

W opisie szeregu wzorcowych SM[83] stosuje się wejściowe oraz wyjściowe stany obojętne (ang. *don't-care*) oznaczone symbolem $-$.

4.2.1 Wejściowe stany obojętne

Wejściowe stany obojętne nie dają możliwości wyboru stanu linii, lecz są konsekwencją uproszczenia specyfikacji SM i w trakcie syntezy układu wymagają rozwinięcia do pełnego opisu, co często wiąże się z dużą złożonością obliczeniową związaną z eksplozją kombinatoryczną. Przeanalizujmy przykładowe przejście pomiędzy stanami A oraz B , zapisane w tablicy przejść jako:

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & & & & \\ 0 & 0 & - & A & \rightarrow & B & \end{array}$$

Zapis ten oznacza, że układ, który jest w stanie A , zmieni stan wewnętrzny na B , jeżeli linie x_1x_2 będą w stanie 0, bez względu na stan linii x_3 . Wynika stąd, że zarówno słowo 000 jak i 001 spowoduje przejście układu ze stanu A do B . W celu syntezy układu TSC należy przeprowadzić modyfikację opisu układu w taki sposób, aby wyeliminować wejściowe stany obojętne poprzez powielenie słowa wejściowego, w którym występuje stan obojętny na linii x_k i przyjęciu dla jednego słowa $x_k = 0$ a dla drugiego $x_k = 1$. Otrzymujemy w ten sposób dwa słowa wejściowe: pierwsze, w którym $x_k = 0$ oraz drugie, w którym $x_k = 1$. Przykładowo, w wyniku eliminacji stanów obojętnych, ww. przejście zapisane będzie jako:

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & & & & \\ 0 & 0 & 0 & A & \rightarrow & B & \\ 0 & 0 & 1 & A & \rightarrow & B & \end{array}$$

W przypadku większej liczby stanów obojętnych w słowie, metodę trzeba zastosować dla każdego z nich. W wyniku otrzymujemy pełny opis funkcji przejść układu. Niestety podejście takie powoduje, , rozbudowę opisu układu, tym samym mogą pojawić się problemy związane z analizą własności otrzymanych układów. Modyfikacja taka jest jednak konieczna, gdyż każdemu ze słów wejściowych zostaną przyporządkowane odpowiednie bity kontrolne, zależne od użytego kodu detekcyjnego. Należy tutaj zaznaczyć, że nazwa "wejściowe stany obojętne" nie oddaje istoty problemu, gdyż faktycznie tzw. wejściowy stan obojętny stanowi tylko skróconą formę zapisu.

4.2.2 Wyjściowe stany obojętne

Wyjściowy stan obojętny oznacza, że stan danej linii wyjściowej może być ustalony jako 0 lub 1 w czasie syntezy układu. Własność ta ma szczególne znaczenie w syntezie układów TSC, gdyż od odpowiedniego doboru stanu wyjściowych linii obojętnych może zależeć rozmiar słowa kodu wyjściowego oraz, co ważniejsze, istnienie i złożoność STC dla użytego kodu wyjściowego. W literaturze jedynie w pracy [39] poruszono kwestię wyjściowych stanów obojętnych. Wszyscy autorzy przyjmują, że wszystkim wyjściowym stanom obojętnym odpowiadają 0.

Dla przykładu przeanalizujemy zbiór słów wyjściowych Z , w którym występują następujące stany obojętne:

	z_1	z_2	z_3	z_4
Z_1	–	1	–	–
Z_2	0	0	–	–
Z_3	1	–	–	0
Z_4	–	–	1	0

Ustalając arbitralnie 0 w miejscu stanów obojętnych, otrzymujemy następujące słowa kodowe:

	z_1	z_2	z_3	z_4		z_1	z_2	z_3	z_4
Z_1	–	1	–	–	→	0	1	0	0
Z_2	0	0	–	–	→	0	0	0	0
Z_3	1	–	–	0	→	1	0	0	0
Z_4	–	–	1	0	→	0	0	1	0

Aby uzyskać wyjściowy systematyczny kod nieuporządkowany, musimy dodać jedną linię z_5 , celem uzyskania kodu 1/5:

	z_1	z_2	z_3	z_4		z_1	z_2	z_3	z_4	z_5
Z_1	0	1	0	0	→	0	1	0	0	0
Z_2	0	0	0	0	→	0	0	0	0	1
Z_3	1	0	0	0	→	1	0	0	0	0
Z_4	0	0	1	0	→	0	0	1	0	0

Niestety, uzyskany kod jest niekompletny (brak słowa 00100) i dla kodu tego nie istnieje układ STC. Co więcej, zastosowanie jakiegokolwiek innego kodu nieuporządkowanego również nie zagwarantuje istnienia STC, ponieważ linia wyjściowa z_4 ma stały stan 0.

Zauważmy, że po przyjęciu stanu linii $z_4 = 1$ w Z_2

	z_1	z_2	z_3	z_4		z_1	z_2	z_3	z_4
Z_1	–	1	–	–	→	0	1	0	0
Z_2	0	0	–	–	→	0	0	0	1
Z_3	1	–	–	0	→	1	0	0	0
Z_4	–	–	1	0	→	0	0	1	0

nie tylko liczba linii nie ulega zmianie, ale również otrzymujemy kompletny nieuporządkowany kod wyjściowy 1/4, dla którego znany jest układ STC.

Zagadnienie doboru kodu wyjściowego stanowi otwarty problem i nie są znane w literaturze metody konstrukcji kodów wyjściowych, które gwarantują istnienie STC. Metody automatycznego doboru takiego kodu dla zadanej specyfikacji wyjść zostaną pokazane w następnym podrozdziale.

Gwarancja istnienia STC dla wejść i stanów wewnętrznych poprzez dobór kodów

Obok stosunkowo prostych problemów związanych z analizą specyfikacji układu, kluczowe znaczenie dla syntezy TSC SM ma zagadnienie kodowania wejść, stanów wewnętrznych oraz wyjść układu. Tutaj klasa kodów detekcyjnych ograniczona jest oczywiście do kodów nieuporządkowanych.

Przypomnijmy, że żadna z podanych w literaturze metod syntezy TSC SM, nie gwarantuje istnienia układów STC dla zastosowanych kodów. Ponieważ autorzy nie weryfikują własności ST układów STC dla zastosowanych kodów, stąd gdy STC nie istnieją lub nie są ST, cały SM również nie jest TSC. W rozdziale tym podane zostaną metody kodowania zapewniające istnienie układów STC i weryfikujące ich własności ST dla kodów niekompletnych.

4.3.1 Kodowanie wejść

Kodowanie wejść SM może być dokonane dopiero po wygenerowaniu pełnego opisu układu, tj. po rozwinięciu wszystkich wejściowych stanów obojętnych. Do kodowania wejść mają zastosowanie wszystkie ze znanych systematycznych kodów nieuporządkowanych (kody niesystematyczne m/n w bardzo ograniczonym zakresie). Trudno podać uniwersalną metodę doboru kodu wejściowego, który zależy od wybranej struktury układu (TSC/CD lub TSC/STC) i zbioru wejściowych słów kodowych. Dla układów TSC/CD można kodować wejścia kodem Bergera pod warunkiem, że otrzymany układ funkcjonalny jest CD (nie ma konieczności stosowania STC dla kodu wejściowego), w przeciwnym wypadku należy zakodować wejścia kodem z inwersyjnym powtórzeniem. W układach TSC/STC wejścia można kodować kodem Bergera przy zapewnieniu istnienia STC dla wybranego kodu, natomiast w przypadku braku takiego STC, należy zakodować wejścia kodem z inwersyjnym powtórzeniem, dla którego łatwiej podać STC. Gdy kod wejściowy jest niekompletny, konieczna jest weryfikacja własności ST układów STC, która w opracowanym pakiecie oprogramowania TSCSM/CAD jest wykonywana z wykorzystaniem symulatora uszkodzeń.

4.3.2 Kodowanie stanów wewnętrznych

Do kodowania stanów wewnętrznych TSC SM najczęściej stosuje się kody m/n , a wśród nich kody 1/ n oraz 2/ n . Głównymi czynnikami wpływającymi na wybór kodu jest liczba stanów wewnętrz-

nych SM oraz pojemność kodu, zgodnie z zależnością $|S| \leq |C_{kod}|$, gdzie $|S|$ oznacza liczbę stanów wewnętrznych, a $|C_{kod}|$ pojemność kodu. Dla układów o małej liczbie stanów wewnętrznych (< 16) można wykorzystać kody $1/n$. W układach większych rozmiar słowa stanu dla kodu $1/n$ byłby zbyt duży, co powodowałoby zwiększenie rozmiaru układu spowodowane koniecznością zastosowania dodatkowych przerzutników, w takim przypadku mają zastosowanie kody $2/n$. Przy wyborze kodu należy zapewnić, aby zachodziła następująca zależność:

$$|T_{min}| \leq |S| \leq |C_{kod}|,$$

gdzie T_{min} minimalny zbiór słów kodowych zapewniających ST układu STC, natomiast $|T_{min}|$ jego pojemność. Samotestowalność STC można łatwo zapewnić przyporządkowując w pierwszej kolejności słowa kodowe z T_{min} .

4.4 Algorytm kodowania wyjść gwarantujący istnienie STC

Kodowanie wyjść SM stanowi poważny problem obliczeniowy, głównie z powodu rozmiaru słowa wyjściowego, małej liczby użytych słów kodowych oraz występowania stanów obojętnych. Z prac, w których rozważano zagadnienia kodowania wyjść TSC SM można wymienić prace Lai i Wey [39], [40] oraz Lo [43], gdzie rozważano zagadnienie kodowania. Autorzy proponują stosowanie kodu z inwersyjnym powtórzeniem, kodów m/n lub konkatenacji tych kodów. Podstawowym mankamentem przedstawionych tam metod jest brak weryfikacji istnienia STC dla zastosowanych kodów: większość stosowanych tam układów STC nie jest ST dla podzbioru użytych słów kodowych.

W przypadku SM o dużej liczbie linii wyjściowych s i/lub relatywnie małej liczbie stanów wyjściowych w stosunku do liczby wszystkich możliwych kombinacji 2^s , dobór kodu wyjściowego zapewniającego ST dla użytego STC stanowi poważny problem. Przeprowadzona przez nas analiza zbioru słów wyjściowych wielu układów wzorcowych ze zbioru ISCAS89 wykazała niemożność lub nieopłacalność zastosowania kodów z inwersyjnym powtórzeniem lub kodów Bergera. Dla kodu z inwersyjnym powtórzeniem może wystąpić znaczny nadmiar sprzętowy wynikający z podwojenia liczby wyjść. Zastosowanie kodu Bergera uniemożliwia relatywnie dużą złożoność STC oraz problemy z zapewnieniem ich ST dla kodów niekompletnych (często występujących w analizowanych układach). Dlatego rozwiązania należy szukać w systematycznych nieuporządkowanych kodach konstruowanych ad hoc. W pracach [39], [40] przedstawiono algorytm umożliwiający podział słów kodu wyjściowego na kod stanowiący konkatenację kodów $1/n$ oraz kodu z inwersyjnym powtórzeniem. Autorzy zaproponowali metodę podziału kodu wyjściowego na grupy oraz kodowania w taki sposób, aby słowa w otrzymanych grupach tworzyły kompletny kod $1/n$. W rozwiązaniu tym usunięto stany obojętne, przyjmując dla każdego z nich wartość 0, oraz, co najważniejsze, nie zweryfikowano istnienia STC dla otrzymanego kodu.

Przedstawiony poniżej algorytm jest modyfikacją algorytmu z pracy [39], który nie tylko pozwala wykorzystać, stany obojętne ale i gwarantuje ST układów STC dla wykorzystywanego zbioru słów kodowych (m. in. dzięki umiejętnemu wykorzystaniu stanów obojętnych). Podejście takie nie było do tej pory nigdy stosowane w żadnej ze znanych w literaturze metod syntezy TSC SM.

Rozważmy zbiór słów wejściowych C_{Out}^{nc} , który należy zakodować w taki sposób, aby otrzymać kod nieuporządkowany. Problemem, który należy rozwiązać, jest podział i zakodowanie słów z C_{Out}^{nc} kodem

nieuporządkowanym, przy jak najmniejszym nadmiarze informacyjnym oraz zapewnieniu istnienia STC dla otrzymanego kodu.

Przyjmijmy następujące oznaczenia: X_i^{nc} — niezakodowane słowo wyjściowe ($X_i^{nc} \in C_{Out}^{nc}$), oraz G_i — grupa kolumn wydzielona z kodu C_{Out}^{nc} .

Definicja 4.1 Słowo X_i^{nc} ze zbioru C_{Out}^{nc} nazywamy słowem istotnym, jeżeli

$$(\forall X_k^{nc} \in C_{Out}^{nc}, X_k^{nc} \neq X_i^{nc})(X_k^{nc} \leq X_i^{nc}).$$

Słowa istotne są to słowa, które nie są pokrywane przez żadne inne słowa kodu.

Przedstawiony poniżej algorytm umożliwia podział zbioru słów C_{Out}^{nc} w taki sposób, aby stanowił konkatenację grup, w których słowa mają wagę równą lub mniejszą 1.

Definicja 4.2 Wagą kolumny nazywamy liczbę jedynek występujących w danej kolumnie.

Dla kodu C_{Out}^{nc} generujemy listę słów istotnych i rosnąco sortujemy kolumny względem ich wagi (jeżeli kolumny mają taką samą wagę porównujemy indeksy i kolumn). Na tym etapie analizy stany obojętne traktowane są jako 0. Większa waga kolumny oznacza, że należy ona do większej liczby słów istotnych. Kolumna o mniejszej wadze ma większy priorytet wyboru przy podziale na grupy G_i . Oznaczmy przez K_i i -tą kolumnę a przez $K = \{K_{q1}, \dots, K_{qt}\}$ posortowany zbiór wszystkich K_i w kodzie C_{Out}^{nc} .

Algorytm 4.1 Algorytm podziału kodu C_{Out}^{nc} na grupy G_i

- 1 Przyjąć $u = 0$, $S = K$.
- 2 Dodać do grupy G_u pierwszy element (kolumnę) ze zbioru S .
- 3 Dla wszystkich słów istotnych X_i sprawdzić, czy bit w wybranej i -tej kolumnie równy jest 1.
- 4 Jeżeli warunek 3 jest spełniony, to ze zbioru S usunąć wszystkie kolumny,
- 5 w których bity słowa $X_i = 1$.
- 6 Jeżeli $S = \emptyset$, proces wyboru grupy G_u uznać za zakończony;
- 7 w przeciwnym przypadku powtórzyć kroki 2–6 dla kolejnego elementu ze zbioru S .
- 8 Usunąć ze zbioru K kolumny z grupy G_u oraz przyjąć $S = K$ oraz $u = u + 1$.
- 9 Jeżeli $K = \emptyset$, proces podziału jest zakończony; w przeciwnym wypadku przejść do kroku 2.

W wyniku działania algorytmu otrzymujemy u grup, których konkatenacja składa się na kod C_{Out}^{nc} .

Przykład 4.1 Przeanalizujmy algorytm na przykładzie układu mc ze zbioru układów wzorcowych ISCAS89. Poniżej przedstawiono kod wyjściowy układu mc zawierający 8 słów, z których 2 są słowami istotnymi.

Słowa kodu wyjściowego układu mc

Słowa kodu	z_1	z_2	z_3	z_4	z_5
Z_1	0	0	0	1	0
Z_2	0	0	1	1	0
Z_3	0	1	0	0	0
Z_4	0	1	0	0	1
Z_5	1	0	0	1	0
Z_6	1	0	1	1	0
Z_7	1	1	0	0	0
Z_8	1	1	0	0	1

Słowa istotne kodu wejściowego układu mc

Słowa istotne	z_1	z_2	z_3	z_4	z_5
Z_6	1	0	1	1	0
Z_8	1	1	0	0	1

W wyniku sortowania kolumn słów istotnych względem ich wagi otrzymujemy:

Słowa istotne	z_2	z_3	z_4	z_5	z_1
Z_6	0	1	1	0	1
Z_8	1	0	0	1	1

Przyjmujemy więc zbiory $K = \{K_2, K_3, K_4, K_5, K_1\}$ oraz $S = K$ (krok 1 alg. 4.1). Podział kodu na grupy rozpoczynamy przyjmując $G_1 = \{K_2\}$, ponieważ K_2 jest pierwszym elementem ze zbioru S . Następnie znajdujemy słowa istotne, które w kolumnie K_2 mają wartość 1 — warunek ten jest spełniony tylko dla słowa $Z_8 = 11001$. Następnie ze zbioru S usuwamy kolumny K_1 , K_2 i K_5 otrzymując $S = \{K_3, K_4\}$. Ponieważ $S \neq \emptyset$, do zbioru G_1 dodajemy kolumnę K_3 i usuwamy z S kolumny, w których bity słowa X_6 mają wartość 1. W wyniku otrzymujemy $S = \emptyset$, co kończy proces wyboru grupy G_1 . Następnie ze zbioru K usuwamy wszystkie kolumny wybrane do grupy G_1 otrzymując $K = \{K_4, K_5, K_1\}$. Przyjmujemy teraz $S = K$ i rozpoczynamy wybór grupy G_2 w powyższy sposób. Proces wyboru powtarzamy aż do czasu, gdy $K = \emptyset$. W wyniku otrzymujemy trzy grupy: $G_1 = \{K_2, K_3\}$, $G_2 = \{K_4, K_5\}$ oraz $G_3 = \{K_1\}$.

W celu wyboru kodu dla kodowania wyjść przeanalizujemy kombinacje bitów, które pojawiają się w poszczególnych grupach.

Grupa G_1		Proponowane kodowanie			
z_2	z_3	z_2	z_3	z_6	z_7
0	0	0	0	0	1
0	0	0	0	1	0
0	1	0	1	0	0
1	0	1	0	0	0

Grupa G_2		Proponowane kodowanie			
z_4	z_5	z_4	z_5	z_8	z_9
0	0	0	0	0	1
0	0	0	0	1	0
0	1	0	1	0	0
1	0	1	0	0	0

Grupa G_3	Proponowane kodowanie	
	z_1	z_{10}
0	0	1
1	1	0

Dla grup G_1 oraz G_2 dodajemy po dwie dodatkowe linie, otrzymując kod 1/4, a dla G_3 dodajemy jedną linię, otrzymując kod z inwersyjnym powtórzeniem. W grupy G_1 oraz G_2 słowo 00 kodujemy dla części wyjść jako 0001, a dla pozostałych jako 0010. Pozostaje teraz weryfikacja istnienia STC dla otrzymanego kodu (konkatenacja kodów 1/4, 1/4 oraz 1/2). Dla układu mc oraz przyjętego kodowania na wejściach zastosowanych STC mamy kody kompletne, więc układy te są ST. ■

Metoda doboru kodu wyjściowego

W wyniku podziału kodu C_{In}^{nc} otrzymujemy u grupy G_i , w których waga słów $w(G_i)$ wynosi 1 lub 0. Grupy uzyskane w wyniku podziału można podzielić na dwie główne klasy:

1. Grupy o liczbie bitów w słowie większej od 1:
 - (a) waga wszystkich słów w grupie równa 1,
 - (b) waga wszystkich słów w grupie mniejsza lub równa 1.
2. Grupy o liczbie bitów w słowie równej 1.

W przypadku 1(a) otrzymana grupa jest zakodowana kodem 1/ n , gdzie n jest liczbą kolumn w grupie. W przypadku 1(b) konieczne jest rozszerzenie rozmiaru grupy o jedną kolumnę w taki sposób, aby waga każdego słowa w otrzymanym kodzie wynosiła 1. Dla otrzymanych kodów STC istnieją, a kompletność zbioru słów kodowych zapewnia ich ST. W przypadku, gdy otrzymany kod jest kodem 1/3, należy zastosować STC dla kodu 2/5 podany w rozdziale 2.4 lub też zamiast kodu 1/3 zastosować kod 1/4, w taki sposób, aby parze niepustych podzbiorów rozłącznych słów 00 w grupie odpowiadały słowa 0001 kodu 1/4, i słowo 0010 kodu 1/4.

Dla układów, w których istnieją kolumny, których nie można zgrupować z żadną inną grupą, otrzymujemy grupy o liczbie bitów równej 1 (przypadek 2), które można oczywiście zakodować kodem 1/2, co jednak może prowadzić do niepotrzebnego powiększenia układu. W szeregu przypadków istnieje możliwość zakodowania otrzymanych grup innymi kodami nieuporządkowanymi, zgodnie z podaną poniżej metodą, w której kluczowe znaczenie ma umiejętne wykorzystanie wyjściowych stanów obojętnych.

W pierwszym kroku łączymy grupy o liczbie kolumn 1 w jedną grupę. Przykładowo, w wyniku połączenia takich grup dla układu **kirkman**, otrzymamy grupę G_p przedstawioną w tabeli 4.1. W kolumnie pierwszej podane jest słowo kodu, natomiast w drugiej liczba przejść w SM, dla których dane słowo pojawia się na wyjściu układu. Zauważmy, że w kilku słowach wyjściowych pojawiają się stany obojętne, które mogą zostać wykorzystane do zapewnienia kompletności otrzymanego kodu nieuporządkowanego. W tabeli 4.2 podana jest dla porównania lista słów, które otrzymamy przyjmując dla stanów obojętnych wartość 0 (zgodnie z rozwiązaniem z pracy [39]). Przeanalizujemy możliwość zakodowania otrzymanej grupy kodem Bergera $C_{(I,K)}$. W rozwiązaniu przyjmującym stany obojętne równe 0 otrzymamy niekompletny kod $C_{(3,2)}$, składający się z 5 słów, podczas gdy znane STC wymagają co

Tablica 4.1: Lista słów grupy G_p w układzie **kirkman** uwzględniająca stany obojętne

Słowo kodu	Liczba wystąpień
-01	29
11-	8
-0-	32
-1-	32
000	8
010	67
100	22

Tablica 4.2: Lista słów grupy G_p w układzie **kirkman** (stany obojętne równe 0)

Słowo kodu
000
001
010
110
100

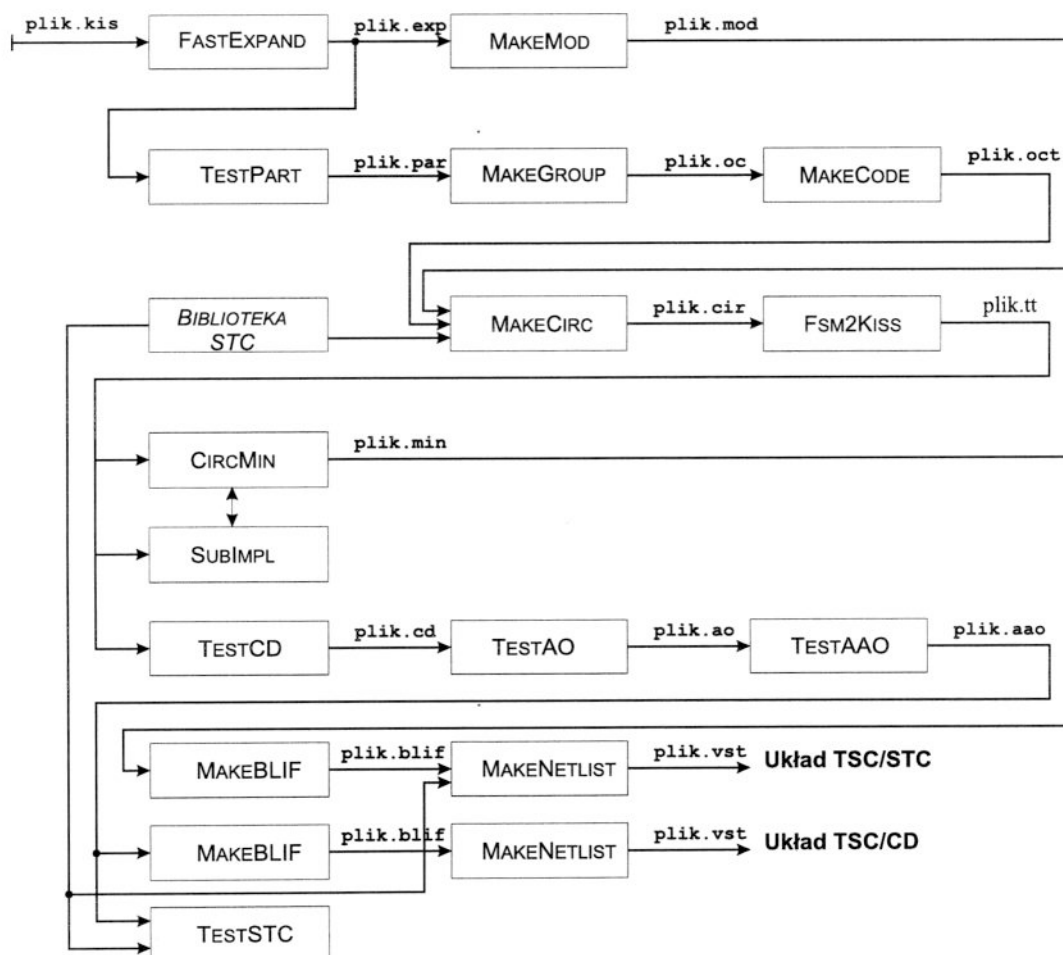
najmniej 6 słów kodowych (tabela 2.3). Łatwo zauważyć, że do kompletności kodu wyjściowego brakuje trzech słów: 011, 101 oraz 111. Zauważmy, że słowo 11– wystąpiło w 8 przejściach, nie ma więc przeszkody, aby dla każdego przejścia wartość stanów obojętnych przyjąć w inny sposób. W omawianym przykładzie dla części wyjść 11– przyjmujemy 110, a dla pozostałych przyjmujemy 111. W ten sposób zbiór słów, które pojawiają się na wyjściu został rozszerzony o słowa 110 oraz 111. W wyniku podobnej analizy stanów obojętnych dla pozostałych słów wyjściowych otrzymamy kompletny zbiór słów przedstawiony w tabeli 4.3, które można zakodować kompletnym kodem Bergera $C_{(3,2)}$. Metoda ta ma oczywiście zastosowanie dla wszystkich kodów nieuporządkowanych. W przypadku, gdy nie ma możliwości konstrukcji kodu kompletnego, należy zweryfikować, czy użyty STC jest ST dla kodu niekompletnego. Na wyjściu zastosowanych układów STC otrzymujemy konkatencję kodów 1/2 czyli kod C_{2r} . Należy zweryfikować, czy dla kodu tego istnieje STC zgodnie z algorytmem z [57].

System automatycznej syntezy TSC układów sekwencyjnych Mealy’ego

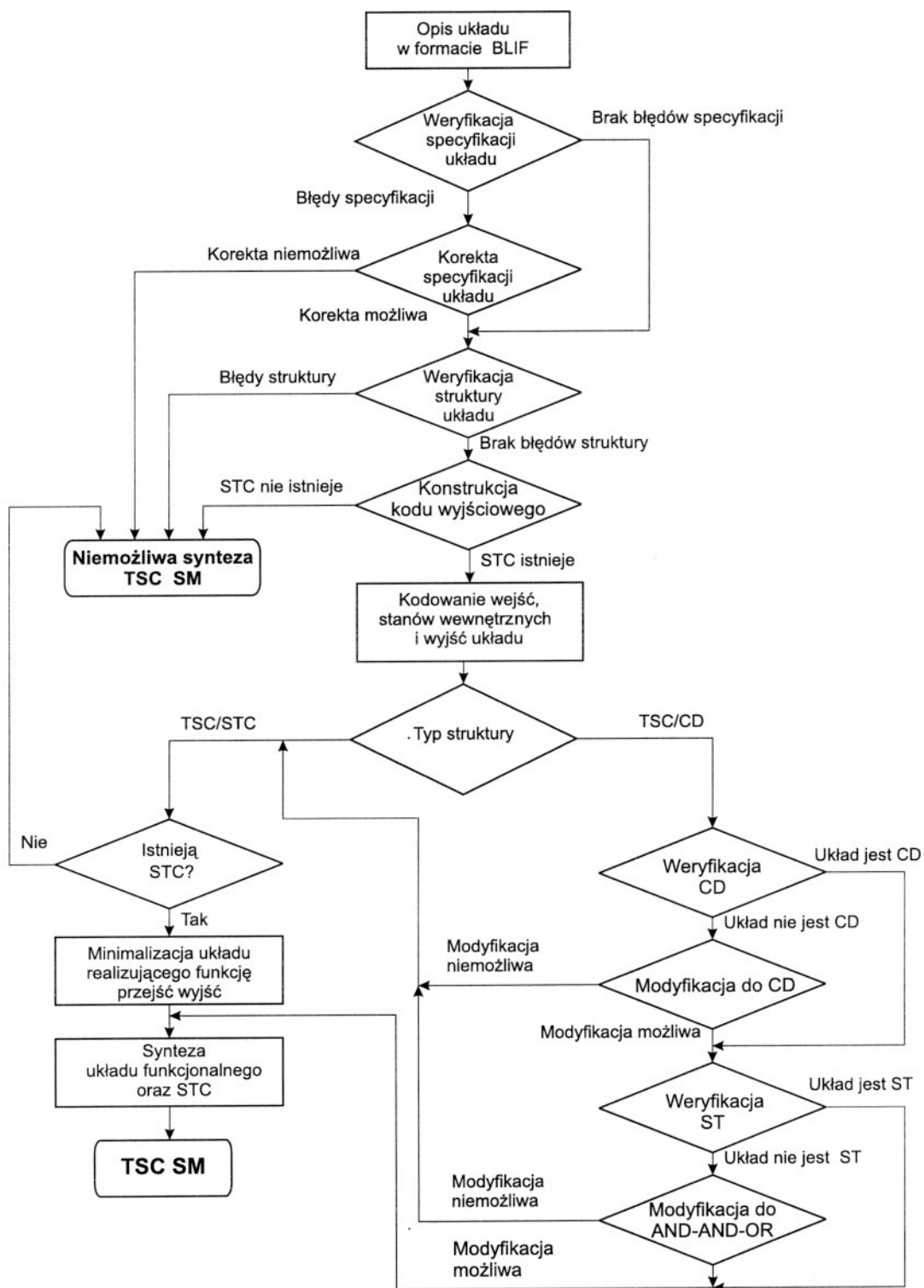
Przyjmijmy, że wejścia, stany wewnętrzne oraz wyjścia SM są zakodowane kodami nieuporządkowanymi. Kod wejściowy układu kombinacyjnego **H** realizującego funkcję przejść-wyjść powstaje poprzez konkatencję słów kodów wejściowego i kodu stanów wewnętrznych SM, natomiast kod wyjściowy jest tworzony przez konkatencję słów kodów wyjściowego i kodu stanów wewnętrznych. Ponieważ układ

Tablica 4.3: Kompletny zbiór słów kodu wyjściowego dla układu kirkman

Słowo kodu z uwzględnieniem stanów obojętnych	Słowo kodu po wyeliminowaniu stanów obojętnych	Część kontrolna kodu Bergera
-01	001	10
-01	101	01
11-	110	01
11-	111	00
-0-	100	10
-1-	010	10
-1-	011	01
000	000	11
010	010	10
100	100	10



Rys. 4.1: Schemat syntezy TSC SM



Rys. 4.2: Schemat blokowy systemu TSCSM/CAD syntezy TSC SM

kombinacyjny **H** odwzorowuje wejściowy kod nieuporządkowany w wyjściowy kod nieuporządkowany, można go zrealizować jako układ bezinwerterowy o strukturze 2-poziomowej AND-OR (podrozdział 2.5). Zatem problem syntezy TSC SM sprowadza się do syntezy układu **H** w wersji TSC.

Funkcja realizowana przez układ **H** może być zaimplementowana w trojaki sposób, zależny od wyboru struktury układu TSC/CD, TSC/ES lub TSC/STC. W zagadnieniach weryfikacji własności układów TSC/CD oraz TSC/ES zastosowanie mają algorytmy 3.1, 3.2, 3.3 i 3.4, natomiast w przypadku TSC/STC algorytm 3.5 eliminacji zmiennych. Należy tutaj zaznaczyć, że dla struktury TSC/STC konieczne jest zastosowanie trzech układów STC dla wszystkich kodów (kodu wejściowego, kodu stanów wewnętrznych oraz kodu wyjściowego), a dla TSC/CD i TSC/ES tylko jednego STC dla kodu wyjściowego.

Podane w pracy metody umożliwiają syntezę TSC SM zgodnie z algorytmem podanym na rys. 4.2. Dane wejściowe opisujące SM w formacie BLIF są wstępnie weryfikowane ze względu na możliwe błędy specyfikacji. W miarę możliwości wykryte błędy są korygowane, a w przypadku niemożności korekty synteza układu jest przerywana. Następnie testowana jest struktura układu poprzez weryfikację grafu przejść na okoliczność występowania nieosiągalnych stanów wewnętrznych lub takich, z których nie ma powrotu. Ponadto sprawdzane jest występowanie linii wejściowych i wyjściowych o stałym stanie. Wystąpienie któregokolwiek z powyższych błędów czyni syntezę niemożliwą. Dla układu, który spełnia wstępne kryteria (brak błędów specyfikacji i struktury), kodowane są wejścia, stany oraz konstruowany jest kod wyjściowy, dla którego istnieje STC. Brak takiego układu także uniemożliwia syntezę SM, który jest TSC dla wszystkich uszkodzeń pojedynczych typu s/z .

Kolejnym etapem syntezy jest wybór struktury TSC/CD lub TSC/STC. W pierwszym przypadku są weryfikowane własności CD oraz ST. W przypadku gdy układ nie jest CD i/lub ST, jest on, o ile to możliwe, modyfikowany zgodnie z algorytmami 3.2 i/lub 3.4. Jeżeli uzyskany układ jest CD i ST kończymy syntezę układu TSC/CD. W niektórych przypadkach można zrezygnować z własności CD na rzecz układów bezpiecznych dla błędów wejściowych (ES), pod warunkiem, że dla przyjętego zastosowania układu własność ES jest akceptowalna (układ maskuje część błędów wejściowych). Wówczas można pominąć modyfikację układu w celu uzyskania CD. W wyniku otrzymujemy strukturę TSC/ES, która często cechuje się mniejszą złożonością niż analogiczny układ TSC/CD.

W przypadku niemożności modyfikacji zapewniającej CD i/lub ST, syntezerowany jest układ o strukturze TSC/STC, dla którego weryfikowana jest ST układów STC dla wybranych kodów wejściowego oraz wyjściowego (nie spełnienie tego warunku uniemożliwia syntezę układu TSC). Następnie syntezerowany jest układ **H** realizujący funkcję przejść/wyjść, który jest minimalizowany zgodnie z algorytmem 3.5. Układ **H** wraz z układami STC składa się na TSC/STC SM.

Opracowane algorytmy i metody syntezy samosprawdzalnych SM zostały zaimplementowane w formie pakietu oprogramowania TSCSM CAD (Totally Self-Checking Sequential Machines CAD). Implementacji dokonano w języku C++ (wersja gcc dla systemu Linux). Funkcje realizowane przez pakiet TSCSM CAD można podzielić na 7 grup zawierających łącznie 15 programów (rys.4.1):

1. Wstępna analiza i weryfikacja (programy: `fastexpand`, `makemod`).
2. Konstrukcja kodu wyjściowego i synteza wyjściowego układu STC (programy: `testpart`, `makegroup`, `makecode`, `makestc`).

Tablica 4.4: Lista modułów pakietu oprogramowania TSC/SM CAD

Nazwa programu	Opis realizowanych funkcji
fastexpand	Usunięcie wejściowych stanów obojętnych oraz weryfikacja opisu układu
makemod	Konwersja z formatu BLIF do wewnętrznego/formatu TSCSM CAD
testpart	Podział kodu wyjściowego na grupy
makegroup	Kodowanie grup
makecode	Synteza kodu wyjściowego
makestc	Synteza układu STC dla kodu wyjściowego
makecirc	Kodowanie wejść oraz wyjść układu
circmin	Minimalizacja i synteza układu TSC/STC
testcd	Weryfikacja CD układu AND-OR
testao	Weryfikacja ST układu AND-OR
testaao	Poprawa ST układu AND-OR poprzez dodanie warstwy bramek AND
makeblif	Synteza pliku BLIF opisującego układ
makenetlist	Przygotowanie listy połączeń elementów AND i OR
teststc	Weryfikacja ST układów STC

- Kodowanie wejść, stanów wewnętrznych i wyjść układu (program: `makecirc`).
- Synteza układu TSC/STC SM (programy: `circmin`, `subimpl`).
- Synteza układu TSC/CD SM (programy: `testcd`, `testao`, `testaao`).
- Synteza listy połączeń układu SM (programy: `makeblif`, `makenetlist`).
- Weryfikacja ST układów STC (program: `teststc`).

Na rys. 4.1 zaznaczono także nazwy plików, w których zapisywane są wyniki poszczególnych etapów syntezy układu. Natomiast w tabeli podano 4.4 bardziej szczegółową specyfikację funkcji realizowanych przez poszczególne programy.

4.6 Przykład syntezy TSC układu sekwencyjnego

Opracowane algorytmy umożliwiają kompletną syntezę TSC SM opisanego w formie tablicy przejść. Dla przykładu, przedstawiona zostanie synteza SM *mc*, ze zbioru układów wzorcowych ISCAS89, którego tablica przejść została przedstawiona w tabeli 4.5.

Układ ma 3 linie wejściowe, 4 stany wewnętrzne (HG, HY, FG, FY), 5 linii wyjściowych oraz 10 przejść. Przedstawiona tablica przejść opisuje układ o wejściach i wyjściach niezakodowanych kodem detekcyjnym i nieokreślonym kodzie stanów wewnętrznych. W celu eliminacji stanów obojętnych, należy przeprowadzić rozszerzenie kodu wejściowego, opisane w podrozdziale 4.2.1. W efekcie otrzymujemy układ, którego tablica przejść przedstawiona jest w tabeli 4.6. W wyniku eliminacji wejściowych

```

.i 3
.o 5
.p 10
.s 4
0-- HG HG 00010
-0- HG HG 00010
11- HG HY 10010
--0 HY HY 00110
--1 HY FG 10110
10- FG FG 01000
0-- FG FY 11000
-1- FG FY 11000
--0 FY FY 01001
--1 FY HG 11001
.end

```

Rys. 4.3: Opis układu mc w formacie KISS

stanów obojętnych liczba przejść wzrosła z 10 do 36. Wejścia tak zmodyfikowanego układu kodujemy kodem Bergera $C_{(3,2)}$ (liczba wejść zwiększa się w ten sposób z 3 do 5 linii (tabela 4.7)), a stany wewnętrzne układu kodem 1/4 (tabela 4.8).

Kod wyjściowy można skonstruować stosując algorytm 4.1 podziału kodu. W wyniku otrzymujemy 3 grupy: grupa G_1 zawiera kolumny 4 oraz 5, grupa G_2 kolumny 2 i 3, natomiast grupa G_3 zawiera kolumnę 1. Grupy G_1 , G_2 , zawierające po dwie kolumny, kodujemy kodem z 1/4, w taki sposób że części słów 00 odpowiadają słowa 0010 natomiast pozostałym 0001 kodu 1/4, natomiast ostatnią grupę G_3 , zawierającą jedną kolumnę, kodujemy kodem 1/2 (tabela 4.9). Dla tak przyjętego kodu wyjściowego istnieje STC. Reasumując, po zmodyfikowaniu układ ma 5 wejść, rozmiar słowa stanów wewnętrznych wynosi 4, a liczba wyjść wynosi 10.

Analiza układu z wykorzystaniem algorytm 3.1 wykazała, że jest on CD. Weryfikacja własności ST, zgodnie z warunkami tw. 3.4, wykazała 10 nietestowalnych uszkodzeń, które zostały wyeliminowane w wyniku modyfikacji struktury układu poprzez dodanie dodatkowej warstwy bramek AND, zgodnie z algorytmem 3.4. Następnie dokonano minimalizacji funkcji układu z użyciem pakietem SIS, co zredukowało liczbę literali w układzie z 448 do 121. Konieczność zastosowania STC dla kodu wyjściowego powoduje powiększenie złożoności układu o 58 literale, a zatem całkowita złożoność układu TSC/CD wynosi 179 literali.

Istnieje też możliwość syntezy układu o strukturze TSC/STC. W wyniku minimalizacji układu zgodnie z algorytmem 3.5 otrzymany układ jest ST na mocy tw. 3.7. Złożoność struktury otrzymanego układu po minimalizacji wynosi 81 literali. Dodatkowo złożoność układu powiększa się o trzy STC: dla kodów Bergera $C_{(3,2)}$, 1/4 i dla kodu wyjściowego, które wynoszą odpowiednio 38, 16 i 58 literali. Złożoność całego układu TSC wynosi więc 193 literali.

Przykład ten pokazuje, że synteza TSC układu mc jest możliwa zarówno w wersji TSC/CD jak

Tablica 4.5: Tablica przejść układu mc

Wejście	Stan początkowy	Stan końcowy	Wyjście
0 - -	HG	HG	0 0 0 1 0
- 0 -	HG	HG	0 0 0 1 0
1 1 -	HG	HY	1 0 0 1 0
- - 0	HY	HY	0 0 1 1 0
- - 1	HY	FG	1 0 1 1 0
1 0 -	FG	FG	0 1 0 0 0
0 - -	FG	FY	1 1 0 0 0
- 1 -	FG	FY	1 1 0 0 0
- - 0	FY	FY	0 1 0 0 1
- - 1	FY	HG	1 1 0 0 1

i TSC/STC. Złożoności uzyskanych układów są zbliżone: TSC/CD – 179 literali, TSC/STC – 196 literali. Należy zaznaczyć, że oba rozwiązania zapewniają 100% pokrycie uszkodzeń i błędów, czyli cel TSC został osiągnięty.

Tablica 4.6: Tablica przejść układu mc

Wejście	Stan początkowy	Stan końcowy	Wyjście
000	HG	HG	00010
000	HG	HG	00010
110	HG	HY	10010
000	HY	HY	00110
001	HY	FG	10110
100	FG	FG	01000
000	FG	FY	11000
010	FG	FY	11000
000	FY	FY	01001
001	FY	HG	11001
100	HG	HG	00010
100	HY	HY	00110
101	HY	FG	10110
110	FG	FY	11000
100	FY	FY	01001
101	FY	HG	11001
010	HG	HG	00010
010	HY	HY	00110
011	HY	FG	10110
010	FY	FY	01001
011	FY	HG	11001
110	HY	HY	00110
111	HY	FG	10110
110	FY	FY	01001
111	FY	HG	11001
001	HG	HG	00010
111	HG	HY	10010
101	FG	FG	01000
001	FG	FY	11000
011	FG	FY	11000
101	HG	HG	00010
111	FG	FY	11000
011	HG	HG	00010

Tablica 4.7: Kod wejściowy układu mc

Bity danych			Bity kontrolne	
x_1	x_2	x_3	x_4	x_5
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

Tablica 4.8: Kodowanie stanów wewnętrznych układu mc

Stan	x_6	x_7	x_8	x_9
HG	1	0	0	0
HY	0	1	0	0
FG	0	0	1	0
FY	0	0	0	1

Tablica 4.9: Kod wyjściowy układu mc

Słowa wyjściowe	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}
0 0 0 1 0	0	0	0	1	0	0	0	0	1	1
0 0 0 1 0	0	0	0	1	0	0	0	1	0	1
0 0 1 1 0	0	0	1	1	0	0	0	0	0	1
0 1 0 0 0	0	1	0	0	0	0	1	0	0	1
0 1 0 0 1	0	1	0	0	1	0	0	0	0	1
1 0 0 1 0	1	0	0	1	0	0	0	0	1	0
1 0 1 1 0	1	0	1	1	0	0	0	0	0	0
1 1 0 0 0	1	1	0	0	0	1	0	0	0	0
1 1 0 0 0	1	1	0	0	0	0	1	0	0	0
1 1 0 0 1	1	1	0	0	1	0	0	0	0	0

Rozdział 5

Wyniki syntezy sekwencyjnych układów wzorcowych

WŁASNOŚCI opracowanych algorytmów syntezy TSC SM zostały zbadane na zbiorze układów wzorcowych (ang. *benchmarks*). Zbiór układów poddanych weryfikacji opracowany został w roku 1991 i znany jest pod nazwą ISCAS'91 jako rozwinięcie poprzednich zbiorów ISCAS'89 z roku 1989 [7] oraz ISCAS'85 z roku 1985 [8]. Zbiory te zawierają przykładowe układy, które są wykorzystywane do oceny jakości oraz weryfikacji metod syntezy i algorytmów automatycznego generowania testów z dziedziny układów cyfrowych.

W tabeli 5.1 przedstawiono podstawowe dane analizowanych układów wzorcowych. Dane te dotyczą układów w formie niezmodyfikowanej, o wejściach stanach wewnętrznych i wyjściach niezakodowanych kodem detekcyjnym. Jak widać, zbiór ten zawiera układy o zróżnicowanym stopniu złożoności. Do najprostszych zaliczyć można *mc*, *dk27*, *ex5*, natomiast do grupy układów o znacznym stopniu złożoności zalicza się *styr*, *ex1*, *keyb*, *kirkman*. Układy opisane są w formacie BLIF podformacie KISS [83]. Dla szeregu układów wstępna weryfikacja poprawności opisu wykazała pewne nieścisłości (brak deklaracji liczby stanów wewnętrznych i/lub przejść, brak znacznika końca pliku), które zostały skorygowane. Tabela 5.1 zawiera dane już zweryfikowane i poprawione.

5.1 Metoda badań

Zbadano własności dwóch rodzajów TSC układów sekwencyjnych TSC/CD: oraz TSC/STC. W obu przypadkach w pierwszym kroku syntezy weryfikowano strukturę układu na okoliczność błędów uniemożliwiających syntezę układów w wersji TSC oraz minimalizowano liczbę stanów wewnętrznych z wykorzystaniem algorytmu *stamina* z pakietu SIS. Wejścia układów kodowano kodem z inwersyjnym powtórzeniem lub kodem Bergera, natomiast stany wewnętrzne kodowano zgodnie z metodą podaną w podrozdziale 4.3.2. W celu porównania przydatności i efektywności kodów syntezerowano układy o stanach wewnętrznych zakodowanych zarówno kodami $1/n$ jak i kodami $2/n$. Następnie konstruowany był kod wyjściowy oraz układ STC dla tego kodu.

W kolejnym kroku syntezerowano dla przyjętego kodowania układ realizujący funkcję przejść/wyjść SM. Dla układów o strukturze TSC/CD weryfikowane były własności CD oraz ST zgodnie z algorytmami 3.1 i 3.3. Jeżeli było to konieczne, układy były modyfikowane w celu uzyskania CD i/lub

Tablica 5.1: Charakterystyka układów wzorcowych ISCAS'89

Układ	Liczba wejść	Liczba wyjść	Liczba przejść	Liczba stanów wewnętrznych	Liczba stanów po minimalizacji
bbara	4	2	60	10	7
bbsse	7	7	56	16	13
bbtas	2	2	24	6	6
beecount	3	4	28	7	4
cse	7	7	91	16	16
dk14	3	5	56	7	7
dk15	3	5	32	4	4
dk16	2	3	108	27	27
dk17	2	3	32	8	8
dk27	1	2	14	7	7
dk512	1	3	30	15	15
donfile	2	1	96	24	1
dvram	8	15	47	6	35
ex1	9	19	138	20	18
ex2	2	2	72	19	14
ex3	2	2	36	10	5
ex4	6	9	21	14	14
ex5	2	2	32	9	4
ex6	5	8	34	8	8
ex7	2	2	36	10	4
fetch	9	15	33	26	24
keyb	7	2	170	19	19
kirkman	12	6	370	17	17
lion9	2	1	25	9	4
lion	2	1	11	4	4
log	9	24	29	17	17
mark1	5	16	22	16	12
mc	3	5	10	4	4
modulo12	1	1	24	12	1
opus	5	6	22	11	9
planet1	7	19	115	48	48
planet	7	19	115	48	48
rie	9	26	49	5	9
s1	8	6	107	20	20
s1a	8	6	107	20	1
s8	4	1	20	5	1
shiftreg	1	1	16	8	8
sse	7	7	56	16	13
styr	9	10	166	30	30
tav	4	4	49	4	4
tbk	6	3	1569	32	16
train11	2	1	25	11	4
train4	2	1	14	4	4

ST algorytmami 3.2 i 3.4. Natomiast układy o strukturze TSC/STC były minimalizowane zgodnie z algorytmem 3.5.

W ostatnim kroku syntezerowano cały układ TSC SM, składający się z układu realizującego funkcję przejść/wyjść, rejestru oraz zastosowanych STC. Następnie obliczano złożoność otrzymanych układów z wykorzystaniem pakietu SIS. Przy ocenie złożoności uwzględniono rejestry, traktując każdy przerzutnik jako 3 literale.

5.2 Analiza struktury układów i minimalizacja liczby stanów wewnętrznych

Analiza struktury wewnętrznej następujących układów wykazała następujące błędy uniemożliwiające synteze układów TSC SM, układy `donfile`, `modulo12`, `s1a` oraz `s8` mają stałe wyjścia, co uniemożliwia synteze układu jako TSC, natomiast w układach `ex2` oraz `dk512` występują stany wewnętrzne, do których układ nie może dotrzeć.

Dla pozostałych układów zminimalizowano liczbę stanów wewnętrznych algorytmem `stamina` z pakietu SIS, co doprowadziło dla szeregu układów do zmniejszenia liczby stanów wewnętrznych (tabela 5.1). Jedynie dla układu `cse` niemożliwe było przeprowadzenie minimalizacji liczby stanów wewnętrznych z powodu błędu w programie SIS. Ponieważ jego specyfikacja nie zawierała błędów w opisie struktury wewnętrznej, dlatego do dalszej analizy wykorzystano układ niezminimalizowany.

5.3 Kodowanie wejść oraz stanów wewnętrznych

Wejścia układu kodowano kodem z inwersyjnym powtórzeniem lub kodem Bergera (jeżeli liczba bitów informacyjnych wynosiła 2^n , stosowano zmodyfikowany kod Bergera). W tabeli 5.2 zebrano dane odnośnie liczby wejść oraz podano złożoności najmniejszych znanych układów STC dla zastosowanych kodów. Nie syntezerowano układów wykorzystujących kod Bergera, w których liczba wejść wynosiła 1 lub 2.

Analiza kompletności kodów wejściowych dla wszystkich układów wzorcowych wykazała, że kody są kompletne i tym samym nie ma konieczności weryfikowania ST układów STC.

Do kodowania stanów wewnętrznych syntezerowanych układów zastosowano kody $1/n$ oraz $2/n$ (tabela 5.3). W kolumnie 2 podano liczbę stanów w poszczególnych układach wzorcowych. Kody dobrano zgodnie z metodą podaną w podrozdziale 4.3.2, co zapewniło ST układów STC dla zastosowanych kodów stanów wewnętrznych układu sekwencyjnego.

5.4 Konstrukcja kodu wyjściowego oraz STC

W kolejnym kroku konstruowano kod wyjściowy oraz układ STC dla tego kodu, zgodnie z metodą podaną w podrozdziale 4.4. W tabeli 5.4 zebrano uzyskane wyniki: kod wyjściowy oraz złożoność STC dla tego kodu. Dla układów (`lion`, `lion9`, `shiftreg`, `train11`, `train4`) nie ma potrzeby stosowania układu STC, ponieważ wyjścia tych układów są 1 parą linii kodu z inwersyjnym powtórzeniem. Należy

Tablica 5.2: Zestawienie kodów zastosowanych do kodowania wejść

Układ	Liczba wejść	Złożoność STC dla kodu z inw. powt.	Złożoność STC dla kodu Bergera
bbara	4	36	50
bbtas	2	12	—
bbse	7	72	81
beecount	3	24	38
cse	7	72	81
dk14	3	24	38
dk15	3	24	38
dk16	2	12	—
dk17	2	12	—
dk27	1	0	—
dvram	8	84	93
ex1	9	96	110
ex3	2	12	—
ex6	5	48	61
fetch	9	96	110
keyb	7	72	81
kirkman	12	132	—
lion	2	12	—
lion9	2	12	—
log	9	96	110
mark1	5	48	61
mc	3	24	38
opus	5	48	61
planet	7	72	81
planet1	7	72	81
rie	9	96	110
s1	8	84	93
shiftreg	1	0	—
sse	7	72	81
styr	9	96	110
tav	4	36	50
tbk	6	60	71
train11	2	12	—
train4	2	12	—

Tablica 5.3: Zestawienie kodów zastosowanych do kodowania stanów wewnętrznych

Układ	Liczba stanów wewnętrznych	Kod 1/n	Kod 2/n
bbara	7	1/7	2/5
bbtas	6	1/6	2/4
bbse	13	1/13	2/6
beecount	4	1/4	2/4
cse	16	1/16	2/7
dk14	7	1/7	2/5
dk15	4	1/4	2/4
dk16	27	1/27	2/8
dk17	8	1/8	2/5
dk27	7	1/7	2/5
dvram	35	1/35	2/9
ex1	18	1/18	2/7
ex3	5	1/5	2/4
ex6	8	1/8	2/5
fetch	24	1/24	2/8
keyb	19	1/19	2/7
kirkman	17	1/17	2/7
lion	4	1/4	2/4
lion9	4	1/4	2/4
mark1	12	1/12	2/6
mc	4	1/4	2/4
planet	48	1/48	2/11
planet1	48	1/48	2/11
rie	29	1/29	2/8
s1	20	1/20	2/7
shiftreg	8	1/8	2/5
sse	13	1/13	2/6
styr	30	1/30	2/9
tav	4	1/4	2/4
tbk	16	1/16	2/7
train11	4	1/4	2/4
train4	4	1/4	2/4

tutaj zaznaczyć, że synteza STC jest możliwa dla prawie wszystkich układów. Jedynie dla układów `ex4`, `ex5`, `log` i `opus` niemożliwe było zapewnienie ST układu STC.

Tablica 5.4: Zestawienie kodów wyjściowych dla układów wzorcowych

Układ	Stosowane kody	Złożoność STC
<code>bbara</code>	1/4	16
<code>bbtas</code>	2p2r	12
<code>bbse</code>	1/6, 2p2r	48
<code>beecount</code>	1/2, 1/2	12
<code>cse</code>	1/4, 1/4, 1/4	72
<code>dk14</code>	1/4, 1/4, 1/2	58
<code>dk15</code>	1/4, 1/4, 1/2	58
<code>dk16</code>	1/4, 1/2	30
<code>dk17</code>	1/4, 1/2	30
<code>dk27</code>	1/4	16
<code>dvrām</code>	1/11, 1/4, 1/4	110
<code>ex1</code>	1/9, 1/4, 1/4, 6p2r	175
<code>ex3</code>	2p2r	12
<code>ex6</code>	1/4, 5p2r	76
<code>fetch</code>	1/16	82
<code>keyb</code>	1/4	16
<code>kirkman</code>	1/4, 4p2r	64
<code>lion</code>	1/2	0
<code>lion9</code>	1/2	0
<code>mark1</code>	1/9, 1/5, 1/3, 2p2r	133
<code>mc</code>	1/4, 1/4, 1/2	58
<code>planet</code>	1/7, 1/4, 1/4, 1/6, 6p2r	195
<code>planet1</code>	1/7, 1/4, 1/4, 1/4, 6p2r	195
<code>rie</code>	1/14, 1/7, 1/4, 3p2r	209
<code>s1</code>	1/5, 2p2r	46
<code>shiftreg</code>	1/2	0
<code>sse</code>	1/6, 2p2r	48
<code>styr</code>	1/6, 1/4, 3p2r	88
<code>tav</code>	1/4, 2p2r	40
<code>tbk</code>	1/4, 1/2	30
<code>train11</code>	1/2	0
<code>train4</code>	1/2	0

5.5 Układy o strukturze TSC/CD

Dla przyjętego kodowania syntezowano układy sekwencyjne zrealizowane w wersji TSC/CD. W tym celu dla układów, dla których możliwa była synteza kodu wyjściowego oraz zapewniono ST wyjściowego STC, weryfikowano CD funkcji przejść/wyjść SM i, jeżeli było to konieczne, modyfikowano układ w celu zapewnienia CD.

W tabeli 5.5 przedstawiono wyniki weryfikacji własności CD układów wzorcowych. Jak można zauważyć, dla części układów udało się uzyskać CD. Dla pozostałych układów wykazano, że są one bezpieczne dla błędów wejściowych (ES). Uzyskane wyniki pokazują, że CD zależy od przyjętego kodowania. Najwięcej układów CD udało się uzyskać dla wejściowego kodu z inwersyjnym powtórzeniem

oraz stanów zakodowanych kodem $1/n$ (17 układów). Dla układów **beecount**, **dk15**, **mc** i **tav** realizacja CD nie zależy od przyjętego kodowania, natomiast bez względu na przyjęte kodowanie nie można podać realizacji CD dla układów **bbsee**, **cse**, **dvram**, **ex6**, **fetch**, **keyb**, **kirkman**, **lion**, **mark1**, **planet**, **planet1**, **sse**, **tbk**, **train11** i **train4**. Analiza części przypadków, dla których niemożliwa jest modyfikacja zapewniająca CD pokazuje, że istnieje możliwość syntezy układu CD, jednak wymaga to modyfikacji grafu przejść poprzez dodanie stanów równoważnych.

Następnie dla wszystkich układów, dla których wykazano CD weryfikowano własność ST układów o strukturze AND-OR realizujących funkcję przejść/wyjść. W tabeli 5.6 podano liczbę nietestowalnych uszkodzeń w otrzymanych układach. Jak można zauważyć, dla układów dwupoziomowych AND-OR uzyskano ST dla 100% uszkodzeń dla większości układów (**bbtas**, **dk14**, **dk15**, **dk16**, **dk17**, **dk27**, **lion9**) bez względu na zastosowane kody. Wpływ przyjętego kodowania na liczbę nietestowalnych uszkodzeń jest niewielki. Jak widać, liczba uszkodzeń nietestowalnych w układach AND-OR jest mniejsza dla układów, których wejścia kodowano kodem z inwersyjnym powtórzeniem. Dla wszystkich układów, które zawierają uszkodzenia nietestowane podjęto próbę syntezy z wykorzystaniem algorytmu 3.4 tak, aby dodać dodatkową warstwę bramek AND i uzyskać układy ST w 100%. Dodanie dodatkowej warstwy bramek AND umożliwiło eliminację wszystkich uszkodzeń nietestowalnych. W tabeli 5.7 podana jest liczba literalów w układach TSC/CD po minimalizacji układu realizującego funkcję przejść/wyjść programem **SIS**, powiększona o złożoność zastosowanego układu STC dla kodu wyjściowego. W tabeli przedstawiono tylko te układy, dla których udało się podać realizację w pełni TSC. Układy o najmniejszej złożoności uzyskano dla wejść zakodowanych kodem z inwersyjnym powtórzeniem oraz stanów wewnętrznych kodem $1/n$. Różnice w rozmiarach układów, w zależności od wybranego kodowania nie są duże. Dla porównania, w tabeli podano także złożoność układu zdublowanego. Niestety, w większości przypadków układy TSC/CD są większe od układów zdublowanych.

Zaproponowano także nową, stanowiącą alternatywę dla TSC/CD, strukturę TSC SM: TSC/ES. W układy o tej strukturze rezygnuje się z CD na rzecz ES. W układach tych wszystkie błędy wejściowe są wykrywane lub korygowane (w układach CD wszystkie błędy wejściowe są wykrywane). Rezygnacja z CD prowadzi do układów o mniejszej złożoności wynikającej z rezygnacji z nadmiaru sprzętowego związanego z zapewnieniem CD. Metoda syntezy układów TSC/ES jest taka sama, jak dla układów TSC/CD z tym, że pominięto weryfikację CD i weryfikowano tylko ST. W tabeli 5.8 zebrano wyniki analizy ST układów AND-OR i modyfikacji w celu poprawy ST do struktury AND-AND-OR. Jak widać liczba nietestowalnych uszkodzeń w układach AND-OR jest znacznie większa niż w układach AND-OR zrealizowanych jako TSC/CD. Jedynie dla układów **dk14**, **dk27** i **shiftreg** istnieje TSC realizacja o strukturze AND-OR. Dla wszystkich układów podjęto próbę eliminacji uszkodzeń nietestowalnych. Dodanie dodatkowej warstwy bramek AND spowodowało znaczny spadek liczby nietestowalnych uszkodzeń. Dla części układów można podać realizację TSC w 100%. Dla pozostałych układów zidentyfikowano wszystkie nietestowane uszkodzenia, które w razie potrzeby mogą być wykrywane deterministycznymi testami off-line.

Dane nt. złożoności układów TSC/ES zebrano w tabeli 5.8, gdzie podano liczbę literalów w układach, powiększoną o złożoność układu STC dla kodu wyjściowego. Najmniejsze układy uzyskano dla wejść zakodowanych kodem Bergera oraz stanów wewnętrznych kodem $1/n$. Złożoności układów w większości przypadku są mniejsze niż realizacji układów zdublowanych (**dk14**, **dk15**, **dk17**, **lion9**,

Tablica 5.5: Wyniki analizy kodowej rozłączności dla TSC/CD SM

Układ	Kodowanie			
	Wejście: kod z inw. powt.		Wejście: kod Bergera	
	St. wew.: 1/n	St. wew.: 2/n	St. wew.: 1/n	St. wew.: 2/n
bbara	CD		CD	
bbtas	CD	CD		
bbsse				
beecount	CD	CD	CD	CD
cse				
dk14	CD		CD	
dk15	CD	CD	CD	CD
dk16	CD	CD		
dk17	CD			
dk27	CD		CD	
dvram				
ex1				CD
ex3	CD	CD		
ex6				
fetch				
keyb				
kirkman				
lion				
lion9	CD	CD		
mark1	CD			
mc	CD	CD	CD	CD
planet				
planet1				
shiftreg	CD		CD	
sse				
styr	CD			
tav	CD	CD	CD	CD
tbk				
train11				
train4				

Uwaga:

pusta rubryka – układ nie jest CD.

lion). Niestety niemożliwe było obliczenie złożoności szeregu układów z powodu błędu w procedurze minimalizacji programu SIS.

Porównanie złożoności układów TSC/CD oraz TSC/ES pokazuje, że koszt zapewnienia CD jest znaczny (np. dla układów **bbara** – 38%, **dk14** – 30%) . W aplikacjach, w których można zrezygnować z CD na rzecz ES, synteza układów TSC/ES jest rozwiązaniem, które daje układy o mniejszej złożoności, oferujące podobny stopień zabezpieczenia przed błędami na wejściu i taki sam stopień zabezpieczenia przed uszkodzeniami wewnętrznymi.

5.6 Układy o strukturze TSC/STC

Wyniki syntezy układów wzorcowych o strukturze TSC/STC zamieszczono w tabeli 5.10 zawierającej dane na temat złożoności uzyskanych realizacji, powiększonej o złożoności zastosowanych STC. Układy

Tablica 5.6: Liczba nietestowalnych uszkodzeń w układach TSC/CD SM

Układ	Kodowanie			
	Wejście: kod z inw. powt.		Wejście: kod Bergera	
	St. wew.: 1/n	St. wew.: 2/n	St. wew.: 1/n	St. wew.: 2/n
bbara	0		0(14)	
bbtas	0	0		
bbse				
beecount	0	0	0(23)	0(28)
cse				
dk14	0		0	
dk15	0	0	0	0
dk16	0	0		
dk17	0			
dk27	0		0	
dvram				
ex1				
ex3	0	0		
ex6				
fetch				
keyb				
kirkman				
lion				
lion9	0	0		
mark1	0			
mc	0	0	0(9)	0(9)
planet				
planet1				
rie				
s1				
shiftreg	0		0	
sse				
styr				
tav	0	0	0(36)	0(36)
tbk				
train11				
train4				

Uwagi:

pusta rubryka – układ nie jest CD, więc ST nie jest analizowana;

0 oznacza 100% wykrywalność uszkodzeń;

w nawiasach podano liczbę nietestowalnych uszkodzeń w układzie AND-OR dla układów, w których konieczne było dodanie dodatkowej warstwy bramek AND.

Tablica 5.7: Złożoności układów TSC/CD

Układ	Kodowanie				Układ zdublowany
	Wejście: kod z inw. powt.		Wejście: kod Bergera		
	St. wew.: 1/n	St. wew.: 2/n	St. wew.: 1/n	St. wew.: 2/n	
bbara	233		348		188
bbsse					419
bbtas	118	121			100
beecount	163	172	145	140	136
cse					517
dk14	344		326		295
dk15	266	269	263	251	257
dk16	514	468			715
dk17	220				197
dk27	103		103		84
dvram					843
ex1					1367
ex3	142	137			96
ex6					398
fetch					585
keyb					886
kirkman					906
lion					45
lion9	92	99			49
mark1	436				434
mc	191	204	208	231	129
planet					1851
planet1					1851
rie					1032
s1					1440
shiftreg	74		74		45
sse					419
styr					2058
tav	221	239	213	246	132
tbk					831
train11					39
train4					59

Uwaga:

złożoność została obliczona z wykorzystaniem oprogramowania SIS i jest podana w literalach.

Tablica 5.8: Liczba nietestowalnych uszkodzeń w układach TSC/ES

Układ	Kodowanie			
	Wejście: kod z inw. powt.		Wejście: kod Bergera	
	St. wew.: 1/n	St. wew.: 2/n	St. wew.: 1/n	St. wew.: 2/n
bbara	26 (264)	0 (266)	14 (163)	0 (165)
bbsse	2368 (8480)	1728 (9568)	0 (1444)	0 (2532)
bbtas	5 (20)	0 (23)	15 (30)	3 (33)
beecount	48 (56)	64 (86)	44 (44)	60 (74)
cse	116 (484)	72 (499)	102 (185)	105 (200)
dk14	0	0 (22)	0	0 (22)
dk15	0	0 (16)	0	0 (16)
dk16	0	0	0	0
dk17	0 (2)	0 (2)	0 (2)	0 (2)
dk27	0	0	0	0
dvram	47168 (68608)	23104 (68608)	35392 (45628)	3136 (45628)
ex1	7104 (48768)	1536 (50048)	1491 (9981)	0 (11261)
ex3	0	0 (3)	0	0 (3)
ex6	68 (808)	44 (840)	0 (269)	0 (301)
fetch	69120 (105984)	36352 (105984)	47730 (60645)	0 (60645)
keyb	5746 (14732)	1761 (15864)	2690 (9122)	686 (10254)
kirkman				307584 (332672)
lion9	2 (12)	1 (22)	0 (4)	0 (14)
lion	2 (12)	5 (26)	0 (6)	3 (20)
mark1	961 (1672)	1153 (1864)	0 (724)	0 (916)
mc	0 (56)	0 (56)	0 (23)	0 (23)
planet1	17216 (37552)	1536 (37488)	12192 (18887)	0 (18725)
planet	17216 (37552)	1536 (37488)	12192 (18887)	0 (18725)
rie	41472 (123392)	18432 (123392)	28638 (57338)	0 (57338)
s1	5120 (32088)	0 (32088)	3136 (14564)	0 (14564)
shiftreg				
sse	2368 (8480)	1728 (9568)	0 (1444)	0 (2532)
styr		9872 (116128)	28638 (40684)	0 (40684)
tav	0 (72)	0 (72)	0 (63)	0 (63)
tbk	7 (3718)	7 (4502)	0 (966)	0 (1750)
train11	0 (17)	7 (24)	9 (15)	6 (22)
train4	0 (6)	2 (18)	8 (14)	6 (26)

Uwagi:

0 oznacza 100% wykrywalność uszkodzeń;

w nawiasach podano liczbę nietestowalnych uszkodzeń w układzie AND-OR dla układów, w których konieczne było dodanie dodatkowej warstwy bramek AND.

Tablica 5.9: Złożoności układów TSC/ES

Układ	Kodowanie				Układ zdublowany
	Wejście: kod z inw. powt.		Wejście: kod Bergera		
	St. wew.: 1/ <i>n</i>	St. wew.: 2/ <i>n</i>	St. wew.: 1/ <i>n</i>	St. wew.: 2/ <i>n</i>	
bbara	153	210	153	199	188
bbsse	341	414			419
bbtas	87	104	85	101	100
beecount	99	119	96	111	136
cse	523	693	477	568	517
dk14	292	338	260	306	295
dk15	227	263	213	232	257
dk16	433	468	427	465	715
dk17	170	210	178	215	197
dk27	80	99	80	99	84
dvrām					843
ex1					1367
ex3	103	118	103	114	96
ex6	294	341	331	422	398
fetch	339				585
keyb	417				886
kirkman					906
lion	50	56	53	60	45
lion9		73	56	74	49
mark1	299	315	332	355	434
mc	157	165	163	178	129
planet					1851
planet1					1851
rie					1032
s1					1440
shiftreg	74	75	74	75	45
sse					419
styr					2058
tav	184	211	208	215	132
tbk					831
train11	55	68	53	73	39
train4	45	55	45	55	59

Uwaga:

Złożoność została obliczona z wykorzystaniem oprogramowania SIS i jest podana w literalach.

minimalizowano zgodnie z algorytmem 3.5 przedstawionym w pracy i wykazano, że są ST na mocy tw. 3.7. Układy o najmniejszej złożoności uzyskano dla wejść zakodowanych kodem z inwersyjnym powtórzeniem oraz stanów wewnętrznych zakodowanych kodem $1/n$. Z powodu nadmiernej złożoności obliczeniowej, wynikającej z konieczności rozwinięcia wejściowych stanów obojętnych, nie była możliwa synteza szeregu układów TSC/STC, których wejścia kodowano kodem Bergera. Porównanie złożoności otrzymanych układów ze złożonością układów w wersji zdublowanej pokazuje, że stosowanie układów o strukturze TSC/STC jest szczególnie opłacalne dla układów dużych, dla których udało się uzyskać wersje mniejsze nawet o 40% od wersji zdublowanych (np. **planet1**). Należy tutaj zaznaczyć, że dla wszystkich układów zastosowanie struktury TSC/STC SM prowadzi do realizacji 100% TSC.

Tablica 5.10: Złożoności układów TSC/STC

Układ	Kodowanie				Układ zdublowany
	Wejście: kod z inw. powt.		Wejście: kod Bergera		
	St. wew.: 1/n	St. wew.: 2/n	St. wew.: 1/n	St. wew.: 2/n	
bbara	212	264	243	273	188
bbsse	409	433	666	743	419
bbtas	104	94			100
beecount	113	117	134	142	136
cse	676	706	609	662	517
dk14	340	344	364	359	295
dk15	250	263	254	274	257
dk16	506	488			715
dk17	209	223			197
dk27	113	112			84
dvram	604	523	748	697	843
ex1	824				1367
ex3	128	132			96
ex6	376	459	498	664	398
fetch	422	403		653	585
keyb	455	548	578	636	886
kirkman					906
lion	72	77			45
lion9	70	71			49
mark1	358	346	412	403	434
mc	172	174	205	219	129
planet	1247	1245			1851
planet1	1247				1851
rie	740	684			1032
s1	637				1440
shiftreg	109	91			45
sse	409	433			419
styr	1243				2058
tav	183	185	208	211	132
tbk	697	800		796	831
train11	70	74			39
train4	73	68			59

Uwaga:

Złożoność została obliczona z wykorzystaniem oprogramowania SIS i jest podana w literalach.

5.7 Podsumowanie

Przeprowadzona analiza zaproponowanych metod syntezy TSC/SM pokazuje, że istnieje możliwość syntezy układów TSC w 100% zarówno w wersji TSC/CD jak i TSC/STC. Dla obu struktur kluczowym zagadnieniem jest zapewnienie ST układu STC dla kodu wyjściowego. Zaproponowane w pracy metody umożliwiają osiągnięcie tego celu dla większości analizowanych układów. Ponadto, dla układów TSC/CD konieczne jest zapewnienie CD oraz ST układu realizującego funkcję przejść/wyjść. Cel ten jest osiągalny, ale kosztem rozbudowy układu funkcjonalnego. Natomiast dla układów TSC/STC zaproponowana metoda minimalizacji prowadzi do realizacji ST układu funkcjonalnego kosztem rozbudowy układu o dwa dodatkowe STC. Wyniki syntezy układów wzorcowych pokazują, że cel 100% TSC łatwiej osiągnąć stosując strukturę TSC/STC, ponieważ łatwo zapewnić ST układu STC dla kodu wejściowego oraz kodu stanów wewnętrznych, natomiast układ realizujący funkcję przejść/wyjść po zminimalizowaniu algorytmem 3.5 jest automatycznie ST. Problemy syntezy STC dla kodu wyjściowego są takie same jak dla układów TSC/CD.

Alternatywą dla struktury TSC/CD może być nowa zaproponowana po raz pierwszy w tej pracy struktura układów TSC SM: TSC/ES. Układy SM zrealizowane jako TSC/ES cechuje mniejsza złożoność przy zachowaniu podobnego stopnia zabezpieczenia przed błędami wejściowymi (błędy są wykrywane lub korygowane) oraz takiego samego stopnia zabezpieczenia przed uszkodzeniami wewnętrznymi, ale traci się wcześniejsze sygnały o wystąpieniu uszkodzenia w układzie generującym wejścia.

Porównanie złożoności uzyskanych układów TSC/STC oraz TSC/CD pokazuje, że koszt związany z rozbudową układów TSC/CD, wynikającą z zapewnienia własności CD, jest duży i nie jest kompensowany brakiem konieczności zastosowania dwóch dodatkowych układów STC. Porównanie złożoności uzyskanych układów z wynikami z prac [33], [40] jest trudne, ponieważ autorzy nie podają, w jaki sposób obliczali liczbę literali. Porównując wyniki z rezultatami znanymi w literaturze należy podkreślić, że uzyskane układy są w 100% TSC oraz wszystkie wyniki uzyskano w pełni automatycznie. Celu tego nie udało się osiągnąć stosując którąkolwiek ze znanych z literatury metod. Ponadto w niniejszej pracy po raz pierwszy rozpatrywane zostało zagadnienie błędów i uszkodzeń na wejściu układu.

Podsumowując, analiza zaproponowanych w pracy metod syntezy TSC SM potwierdza sformułowaną we wstępie tezę, że istnieje możliwość praktycznej realizacji 100% TSC SM zarówno jako układów TSC/CD jak i TSC/STC. Należy także podkreślić, że wszystkie układy syntezowano w pełni automatycznie, z wykorzystaniem opracowanego pakietu CAD.

Rozdział 6

Zakończenie

W PRACY zostały zaproponowane i opracowane nowe metody syntezy całkowicie samosprawdzalnych (TSC) układów sekwencyjnych, które bazują na kodowaniu wejść, stanów wewnętrznych oraz wyjść układów sekwencyjnych kodami nieuporządkowanymi i na syntezie układów jako bezinwerterowych. Zbiór rozpatrywanych uszkodzeń ograniczono do pojedynczych uszkodzeń typu s/z oraz przyjęto, że uszkodzenia nie występują na liniach zegarowych. Rozpatrywano także błędy na wejściach pierwotnych układu, których klasę ograniczono do błędów pojedynczych oraz wielokrotnych błędów jednokierunkowych. Założono, że uszkodzenia i błędy nie występują jednocześnie, a klasę układów sekwencyjnych ograniczono to układów Mealy'ego.

W pracy zostały przeanalizowane dwie struktury TSC układów sekwencyjnych: TSC/CD oraz TSC/STC. Aby osiągnąć cel TSC, konieczne jest zapewnienie samotestowalności (ST), odporności na uszkodzenia (FS) oraz kodowej-rozłączności (CD). W układach TSC/CD zakłada się, że układ funkcjonalny, realizujący funkcję przejść-wyjść, jest CD, FS oraz ST. Dla układów TSC/STC o układzie funkcjonalnym zakłada się, że jest ST oraz FS natomiast nie jest CD, stąd też konieczne jest stosowanie dodatkowych STC monitorujących wejścia, stany wewnętrzne oraz wyjścia układu. Jak wykazano w [90], układy o wejściach oraz wyjściach zakodowanych kodami nieuporządkowanymi są FS z definicji, dlatego weryfikacji wymagają jedynie własności ST i CD.

W pracy zaproponowano także modyfikację struktury TSC/CD — TSC/ES, gdzie zrezygnowano z CD na rzecz układów bezpiecznych dla błędów wejściowych (ES), która oferuje podobny stopień zabezpieczenia przed błędami wejściowymi oraz taki sam stopień zabezpieczenia przed uszkodzeniami wewnętrznymi.

Otrzymane wyniki

Przedstawiony przegląd i krytyczna analiza literatury z zakresu syntezy TSC układów sekwencyjnych wykazały poważne ograniczenia znanych metod w zagadnieniach automatycznej syntezy TSC układów sekwencyjnych. Wykazano także, że metody te oferują nieznany stopień pokrycia uszkodzeń i celu 100% TSC nie udaje się osiągnąć.

W związku z brakiem zgodności autorów dotyczącej definicji TSC układu sekwencyjnego, w pracy przyjęto definicję konstruktywną, która pozwala uzyskać weryfikowalne wyniki. Definicja ta implikuje konieczność użycia kodów detekcyjnych we wszystkich interfejsach układu sekwencyjnego i pozwala

na formalną weryfikację podstawowych własności ST, FS oraz CD.

W pracy rozważono dwie odmienne struktury układów TSC: TSC/CD oraz TSC/STC. Należy tutaj zaznaczyć, że struktura TSC/CD nie była nigdy wcześniej rozważana. W pracy podano metodę syntezy TSC/CD układów sekwencyjnych z użyciem kodów nieuporządkowanych, która w przeciwieństwie do wszystkich znanych metod, uwzględnia możliwość wystąpienia błędów i uszkodzeń na wejściu układu. Podano twierdzenia umożliwiające weryfikację własności ST i CD układów o strukturze AND-OR. Dla układów, dla których wykazano brak ST i/lub CD, zaproponowano metody modyfikacji układu, które umożliwiają uzyskanie własności ST i CD. W pracy podano algorytm modyfikacji układu poprzez dodanie par wyjściowych linii kodu z inwersyjnym powtórzeniem, umożliwiającą uzyskanie układów CD. W celu zapewnienia ST zaproponowano metodę dodawania dodatkowej warstwy bramek AND w celu poprawy ST. W odróżnieniu od metod znanych z literatury metoda ta nie modyfikuje, funkcji układu, lecz jedynie jego strukturę. Dzięki temu nie ulega zmianie funkcjonalność układu dla niekodowych pobudzeń, a tym samym nie dochodzi do utraty CD. W pracy podano także metodę syntezy układów TSC/STC stanowiącą rozwinięcie metody z pracy [22]. Zaproponowano algorytm minimalizacji funkcji układu i wykazano także, że zminimalizowany układ jest z definicji ST. Ponadto metoda z [22] ograniczała się wyłącznie do kodów m/n , natomiast metoda podana tutaj stosuje się do układów o wejściach oraz wyjściach zakodowanych dowolnymi kodami nieuporządkowanymi. Niestety, zaproponowana metoda modyfikuje funkcję układu dla niekodowych pobudzeń, co może prowadzić do utraty CD. Z tego też powodu konieczne jest zastosowanie układów STC dla monitorowania wejść, stanów wewnętrznych i wyjść.

W odróżnieniu od innych znanych z literatury metod, w pracy położono nacisk na ST użytych układów STC – szczególnie w przypadku kodów niekompletnych. Synteza układu sekwencyjnego zarówno TSC/CD jak i TSC/STC wymaga zapewnienia istnienia ST układu STC dla kodu wyjściowego. Autorzy większości prac stwierdzają konieczność stosowania STC, jednak faktu jego istnienia nie weryfikują. W efekcie stosowane przez nich układy nie są ST dla kodów niekompletnych. W pracy podano algorytm konstrukcji kodu wyjściowego oraz syntezy ST układu STC dla tego kodu. Algorytm konstrukcji kodu polega na podziale kodu wyjściowego tak, aby stanowił on złożenie kodów $1/n$ oraz innych kodów nieuporządkowanych. Przedstawiony algorytm wykorzystuje stany obojętne i gwarantuje ST układów STC dla wykorzystywanego zbioru słów kodowych (m. in. dzięki umiejętnemu wykorzystaniu stanów obojętnych). Podejście takie nie było do tej pory nigdy stosowane w żadnej ze znanych w literaturze metod syntezy TSC SM. W pracy przedstawiono również metodę doboru kodowania stanów wewnętrznych, która gwarantuje ST układów STC dla kodu stanu wewnętrznych.

Opracowane algorytmy zostały zaimplementowane w formie systemu automatycznego projektowania układów sekwencyjnych TSCSM/CAD. W rozdziale 5 wykazano, że dla szeregu wzorcowych układów sekwencyjnych (ang. *benchmarks*), istnieje realizacja gwarantująca, że układ jest TSC dla 100% wszystkich uszkodzeń pojedynczych typu sklejenie. Warto tu odnotować fakt, że autorzy wszystkich znanych prac skrupulatnie przemilczają, że pokrycie uszkodzeń w ich układach jest mniejsze niż 100%. Ponadto, należy zaznaczyć, że w pracy nie ograniczono się tylko do kodów m/n oraz z inwersyjnym powtórzeniem (dotychczas tylko te kody nieuporządkowane były rozpatrywane), lecz rozważania dotyczą dowolnych kodów nieuporządkowanych. Synteza układów wzorcowych pokazała, że możliwe jest osiągnięcie celu TSC zarówno dla układów o strukturze TSC/CD i/lub TSC/STC dla większo-

ści układów wzorcowych. Tylko dla nielicznych układów opracowane metody zawiodły z powodu nadmiernej złożoności obliczeniowej.

Reasumując, za najistotniejsze rezultaty niniejszej rozprawy, traktowane jako dowód postawionej w rozdziale 1 tezy należy uznać:

- sformułowanie warunków weryfikacji CD układów AND-OR,
- podanie metody modyfikacji układów AND-OR w celu uzyskania CD,
- sformułowanie warunków weryfikacji ST układów AND-OR,
- podanie metody modyfikacji układów AND-OR w celu uzyskania ST,
- podanie metody minimalizacji układów AND-OR i wykazanie, że tak zminimalizowane układy są z definicji ST,
- podanie kryterium wyboru kodu stanów wewnętrznych i algorytmu konstrukcji kodu wyjściowego zapewniających samotestowalność STC.

Wyniki uzyskane w pracy pozwalają zatem na stwierdzenie, że istnieje możliwość automatycznej syntezy całkowicie samosprawdzalnych (TSC) układów sekwencyjnych oferujących 100% pokrycie uszkodzeń i błędów.

Otwarte problemy badawcze

Przeprowadzone badania wskazują na następujące otwarte zagadnienia, dotyczące metod syntezy TSC układów sekwencyjnych:

- redukcja złożoności wynikająca z konieczności zapewnienia własności CD,
- metody modyfikacji grafu przejść w celu zapewnienia własności ST i CD,
- opracowanie algorytmów dekompozycji złożonych układów sekwencyjnych na kilka współpracujących TSC układów sekwencyjnych,
- opracowanie metody syntezy współpracujących TSC układów sekwencyjnych.

Przedstawione w rozprawie algorytmy pokazują, że istnieje możliwość automatycznej syntezy układów TSC dla szeregu przypadków. Jednakże w dalszym ciągu otwarty pozostaje problem syntezy układów, których synteza w chwili obecnej nie jest możliwa z powodu nadmiernej złożoności obliczeniowej. Opracowanie metod heurystycznych umiejętnie wykorzystujących wejściowe stany obojętne w przypadku układów o bardzo dużej liczbie wejść i stanów wewnętrznych może doprowadzić do rozwiązania problemu syntezy układów o dużym stopniu złożoności. Alternatywną metodą redukcji złożoności jest dekompozycja układu i syntezy układu jako układów współpracujących. Ponadto pewnym mankamentem opracowanych metod modyfikacji struktury układu zapewniających CD jest duża złożoność otrzymanych układów.

Analiza szeregu nierozwiązanych przypadków, w tym układów dla których synteza z definicji nie jest możliwa, wykazuje, że modyfikacja grafu przejść może prowadzić do poprawy własności ST i/lub CD. Mogą mieć tutaj zastosowanie metody analizy relacji w grafie przejść. W połączeniu z metodami podanymi w pracy prowadzić mogą one do układów o mniejszej złożoności.

Bibliografia

- [1] D. A. ANDERSON, *Design of self-checking digital networks using coding techniques*, rozprawa doktorska, Univ. of Illinois, Urbana, Oct. 1971.
- [2] D. A. ANDERSON, G. METZE, Design of totally self-checking check circuits for m -out-of- n codes, *IEEE Transactions on Computers*, vol. C-22, s. 263–269, Mar. 1973.
- [3] M. J. ASHJAEI, S. M. REDDY, Totally self-checking checkers for a class of separable codes, *Proceedings 12th Annual Allerton Conference Circuit and System Theory*, s. 238–242, Monticello, IL, Oct. 1974.
- [4] M. J. ASHJAEI, S. M. REDDY, On totally self-checking checkers for separable codes, *IEEE Transactions on Computers*, vol. C-26, s. 737–744, Aug. 1977.
- [5] C. BOLCHINI ET AL., Design of VHDL-based totally self checking finite state machine and data-path description, *IEEE Transactions on VLSI Systems*, vol. 8, s. 98–108, Feb. 2000.
- [6] C. BOLCHINI, R. MONTANDON, F. SALICE, D. SCIUTO, Self-checking FSMs based on constant distance state encoding, *Proceedings IEEE International Symposium Defect and Fault Tolerance in VLSI*, s. 269–277, Lafayette, LA, 13–15 Nov. 1995.
- [7] F. BRGLEZ, D. BRYAN, K. KOZMINSKI, Notes on ISCAS'89 benchmark circuits, *MCNC*, Oct. 1989.
- [8] D. BRYAN, The ISCAS'85 benchmark circuits and netlist format, *MCNC*, 1985.
- [9] F. Y. BUSABA, P. K. LALA, Self-checking sequential circuits design using m -out-of- n codes, *Electronics Letters*, vol. 29(1), s. 7–9, Jan. 1993.
- [10] F. Y. BUSABA, P. K. LALA, Self-checking combinational circuits design for single and unidirectional multibit errors, *Journal of Electronics Testing: Theory and Applications*, vol. 5, s. 19–28, 1994.
- [11] W. C. CARTER, P. R. SCHNEIDER, Design of dynamically checked computers, *Proceedings IFIP Conference*, s. 878–883, Edinburgh, Scotland, Aug. 1968.
- [12] W. F. CHANG, C. W. WU, Low-cost modular totally self checking checker design for m -out-of- n code, *IEEE Transactions on Computers*, vol. 48(8), s. 815–826, Aug. 1999.

- [13] H. Y. H. CHUANG, S. DAS, Design of fail-safe machines using separable codes, *IEEE Transactions on Computers*, vol. C-27, s. 249–252, Mar. 1978.
- [14] R. W. COOK, W. H. SISSON, T. F. STOREY, W. N. TOY, Design of a self-checking microprogram control, *IEEE Transactions on Computers*, vol. C-22, s. 255–262, Mar. 1973.
- [15] Y. CROUZET, C. LANDRAULT, Design of self-checking MOS-LSI circuits: Application to a four-bit microprocessor, *IEEE Transactions on Computers*, vol. C-29, s. 532–537, Jun. 1980.
- [16] D. H. SAWIN, Design of reliable synchronous sequential circuits, *IEEE Transactions on Computers*, vol. C-24, s. 567–570, May 1974.
- [17] R. DANDAPANI, Derivation of minimal test sets for monotonic circuits, *IEEE Transactions on Computers*, vol. C-22(6), s. 657–661, Jul. 1973.
- [18] V. V. DANILOV, N. V. KOLESOV, B. P. PODKOPAIEV, An algebraic model for the hardware monitoring of automata, *Autom. Remote Control*, vol. 36(6), s. 984–981, 1975.
- [19] R. DAVID, P. THEVENOD-FOSSE, Design of totally self-checking asynchronous modular circuits, *J. Des. Autom. Fault-Tolerant Comput.*, vol. 2, s. 271–278, Oct. 1978.
- [20] K. DE ET AL., SYN: A system for automated synthesis of reliable multilevel circuits, *IEEE Transactions on VLSI Systems*, vol. 2, s. 186–195, Jun. 1994.
- [21] M. DIAZ, Design of totally self-checking and fail-safe sequential machines, *Dig. Pap. 4th Int. Symp. on Fault-Tolerant Computing*, s. 3–19 – 3–24, Urbana, IL, 1974.
- [22] M. DIAZ, P. AZEMA, J. M. AYACHEE, Unified design of self-checking and fail-safe combinational circuits and sequential machines, *IEEE Transactions on Computers*, vol. C-28, s. 276–281, Mar. 1979.
- [23] M. DIAZ, J. M. DE SOUZA, Design of self-checking microprogrammed controls, *Dig. Pap. 5th Int. Symp. on Fault-Tolerant Computing*, s. 137–142, Paris, France, Jun. 1975.
- [24] M. DIAZ, J. C. GEFFROY, M. COURVOISIER, On-set realization of fail-safe sequential machines, *IEEE Transactions on Computers*, vol. C-23, s. 133–138, Feb. 1974.
- [25] C. P. DISPARTE, A design approach for an electronic engine controller self-checking microprocessor, *Proceedings EUROMICRO Conference*, s. 243–247, North-Holland, 1981.
- [26] TH. LINDENKREUZ E. BOHL, R. STEPHAN, The fail-stop controller AE11, *Proceedings International Test Conference*, s. 567–77, Washington, DC, USA, 1997.
- [27] M. GOSSEL, Optimal error detection circuits for sequential circuits with observable states, *Dig. Pap. 5th Int. Symp. on Fault-Tolerant Computing*, s. 175–180, 25–27 Sep. 1991.
- [28] J. W. GREBLICKI, S. J. PIESTRAK, Design of totally self-checking code-disjoint synchronous sequential circuits, *Lecture Notes in Computer Science*, vol. 1667, s. 250–266, 15–17 Sep. 1999.

- [29] C. HALATSIS, N. GAITANIS, Positive fail-safe realization of synchronous sequential machines, *IEEE Transactions on Computers*, vol. C-28, s. 167–172, Feb. 1979.
- [30] M. P. HALBERT, S. M. BOSE, Design approach for a VLSI self-checking MIL-STD-1750A microprocessor, *Dig. Pap. 14th Int. Symp. on Fault-Tolerant Computing*, s. 254–259, Kissimmee, Florida, Jun. 20–22, 1984.
- [31] S. HELLEBRAND, H. J. WUNDERLICH, A. HERTWIG, Synthesis fast, online-testable control units, *IEEE Design and Test of Computers*, s. 36–41, Nov.–Dec. 1998.
- [32] R. HORST ET AL., The risk of data corruption in microprocessor-based systems, *Dig. Pap. 23th Int. Symp. on Fault-Tolerant Computing*, s. 576–585, Toulouse, France, 22–24 Jun. 1993.
- [33] N. K. JHA, S.-J. WANG, Design and synthesis of self-checking VLSI circuits, *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 12, s. 878–887, Jun. 1993.
- [34] S. M. KIA, S. PARAMESWARAN, Design automation of self checking circuits, *Proceedings Euro-DAC'94*, s. 252–257, Grenoble, France, 19–23 Sep. 1994.
- [35] S. M. KIA, S. PARAMESWARAN, Design for self checking flip-flops, *IEE Proceedings Comput. Digit. Tech.*, vol. 145(2), s. 81–88, Mar. 1998.
- [36] S.M. KIA, S. PARAMESWARAN, Self-checking synchronous controller design, *IEE Proceedings - Computers and Digital Techniques*, vol. 146, Jan. 1999.
- [37] Z. KOHAVI, *Switching and Finite Automata Theory*. McGraw-Hill Book Co., N.Y., 1978.
- [38] M. KOTOČOVA, Design of self-checking check circuits for some 1-out-of- n codes, *Proceedings FTSD-4 — 4th International Conference on Fault-Tolerant Systems and Diagnostics*, s. 241–247, Brno, Czechoslovakia, 28–30 Sep. 1981.
- [39] C.-S. LAI, C.-L. WEY, An efficient output function partitioning algorithm reducing hardware overhead in self-checking circuits and systems, *Proceedings 35th Midwest Symposium Circuits System*, s. 1538–1541, 1992.
- [40] C.-S. LAI, C.-L. WEY, SOLIT: an automated system for synthesising reliable sequential circuits with multilevel logic implementation, *IEE Proceedings-Comput. Digit. Tech.*, vol. 142, s. 49–54, Jan. 1995.
- [41] D. LARNER, Sun flips bits in chips, *Electronics Times*, s. 72 and 24, Nov. 1997.
- [42] R. LEVEUGLE, G. SAUCIER, Optimized synthesis of concurrently checked controllers, *IEEE Transactions on Computers*, vol. 39, s. 419–425, Apr. 1990.
- [43] J.-C. LO, A hyper optimal scheme for self-checking circuits, *IEEE Transactions on Computers*, vol. 45, s. 1022–1029, Sep. 1996.
- [44] A. L. LOPUKHA, V. V. SAPOZHNIKOV, VL. V. SAPOZHNIKOV, Synthesis of synchronous automata with self-checking according to output states, *Engineering Cybernetics*, vol. 19(1), 1981.

- [45] G. K. MAKI, A self-checking microprocessor design, *J. Des. Autom. Fault-Tolerant Comput.*, vol. 3, s. 15–27, Jan. 1978.
- [46] G. K. MAKI, III D. H. SAWIN, Real-time fault detection and fault-tolerant implementation for sequential circuits, *Rational Fault Analysis*, s. 32–51, 1977.
- [47] M. A. MAROUF, A. D. FRIEDMAN, Design of self-checking checkers for Berger codes, *Dig. Pap. 8th Int. Symp. on Fault-Tolerant Computing*, s. 179–184, Toulouse, France, Jun. 1978.
- [48] V. I. MAZNEV, Design of self-checking $1/p$ checkers, *Autom. Remote Contr.*, vol. 39(2), s. 1380–1383, Sep. 1978.
- [49] P. MELLOR, CAD: Computer-aided disaster, *High Integrity Systems*, vol. 1(2), s. 101–156, 1994.
- [50] M. N. MEYERS, The AT&T telephone network outage of January 15th 1990, *Special presentation at the 20th International FTC Symposium*, Newcastle upon Tyne, UK, 26–28 Jun. 1990.
- [51] G. DE MICHELI, *Synteza i optymalizacja układów cyfrowych*. Wydawnictwa Naukowo Techniczne, Warszawa, 1998.
- [52] M. MUELLER ET AL., RAS strategy for IBM S/390 G5 and G6, *IBM Journal of Research and Development*, vol. 43, s. 875–888, Sep./Oct. 1999.
- [53] T. NANYA, T. KAWAMURA, Error secure/propagating concept and its application to the design of strongly fault secure processors, *IEEE Transactions on Computers*, vol. C-37, s. 14–24, Jan. 1988.
- [54] T. NANYA, Y. TOHMA, Design of self-checking asynchronous sequential circuits, *Dig. Pap. 10th Int. Symp. on Fault-Tolerant Computing*, s. 278–281, Kyoto, Japan, Oct. 1980.
- [55] T. NANYA, M. UCHIDA, A strongly fault-secure and strongly code-disjoint realization of combinational circuits, *Dig. Pap. 19th Int. Symp. on Fault-Tolerant Computing*, s. 390–397, Chicago, IL, Jul. 1989.
- [56] M. NICOLAIDIS, Evaluation of a self-checking version of the MC68000 microprocessor, *Microproc. and Microprogr.*, vol. 20, s. 235–247, 1987.
- [57] D. NICOLOS, Self-testing embedded two-rail checkers, *J. Electr. Testing: Theory and Appls.*, vol. 12, s. 69–79, 1998.
- [58] S. NIRAJAN, J. F. FRENZEL, Comparison of fault tolerant state machine architectures for space-born electronics, *IEEE Transactions on Reliability*, vol. 45(1), s. 109–112, Mar. 1996.
- [59] F. ÖZGÜNER, Design of totally self-checking asynchronous and synchronous sequential machines, *Dig. Pap. 7th Int. Symp. on Fault-Tolerant Computing*, s. 124–129, Jun. 1977.
- [60] S. PAGEY, S. D. SHERLEKAR, G. VANKATESH, Reliable VLSI sequential controllers, *International Journal of Electronics*, vol. 71(4), s. 609–620, 1991.

- [61] S. PAGEY, S. D. SHERLEKAR, G. VANKATESH, State encoding and functional decomposition for self-checking sequential circuits design, *Proceedings ATS'93 - 2nd Asian Test Symposium*, s. 293–297, Bombay, India, 3–6 Jan. 1993.
- [62] R. A. PAREKHJI, G. VANKATESH, S. D. SHERLEKAR, State assignment for optimal design of monitored self-checking sequential circuits, *Proceedings of 6th International Conference on VLSI Design*, s. 15–20, Bombay, India, 3–6 Jan. 1993.
- [63] R. A. PAREKHJI, G. VANKATESH, S. D. SHERLEKAR, Fault coverage analysis in monitored sequential circuits, *Proceedings ATS'94 - Third Asian Test Symposium*, s. 33–38, Nara, Japan, 15–17 Nov. 1994.
- [64] R. A. PAREKHJI, G. VANKATESH, S. D. SHERLEKAR, Monitoring machine based synthesis technique for concurrent error detection in finite state machines, *Journal of Electronic Testing: Theory and Applications*, vol. 8, s. 179–201, 1996.
- [65] R. A. PAREKHJI, G. VENKATESH, S. D. SHERLEKAR, Concurrent error detection using monitoring machines, *IEEE Design and Test of Computers*, s. 24–32, Fall 1995.
- [66] A.M. PASCHALIS, C. HALATSIS, G. PHILOKYPROU, Concurrently totally self-checking microprogram control unit with duplication of microprogram sequencer, *Microproc. and Microprogr.*, vol. 20, s. 271–281, 1987.
- [67] S. J. PIESTRAK, *Metody syntezy samotestowalnych układów kontrolnych kodów stałowagowych*, rozprawa doktorska, Instytut Cybernetyki Technicznej, Politechnika Wrocławska, 1982.
- [68] S. J. PIESTRAK, Design method of self-testing checkers for 1-out-of- n codes, *Proceedings FTSD'6 — 6th International Conference on Fault-Tolerant Systems and Diagnostics*, s. 57–63, Brno, Czechoslovakia, 5–7 Sep. 1983.
- [69] S. J. PIESTRAK, Design method of totally self-checking checkers for m -out-of- n codes, *Dig. Pap. 13th Int. Symp. on Fault-Tolerant Computing*, s. 162–168, Milan, Italy, 28–30 Jun. 1983.
- [70] S. J. PIESTRAK, PLA implementation of totally self-checking circuits using m -out-of- n codes, *Proceedings ICCD'85, International Conference on Computer Design: VLSI in Computers*, s. 777–781, Port Chester, N.Y., 1–3 Oct. 1–3 1985.
- [71] S. J. PIESTRAK, Design of fast self-testing checkers for Berger codes, *IEEE Transactions on Computers*, vol. C-36, s. 629–634, May 1987.
- [72] S. J. PIESTRAK, The minimal test set for sorting networks and the use of sorting networks in self-testing checkers for unordered codes, *Dig. Pap. 20th Int. Symp. on Fault-Tolerant Computing*, s. 457–464, Newcastle upon Tyne, UK, 26–28 Jun. 1990.
- [73] S. J. PIESTRAK, Design of TSC code-disjoint inverter-free PLAs for separable unordered codes, *Proceedings ICCD'94 International Conference on Computer Design: VLSI in Computers & Procesor*, s. 128–131, Cambridge, MA, 10–12 Oct. 1994.

- [74] S. J. PIESTRAK, *Design of self-testing checkers for unidirectional error detecting codes*. Prace Naukowe Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej Nr 92, Seria: Monografie Nr 24, Oficyna Wyd. Polit. Wrocł., Wrocław, 1995.
- [75] S. J. PIESTRAK, Limitations of design methods of self-checking synchronous sequential machines. *Dig. Pap. 29th Int. Symp. on Fault-Tolerant Computing: Fast Abstracts*, s. 19–20, Wisconsin, USA, 15–18 Jun. 1999.
- [76] S. J. PIESTRAK, Feasibility study of design tsc sequential circuits with 100% fault coverage, *Proceedings 17th. IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, s. 354–362, Vancouver, British Columbia, Canada, 6–8 November 2000.
- [77] S. J. PIESTRAK, Design method of a class of embedded combinational self-testing checkers for 2-rail codes, *IEEE Transactions on Computers*, vol. 51, s. 229–234, Feb. 2002.
- [78] V. RABARA, Design of self-checking checker for 1-out-of- n code ($n > 3$), *Proceedings FTSD'4 — 4th International Conference on Fault-Tolerant Systems and Diagnostics*, s. 234–240, Brno, Czechoslovakia, 28–30 Sep. 1981.
- [79] V. RABARA, A. RABAROVA, Three-level TSC checkers for 1-out-of- n code, *FTSD'8 — Proceedings 8th International Conference on Fault-Tolerant Syst. Diagnostics*, s. 223–229, Katowice, Poland, 1985.
- [80] S. M. REDDY, A note on self-checking checkers, *IEEE Transactions on Computers*, vol. C-22, s. 1100–1102, Oct. 1974.
- [81] S. H. ROBINSON, J. P. SHEN, Direct method for synthesis of self-monitoring state machines, *Dig. Pap. 22th Int. Symp. on Fault-Tolerant Computing*, s. 306–315, Jul. 1992.
- [82] R. ROCHET, R. LEVEUGLE, G. SAUCIER, ASYL-SdF: A synthesis tool for dependability of controllers, *IEICE Transactions Information and Systems*, vol. E79-D(10), s. 1382–1388, Oct. 1996.
- [83] E. M. SANTOVITCH ET AL., *SIS: A System for Sequential Circuit Synthesis*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.
- [84] V. V. SAPOZHNIKOV, VL. V. SAPOZHNIKOV, Synthesis of totally self-checking asynchronous automata, *Automat. Remote Contr.*, vol. 40, s. 124–134, Feb. 1979.
- [85] V. V. SAPOZHNIKOV, VL. V. SAPOZHNIKOV, *Discrete Automata with Error Detection*. Energoatomizdat, Leningrad, SU, 1984 (w j. rosyjskim).
- [86] V. V. SAPOZHNIKOV, VL. V. SAPOZHNIKOV, V. G. TROKHOV, Design of self-checking sequential networks, *Avtom. Vychisl. Tekh.*, vol. 11(3), s. 6–11, 1977.
- [87] N. S. SHCHERBAKOV, B. P. PODKOPAEV, Strukturalnaya teoria apparatnogo kontrolya cifrowych avtomatov, *Mashinostroenie, Moskwa*, 1982. (w j. rosyjskim).

- [88] D. P. SIEWIOREK, R. S. SWARZ, *Reliable Computer Systems: Design and Evaluation*. 2nd Ed., Digital Press, Bedford, MA, 1992.
- [89] T. J. SLEGEL, R. J. VERACCA, Design and performance of the IBM Enterprise System/9000 Type 9121 Vector Facility, *IBM Journal of Research and Development*, vol. 35, s. 367–381, May 1991.
- [90] J. E. SMITH, The design of totally self-checking check circuits for a class of unordered codes, *J. Des. Autom. Fault-Tolerant Comput.*, vol. 2, s. 321–342, Oct. 1977.
- [91] J. E. SMITH, G. METZE, Strongly fault-secure logic networks, *IEEE Transactions on Computers*, vol. C-27, s. 491–499, Jun. 1978.
- [92] YE. S. SOGOMONYAN, YE. V. SLABAKOV, Samoproveraemyje ustrojstva i otkazoustojcivyye sistemy, *Radio i Svyaz, Moskwa*, 1989 (w j. rosyjskim).
- [93] J. SOSNOWSKI, Analiza oraz metody tolerowania błędów przemijających w systemach cyfrowych, *Prace Naukowe Politechniki Warszawskiej, Elektronika, Warszawa*, vol. 103, s. 3–96, 1993.
- [94] L. SPAINHOWER, T. A. GREGG, G4: A fault-tolerant CMOS mainframe, *Digest of Papers 29th International Symposium on Fault-Tolerant Computing*, s. 432–440, 15–18 Jun. 1999.
- [95] T. F. STOREY, Design of a microprogram control for a processor in an electronic switching system, *BSTJ*, vol. 56, s. 183–232, Feb. 1976.
- [96] Y. TOHMA, Design technique of fail-safe sequential machines by using flip-flops for internal memory, *IEEE Transactions on Computers*, vol. C-23, s. 1149–1154, Nov. 1974.
- [97] Y. TOHMA, Y. OHYAMA, R. ZAKAI, Realization of fail-safe sequential machines by using k -out-of- n codes, *IEEE Transactions on Computers*, vol. C-20, s. 1270–1275, Nov. 1971.
- [98] N. A. TOUBA, E. J. MCCLUSKEY, Logic synthesis of multicell circuits with concurrent error detection, *IEEE Transactions on Computers*, vol. 16, s. 783–789, Jul. 1997.
- [99] I. S. VIZIREV, Design of totally self-checking checkers for constant-weight codes, *Autom. Contr. Comput. Sci.*, vol. 16(1), s. 34–40, 1982.
- [100] I. WILLIAMSON, Design of self-checking and fault-tolerant microprogrammed controllers, *Radio and Electronic Engineer*, vol. 47, s. 449–454, Oct. 1977.
- [101] C. Y. WONG, W. K. FUCHS, J. A. ABRAHAM, E. S. DAVIDSON, The design of a microprogram control unit with concurrent error detection, *Dig. Pap. 13th Int. Symp. on Fault-Tolerant Computing*, s. 476–483, Milan, Italy, 28–30 Jun. 1983.
- [102] M. M. YEN, W. K. FUCHS, J. A. ABRAHAM, Designing for concurrent error detection in VLSI: Application to a microprogram control unit, *IEEE Journal of Solid-State Circuits*, vol. C-22, s. 595–605, Aug. 1987.

- [103] C. ZENG, N. SAXENA, E. J. MCCLUSKEY, Finite state machine synthesis with concurrent error detection, *Proceedings of International Test Conference 1999*, s. 672–680, 1999.
- [104] J. F. ZIEGLER ET AL., IBM experiments in soft fails in computer electronics (1978–1994), *IBM Journal of Research and Development*, vol. 40(1), s. 3–15, Jan. 1996.

