POLITECHNIKA KOSZALIŃSKA

WYDZIAŁ ELEKTRONIKI I INFORMATYKI

Rozprawa doktorska

JEDNOSTKI OPERACYJNE ZBUDOWANE W OPARCIU O BRAMKI PRĄDOWE DLA JEDNOUKŁADOWYCH SYSTEMÓW VLSI

mgr inż. Robert Berezowski

Promotor: prof. nadzw. dr hab. inż. Oleg Maslennikow

Koszalin 2007

Pracę dedykuję mojemu promotorowi

i mojej rodzinie

Spis treści

Wykaz skrótów

BIN	– binarny system liczbowy
CLB	– blok logiczny (komórka) układu FPGA (ang. configurable logic block)
CPA	– sumator wielobitowy z propagacją przeniesienia (ang. carry propagate adder)
CSA	– sumator z osobnym układem formującym przeniesienie (ang. carry saved adder)
FA	– pełny sumator jednobitowy (ang. full-adder)
FIFO	– kolejka (ang. first in – first out)
FPGA	– reprogramowalny układ cyfrowy (ang. field programmable gate array)
KBN	– Polski Komitet Badań Naukowych
LUT	– generator funkcji (ang. look-up-table)
MOMA	– wieloargumentowy sumator resztowy (ang. multioperand modular adder)
MVL	– logika wielowartościowa (ang. multiple-valued logic)
NP-trudny	– pojęcie z teorii złożoności obliczeniowej (ang. nondeterministic polynomial)
RAM	– pamięć o dostępie swobodnym (ang. random access memory)
RNS	– resztowy system liczbowy (ang. residue number system)
ROM	– pamięć tylko do odczytu (ang. read-only memory)
SoC	– system jednoukładowy czasu rzeczywistego (ang. system-on-chip)
VHDL	– język opisu sprzętu wysokiego poziomu (ang. very high speed integrated circuits hardware description language)

Wstęp

Na początku lat 90-tych ubiegłego wieku profesor Andrzej Guziński opracował koncepcję bramki cyfrowej pracującej w trybie prądowym oraz jej realizację praktyczną w technologii CMOS. Była to bramka inwertera binarnego. Wraz z rozwojem badań nad nowymi bramkami oraz nad złożonymi z nich układami cyfrowymi (które zostały skrótowo nazwane odpowiednio *bramkami* i *układami prądowymi*) powiększała się grupa osób współpracująca z profesorem. Od 1997 roku badania są prowadzone przez zespół pracowników Wydziału Elektroniki i Informatyki Politechniki Koszalińskiej, a ich aktualność, jak i wysoką wartość naukową otrzymanych wyników potwierdza fakt, że badania te były wykonywane w ramach 3 projektów badawczych finansowanych przez Polski Komitet Badań Naukowych (KBN): grantu KBN nr 8T11B 042 14 (lata 1998-1999, kierownik prof. Andrzej Guziński), grantu KBN nr 7T11B 004 20 (lata 2001-2003, kierownik prof. Michał Białko) oraz grantu KBN nr 3T11B 059 26, (lata 2004-2006, kierownik dr hab. inż. Oleg Maslennikow). Ponadto zespół badawczy, do którego od 1999 roku należy również autor niniejszej rozprawy doktorskiej, opracował wiele publikacji naukowych poświęconych w/w tematyce – referatów wygłoszonych na konferencjach krajowych, międzynarodowych i IEEE oraz artykułów opublikowanych w czasopismach krajowych, międzynarodowych i PAN, a trzech członków zespołu badawczego wykorzystało wyniki swoich badań nad bramkami i układami prądowymi we własnych rozprawach doktorskich (dr inż. Natalia Maslennikowa i dr inż. Piotr Pawłowski) oraz w rozprawie habilitacyjnej (dr hab. inż. Oleg Maslennikow).

Zgodnie z koncepcją prof. A. Guzińskiego bramki prądowe i złożone z nich układy cyfrowe powstały jako alternatywa dla klasycznych bramek CMOS pracujących w trybie napięciowym i złożonych z nich układów (dalej w pracy nazywanych skrótowo *bramkami* i *układami napięciowymi*) w przypadku konstruowania mieszanych analogowo-cyfrowych systemów VLSI, tzw. systemów jednoukładowych SoC (*ang.* SoC – System-on-Chip). W bramkach prądowych stany logiczne (np. „0" i „1") reprezentowane są przez określone wartości natężenia prądu, a moc pobierana przez bramkę ze źródła zasilania (a dokładniej wartość prądu pobieranego ze źródła zasilania) jest stała, tj. nie zależy od stanu, w którym bramka się znajduje (poziom logicznego „0", logicznej „1" lub zmiana poziomu na wyjściu bramki). Z tego powodu bramki prądowe cechują się znacznie mniejszym (do 20 razy, w zależności od wymaganej szybkości pracy) poziomem zakłóceń podłożowych

powstających podczas pracy części cyfrowej układu scalonego (jest to tzw. szum cyfrowy) i nie zakłócają pracy układów analogowych.

Badania zespołu doprowadziły do rozwoju koncepcji profesora Guzińskiego. Opracowano binarne bramki prądowe (tj. bramki działające w logice binarnej) czterech różnych typów oraz cyfrowe układy prądowe realizujące w tej logice podstawowe operacje arytmetyczne, różnego rodzaju układy kombinacyjne i sekwencyjne, itd., z których część została wyprodukowana w postaci układów ASIC i zweryfikowana praktycznie. Równolegle prowadzono badania nad algebrą logiki bramek prądowych (w skrócie algebry prądowej), która wprowadza matematyczny aparat do opisu i minimalizacji funkcji logicznych przeznaczonych do realizacji w układach prądowych. Algebra ta ma cechy logiki wielowartościowej MVL (ang. Multiple-Valued Logic), dlatego różni się od algebry Boole'a. Opracowano m.in. podstawowe aksjomaty i tożsamości algebry bramek prądowych oraz wyrażenia do konwersji wyrażeń boolowskich w odpowiednie wyrażenia algebry prądowej (i na odwrót). Dowiedziono, że dowolna funkcja binarna może być zrealizowana w oparciu o bramki prądowe, przy czym złożoność sprzętowa otrzymanego układu prądowego pod względem liczby wykorzystanych bramek jest jednakowa (a w niektórych przypadkach nawet mniejsza) od złożoności sprzętowej układu zbudowanego z klasycznych (napięciowych) bramek CMOS. Jednak pod względem liczby tranzystorów układ prądowy będzie bardziej złożonym, ponieważ bramki prądowe składają się z większej liczby tranzystorów niż bramki CMOS. Z tego powodu **głównym celem** przedstawionych w rozprawie badań jest dostosowanie znanych metod minimalizacji funkcji logicznych (np. Quine'a-McCluskey'a, Veitcha-Karnaugha lub innych) do minimalizacji funkcji binarnych w algebrze bramek prądowych w ten sposób, żeby zmniejszyć złożoność sprzętową otrzymanego układu prądowego pod względem zajmowanej przez niego w układzie ASIC powierzchni (tj. właśnie pod względem liczby wykorzystanych tranzystorów oraz ewentualnie pod względem liczby połączeń w układzie) oraz wykorzystanie zmodyfikowanych metod do projektowania różnego rodzaju prądowych jednostek operacyjnych.

Najbardziej charakterystyczną operacją w algebrze bramek prądowych jest operacja dodawania algebraicznego, która jest realizowana sprzętowo poprzez połączenie wyjść bramek reprezentujących poszczególne argumenty operacji w jeden węzeł (tzn. jest realizowana bez wykorzystania żadnych dodatkowych bramek). Powoduje to możliwość pojawienia się, w każdym takim węźle, poziomów logicznych różniących się od „0" i „1", mimo to, że na wejścia układu podawane są wyłącznie te wartości binarne. Z tego powodu

układy prądowe działają (w przypadku ogólnym) w logice wielowartościowej MVL, a algebra bramek prądowych jest w przypadku ogólnym algebrą wielowartościową (ponieważ może działać na liczbach przedstawionych w systemie liczbowym z podstawą $N > 2$). Ta cecha spowodowała zainteresowanie członków zespołu badawczego projektowaniem układów cyfrowych działających w systemach liczbowych z podstawą $N > 2$, i w szczególności, w arytmetyce resztowej RNS (*ang.* Residue Number System arithmetic) opartej o arytmetykę *modulo N*. W ramach tego kierunku badań opracowano nową koncepcję podstawowej bramki prądowej inwertera działającej w logice N-wartościowej z dowolną podstawą $N < 12$ oraz opracowano bramki pozostałych trzech typów. Nowe bramki mają modułową i regularną budowę, zawierają moduły (bloki) maksymalnie czterech różnych typów, a liczba modułów w bramce zależy od podstawy N systemu liczbowego. W oparciu o nowe bramki członkowie zespołu opracowali projekty układów prądowych sumatorów jednocyfrowych i przerzutników typu D przeznaczonych do działania w systemie liczbowym z podstawą N oraz sumatorów i bloków mnożących *modulo N*. Wstępne oszacowania złożoności sprzętowej opracowanych układów wykazały, że są one prostsze od analogicznych układów napięciowych pod względem liczby wykorzystanych bramek, tranzystorów i linii połączeń. Z tego powodu **drugim celem badań** autora niniejszej rozprawy było wykazanie zalet stosowania układów prądowych w układach cyfrowych przeznaczonych do działania w arytmetyce resztowej RNS i N-wartościowej poprzez optymalizację starszych i opracowanie nowych, bardziej skomplikowanych projektów układów prądowych (np. wielooperandowych sumatorów resztowych, bloków mnożących przez wartość stałą, konwerterów liczb z systemu binarnego do RNS i na odwrót, itd.) oraz ich porównanie pod względem złożoności sprzętowej z odpowiednimi układami zbudowanymi z klasycznych bramek CMOS.

Wraz z powstaniem projektów coraz bardziej złożonych układów prądowych wzrastała potrzeba opracowania narzędzi programowych wspomagających ich opis, weryfikację i wizualizację na poziomie logicznym, a w ideale – wspomagających także projektowanie takich układów. Autor niniejszej rozprawy od początku był zaangażowany w rozwój tego kierunku badań zespołu ze szczególnym akcentem na komputerowe wspomaganie projektowania binarnych układów prądowych. Dlatego **dodatkowym celem badań** autora było opracowanie narzędzi programowych umożliwiających zautomatyzowanie procesu projektowania i weryfikacji cyfrowych układów prądowych.

Główna teza rozprawy składa się z dwóch następujących części:

1. Zastosowanie opracowanego sposobu minimalizacji binarnych funkcji logicznych opartego o funkcje wzorcowe typu T i XOR wraz ze znanymi metodami minimalizacji (np. Quine'a-McCluskey'a, Veitcha-Karnaugha lub innymi) umożliwia otrzymanie prądowych układów kombinacyjnych cechujących się mniejszą złożonością sprzętową (pod względem zajmowanej w układzie ASIC powierzchni) w porównaniu do analogicznych układów prądowych, a niekiedy nawet w porównaniu do analogicznych układów zaprojektowanych za pomocą znanych metod minimalizacji i zrealizowanych w oparciu o klasyczne bramki CMOS.
2. Zastosowanie bramek prądowych w jednostkach arytmetyczno-logicznych działających w N-wartościowych systemach liczbowych, w tym w systemach resztowych RNS, pozwala znacznie uprościć budowę takich jednostek, zmniejszając równocześnie powierzchnię, poziom generowanych zakłóceń i koszty wytwarzania układu.

Autor rozprawy jest współautorem blisko 20 prac naukowych związanych z tematyką bramek i układów prądowych, które były opublikowane w materiałach konferencji krajowych i międzynarodowych, w tym konferencji pod egidą IEEE. W niniejszej pracy zastosowano osobną numerację dla tych publikacji - z literą A umieszczoną przed numerem kolejnym pracy (od [A1] do [A18]).

Do swoich **osiągnięć naukowych** autor rozprawy zalicza:

1. opracowanie nowego sposobu minimalizacji funkcji binarnych w algebrze bramek prądowych opartego o wykorzystanie funkcji wzorcowych typu T i XOR (tzw. TBloków i XBloków); sposób ten nadaje się do realizacji komputerowej i może być wykorzystany w parze z dowolną wybraną i znaną metodą minimalizacji, a jego zastosowanie umożliwia zaprojektowanie prądowych układów kombinacyjnych cechujących się mniejszą złożonością sprzętową w porównaniu do analogicznych układów zaprojektowanych za pomocą wyłącznie wybranej znanej metody i realizowanych w oparciu o bramki prądowe, a niekiedy nawet o bramki klasyczne CMOS;
2. algorytmy wyszukiwania n-argumentowych funkcji wzorcowych typu T w tablicy prawdy zadanej N-argumentowej funkcji binarnej ($N \geq n$) lub wśród zadanego zbioru jej implikantów prostych; algorytmy te nadają się do bezpośredniej realizacji komputerowej, a ich złożoność obliczeniowa jest porównywalna ze złożonością

obliczeniową metody Quine'a-McCluskey'a (jeśli funkcja wejściowa jest zadana w postaci tablicy prawdy) lub jest rzędu $O(2 \cdot L^2)$ (jeśli funkcja wejściowa jest zadana w postaci listy implikantów prostych, a L – to jest liczba (N - 1)-argumentowych implikantów prostych);

3. wyrażenia logiczne opisujące funkcje logiczne XOR dla różnej liczby argumentów w algebrze bramek prądowych, których stosowanie pozwala uprościć złożoność sprzętową projektowanych układów prądowych;
4. projekty układów prądowych cechujących się porównywalną liczbą tranzystorów i powierzchnią w porównaniu do układów napięciowych, m.in. sumatora 4-bitowego z równoległym przeniesieniem (ang. look-ahead adder), modułu szybkiego przeniesienia komórki CLB (*ang.* Configurable Logic Block) układów FPGA Virtex oraz układów kombinacyjnych realizujących funkcje S-bloku w algorytmie kryptograficznym DES;
5. udział w opracowaniu koncepcji nowych bramek prądowych o regularnej i modułowej budowie przeznaczonych do wykorzystania w układach działających w systemach liczbowych z podstawą $N > 2$ i w arytmetyce resztowej RNS;
6. projekty układów prądowych m.in. sumatorów, układów mnożących przez stałą oraz konwerterów liczb z systemu binarnego (BIN) do systemu resztowego i odwrotnie, działających w arytmetykach N-wartościowej i resztowej RNS, cechujących się mniejszą liczbą bramek, połączeń, tranzystorów, a w niektórych projektach nawet mniejszą liczbą komórek pamięci ROM i sumatorów w porównaniu do analogicznych układów napięciowych;
7. opracowanie i weryfikacja modeli w/w projektów za pomocą opracowanych z udziałem autora narzędzi programowych oraz udział w ich weryfikacji praktycznej.

Praca zawiera wykaz skrótów, wstęp, cztery rozdziały, podsumowanie i bibliografię, w dodatkach zamieszczono spis tabel i rysunków.

Rozdział pierwszy krótko przedstawia bieżący stan dotychczasowych badań zespołu nad bramkami i układami prądowymi. W tym rozdziale przedstawiono pierwotną koncepcję bramki prądowej inwertera binarnego autorstwa prof. Guzińskiego oraz jej modyfikację dokonaną przez członków zespołu badawczego w celu ulepszenia podstawowych parametrów technicznych bramki. Zaprezentowano podstawowe typy bramek binarnych, strukturę modułową bramek wielowartościowych oraz zasadę konstruowania bramek N-wartościowych w oparciu o moduły K, I, AI, SI. Ponadto, krótko przedstawiono tożsamości i aksjomaty

algebry bramek prądowych oraz sposoby minimalizacji funkcji logicznych w tej algebrze wraz z projektami układów prądowych różnego stopnia złożoności opracowanych w zespole badawczym.

W rozdziale drugim autor przedstawia opis nowego sposobu minimalizacji binarnych funkcji logicznych przeznaczonych do realizacji na bramkach prądowych, który jest oparty o wykorzystanie tzw. funkcji wzorcowych, tj. *n*-argumentowych funkcji binarnych, których realizacja w technologii prądowej jest prostsza od realizacji w technologii napięciowej (na bramkach klasycznych CMOS) nawet pod względem liczby tranzystorów i liczby połączeń w układzie (obliczanej jako ogólna liczba wejść wszystkich bramek w układach napięciowych lub wyjść wszystkich bramek w układach prądowych). Sposób ten pozwala otrzymać prostsze opisy wyżej wymienionych funkcji w porównaniu z ich opisami otrzymanymi za pomocą znanych metod minimalizacji Quine'a-McCluskey'a, Veitcha-Karnaugha, Espresso i in. i jest uzupełnieniem tych metod o opracowane przez autora algorytmy poszukiwania funkcji wzorcowych typu T i wyrażenia opisujące funkcję XOR. Ponadto, opracowany sposób nadaje się do realizacji komputerowej i nie ma ograniczeń na liczbę argumentów N funkcji wejściowej jak również na liczbę n argumentów funkcji wzorcowych. Efektywność zaproponowanego sposobu minimalizacji potwierdzają znajdujące się w tym i w trzecim rozdziale projekty kilku prototypów prądowych standardowych układów cyfrowych (sumatorów i innych jednostek operacyjnych).

W drugiej części tego rozdziału przedstawiono heurystyczny sposób minimalizacji pewnego zbioru wielowartościowych funkcji logicznych $Y=f(x_1, x_2,..., x_p)$, mianowicie funkcji MVL z podstawą $N>2$, niesymetrycznych ($Y \in \{ 0, 1, 2,..., N-1\}$) lub symetrycznych ($Y \in \{ -(N/2-1), ... ,-1, 0, 1,..., (N/2-1)\}$), których argumenty są binarne, tj. $x_i \in \{ 0, 1\}$, $i=1,2,...,p$. Takie funkcje w pracy nazwane są funkcjami N-wartościowymi (odpowiednio symetrycznymi lub niesymetrycznymi) z argumentami binarnymi, a opracowany sposób ich minimalizacji nadaje się do projektowania prądowych układów kombinacyjnych przeznaczonych do wykorzystania w jednostkach arytmetycznych działających w systemach RNS lub w systemach liczbowych z podstawą $N > 2$. Sposób ten jest oparty o właściwości algebry prądowej, m.in. o możliwość realizacji operacji arytmetycznego dodawania i odejmowania poprzez połączenie wyjść bramek reprezentujących poszczególne argumenty operacji w jeden węzeł (bez wykorzystania żadnych dodatkowych bramek).

Opracowane za pomocą tego sposobu projekty jednostek operacyjnych dla systemu resztowego RNS, jak np. wieloargumentowy sumator *modulo N* czy też układy konwerterów

liczb z systemu RNS do systemu binarnego (BIN) i na odwrót umieszczono w rozdziale trzecim. Należy zaznaczyć że, dzięki wykorzystaniu bramek prądowych pojemność bloku pamięci ROM w konwerterze RNS-BIN udało się skrócić z 2^q do q komórek (gdzie q jest ilością cyfr w reprezentacji liczby w systemie RNS).

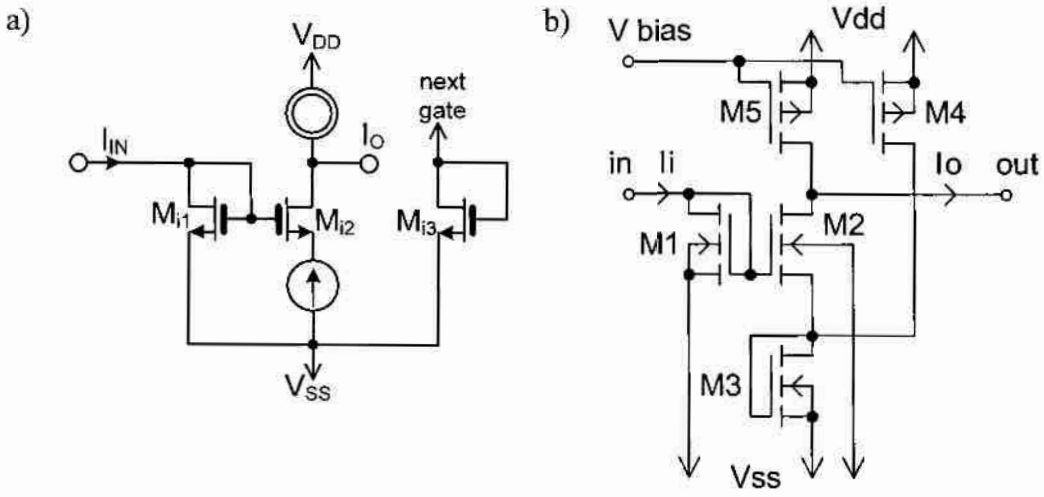
W rozdziale czwartym przedstawiono opis opracowanych z udziałem autora narzędzi programowych służących do konstruowania i weryfikacji układów prądowych na poziomie logicznym - MPUK dla układów binarnych i StreamSim dla układów wielowartościowych i RNS. Ponadto, przedstawiono opis opracowanej z udziałem autora biblioteki bramek prądowych, która pozwala na wykorzystanie środowiska Active-HDL firmy Aldec, powszechnie używanego do projektowania napięciowych układów cyfrowych, do opisu, symulacji i wizualizacji układów prądowych.

Rozprawa kończy się podsumowaniem, w którym zaprezentowano najważniejsze wyniki badań autora.

1. Cyfrowe układy prądowe: stan obecny, perspektywy rozwoju i zastosowania

1.1. Koncepcje, typy i właściwości logiczne binarnych bramek prądowych

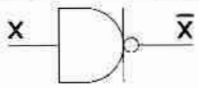
Koncepcja bramki opracowana przez profesora A. Guzińskiego zakłada, że wartości logiczne w układach prądowych będą reprezentowane przez odpowiednie wartości natężenia prądu: wartości logicznego „0" odpowiada natężenie prądu bliskie 0μA, a logicznej „1" - prąd o ustalonej wartości natężenia (np. 30μA dla technologii krzemowej CMOS 0,6μm) [11]. Pierwotna koncepcja zakładała, że poziomowi logicznej „1" na wejściu bramki prądowej (prąd wpływający) odpowiada poziom logicznego „0" na jej wyjściu, natomiast dla poziomu logicznego „0" na wejściu odpowiada poziom logicznej „1" na wyjściu. Podstawowa bramka zaproponowana przez profesora Guzińskiego realizuje zatem funkcję boolowskiej negacji (NOT) i stąd jej nazwa: bramka prądowa inwertera (rys. 1.1). Szczegółowy opis budowy, zasady działania oraz parametrów elektrycznych tej bramki przedstawione są w pracy [12].



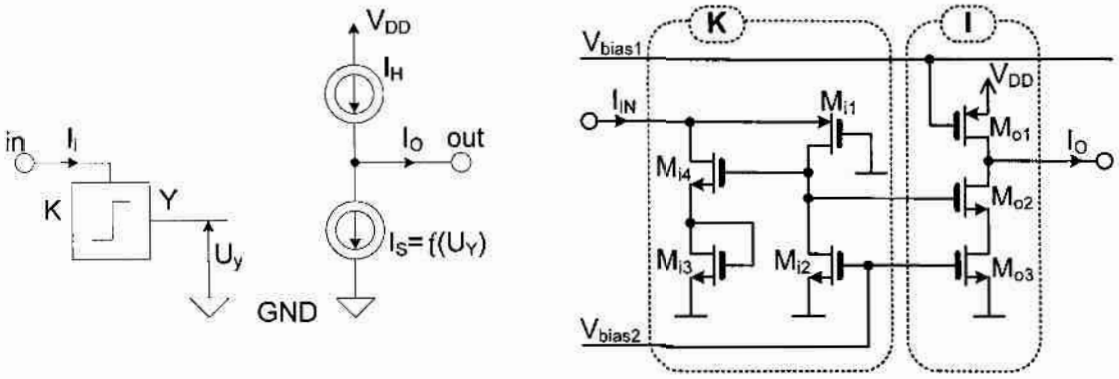
Rys. 1.1. Schemat ideowy (a) oraz przykładowa realizacja (b) bramki prądowej inwertera

W trakcie kilkuletnich badań prowadzonych przez zespół badawczy koncepcja podstawowej bramki prądowej została zmodyfikowana w celu ulepszenia jej parametrów elektrycznych i dynamicznych oraz zaprojektowano i zweryfikowano bramki prądowe trzech innych typów [7, 8, 20, 26, 28]. Obecna koncepcja bramki prądowej z wyjściem typu inwerter i jej realizacja w technologii CMOS przedstawione są odpowiednio na rys. 1.2a i rys. 1.2b. Bramka ta ma oznaczenie skrótowe R1 i realizuje funkcję logiczną (1.1).

Tab. 1.1. Bramka prądowa z wyjściem typu inwerter

Nazwa	Symbol	Realizowana operacja
R1		$y_1 = \bar{x} = \begin{cases} 1 \text{ , dla } x = 0, -1, -2, \ldots \\ 0 \text{ , dla } x = 1, 2, 3, \ldots \end{cases}$ (1.1)

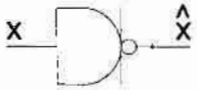
a) b)



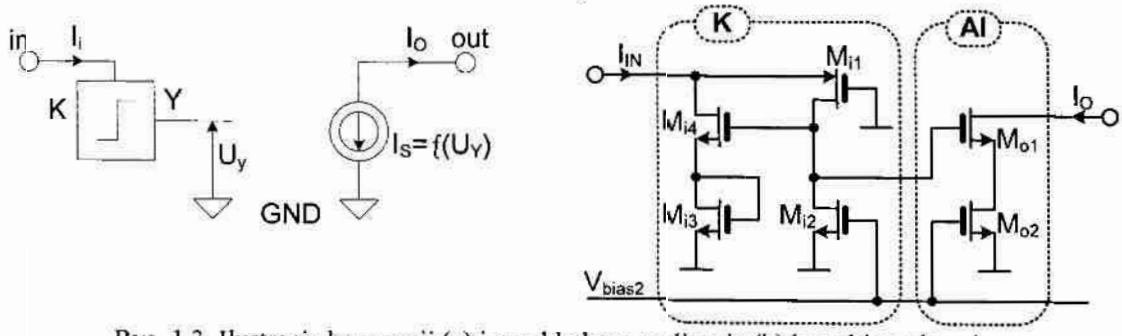
Rys. 1.2. Ilustracja koncepcji (a) i przykładowa realizacja (b) bramki prądowej z wyjściem typu inwerter

Na rys. 1.3 pokazany jest drugi typ bramki prądowej z wyjściem typu anty-inwerter (w skrócie: anty-inwerter lub R2). Funkcję logiczną realizowaną przez tę bramkę przedstawia wyrażenie (1.2). Bramka ta nie pobiera prądu ze źródła zasilania bezpośrednio, prąd może wpływać do jej wejścia lub wyjścia z wyjść innych bramek lub spoza układu.

Tab. 1.2. Bramka prądowa z wyjściem typu anty-inwerter

Nazwa	Symbol	Realizowana operacja
R2		$y_2 = \hat{x} = \begin{cases} 0 \text{ , dla } x = 0, -1, -2, \ldots \\ -1 \text{ , dla } x = 1, 2, 3, \ldots \end{cases}$ (1.2)

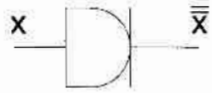
a) b)



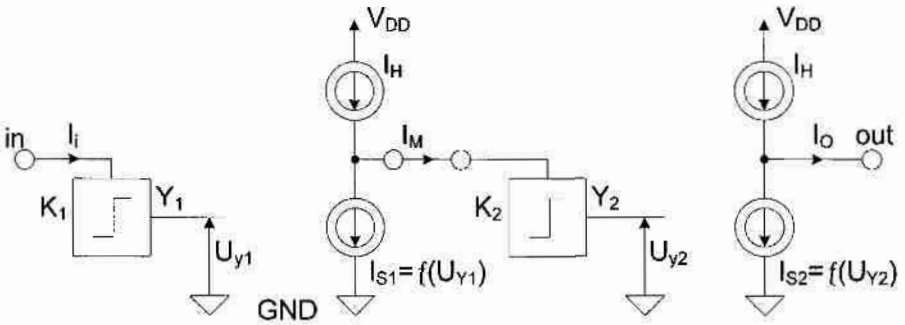
Rys. 1.3. Ilustracja koncepcji (a) i przykładowa realizacja (b) bramki prądowej z wyjściem typu anty-inwerter

Rys. 1.4 przedstawia bramkę prądową trzeciego typu z wyjściem typu podwójny inwerter (w skrócie: podwójny-inwerter lub R3). Funkcję logiczną realizowaną przez tę bramkę przedstawia wyrażenie (1.3).

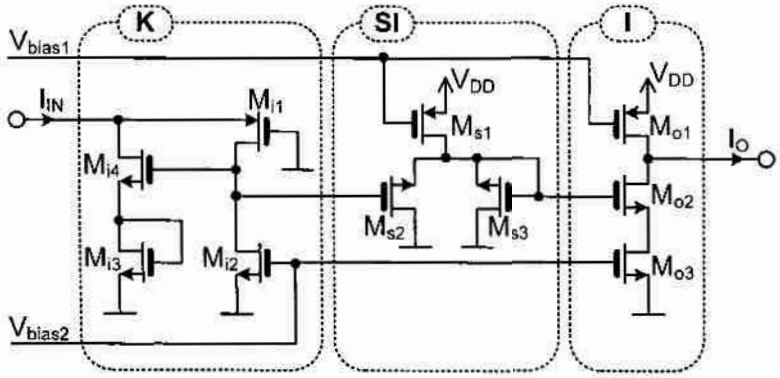
Tab. 1.3. Bramka prądowa z wyjściem typu podwójny-inwerter

Nazwa	Symbol	Realizowana operacja
R3		$y_3 = \overline{\overline{x}} = \begin{cases} 0 \text{ , dla } x = 0, -1, -2, \ldots \\ 1 \text{ , dla } x = 1, 2, 3, \ldots \end{cases}$ (1.3)

a)



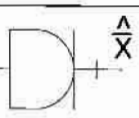
b)



Rys. 1.4. Ilustracja koncepcji (a) i przykładowa realizacja (b) bramki prądowej z wyjściem typu podwójny-inwerter

Czwartym typem jest bramka prądowa z wyjściem typu anty-podwójny-inwerter (w skrócie: anty-podwójny-inwerter lub R4). Rys. 1.5 przedstawia koncepcję tej bramki. Funkcję logiczną realizowaną przez tę bramkę przedstawia wyrażenie (1.4).

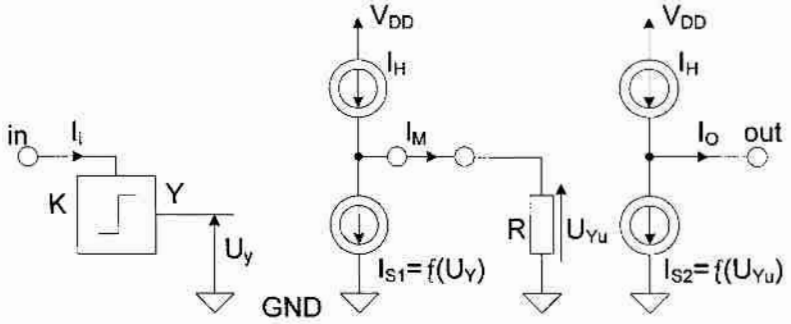
Tab. 1.4. Bramka prądowa z wyjściem typu anty-podwójny-inwerter

Nazwa	Symbol	Realizowana operacja
R4		$y_4 = \hat{\hat{x}} = \begin{cases} -1 \text{ , dla } x = 0, -1, -2, \ldots \\ 0 \text{ , dla } x = 1, 2, 3, \ldots \end{cases}$ (1.4)

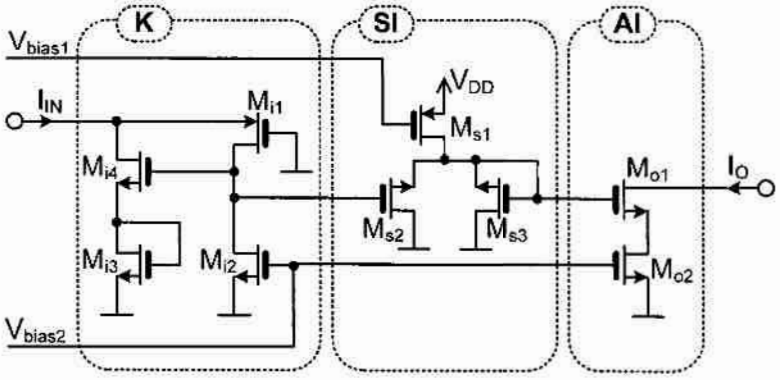
Wszystkie podane powyżej oznaczenia skrótowe stosowane są tylko do bramek, które posiadają wyjścia jednego typu. Dokładny opis przedstawionych bramek prądowych znajduje się w pracy [26].

Analiza wyrażeń (2.1) – (2.4) reprezentujących funkcje logiczne y1,..., y4 pokazuje, że na wyjściach bramek mogą się pojawiać tylko dwie różne wartości (poziomy logiczne): „0" i „1" lub „0" i „–1" (mimo to, że na wejściach tych bramek mogą pojawiać się również inne wartości całkowite).

a)



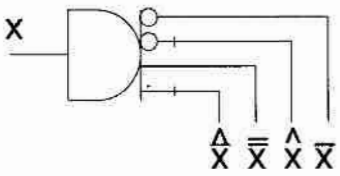
b)



Rys. 1.5. Ilustracja koncepcji (a) i przykładowa realizacja (b) bramki prądowej z wyjściem typu anty-podwójny-inwerter

Z tego powodu bramki przedstawione na rys. 1.2 – rys. 1.5 w rozprawie nazywane są bramkami binarnymi.

Technologia bramek prądowych pozwala na realizację binarnych bramek wielowyjściowych, w tym z wyjściami różnych typów. Realizacja sprzętowa funkcji logicznych w oparciu o bramki wielowyjściowe pozwala zmniejszyć liczbę tranzystorów potrzebnych do realizacji odpowiedniego układu prądowego w porównaniu do analogicznego układu zawierającego wyłącznie bramki jednowyjściowe, a zatem możliwe jest zmniejszenie powierzchni projektowanego układu. Na rys. 1.6 przedstawiono przykład realizacji prądowej bramki binarnej z wyjściami czterech różnych typów, której oznaczenie, zgodnie z powyższą notacją przedstawia zapis R1234.



Rys. 1.6. Przykład bramki wielowyjściowej o czterech różnych typach wyjść (R1234)

Przykładowo zapis R223 oznacza bramkę prądową 3-wyjściową z dwoma wyjściami typu anty-inwerter i z jednym wyjściem typu podwójny inwerter. Badania eksperymentalne bramek wielowyjściowych wykazały, że maksymalną liczbę wyjść bramki prądowej warto ograniczyć liczbą 12.

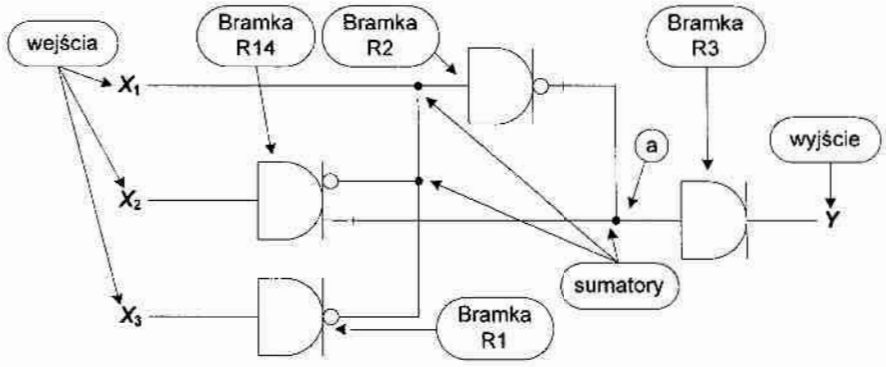
W kolejnych rozdziałach przedstawione zostaną przykładowe układy cyfrowe zbudowane w oparciu o bramki prądowe. W związku z tym, że w celu udowodnienia głównych tez pracy niezbędnym będzie wykonanie porównania złożoności sprzętowej otrzymanych układów prądowych i analogicznych układów napięciowych (zrealizowanych w oparciu o klasyczne bramki CMOS). Układy te porównywane będą pod względem liczby użytych bramek i połączeń (tj. liczby wyjść wszystkich bramek w układach prądowych i wejść wszystkich bramek w układach napięciowych) oraz pod względem liczby tranzystorów potrzebnych do ich realizacji. Na podstawie przedstawionych w pracy [26] oraz na rys. 1.2 - rys. 1.5 schematów bramek prądowych (na poziomie tranzystorów) w niniejszej rozprawie przyjęto, że:

- każda binarna bramka prądowa posiada jeden układ wejściowy składający się z 4 tranzystorów (jest to tzw. moduł K, przedstawiony na rys. 2.2b, rys. 2.3b, rys. 2.4b i rys. 2.5b);
- każde wyjście bramki binarnej typu inwerter jest realizowane za pomocą 3 tranzystorów (jest to tzw. moduł I, przedstawiony na rys. 2.2b);
- każde wyjście bramki binarnej typu anty-inwerter to dodatkowe 2 tranzystory (jest to moduł AI, przedstawiony na rys. 2.3b);
- bramki binarne z wyjściem typu podwójny-inwerter i anty-podwójny-inwerter zawierają dodatkowo uproszczony układ inwersji składający się z 3 tranzystorów – tylko jeden układ w jednej bramce (jest to tzw. moduł SI, przedstawiony na rys. 2.4b i rys. 2.5b);
- każde wyjście bramki typu podwójny-inwerter jest realizowane za pomocą jednego modułu I oraz (jednego dla danej bramki) modułu SI;

- każde wyjście bramki typu anty-podwójny-inwerter jest realizowane za pomocą modułu AI oraz (jednego dla danej bramki) modułu SI.

Na podstawie tych wartości można obliczyć z ilu tranzystorów składa się dowolna bramka binarna lub układ zbudowany z binarnych bramek prądowych. Przykładowo bramka z rys. 1.6 składa się z 17 tranzystorów – układ wejściowy zawiera 4 tranzystory, a poszczególne wyjścia bramki (idąc z góry w dół) składają się odpowiednio z 3, 2, (3+3) i 2 tranzystorów.

Najbardziej charakterystyczną operacją w algebrze bramek prądowych jest operacja dodawania algebraicznego, która jest realizowana sprzętowo poprzez połączenie wyjść bramek reprezentujących poszczególne argumenty operacji w jeden węzeł (tzn. jest realizowana bez wykorzystania żadnych dodatkowych bramek). Z tego powodu sumator, tj. układ realizujący operację dodawania, na schematach oznaczany jest symbolem kropki. Na rys. 1.7 pokazany jest przykładowy schemat układu prądowego realizujący przykładową funkcję $Y = f(X_1, X_2, X_3)$ (której tabelę prawdy przedstawia tab. 1.5) i prezentujący różne elementy opisujące układy prądowe.



Rys. 1.7. Schemat ilustrujący różne elementy występujące na schematach układów zbudowanych z binarnych bramek prądowych

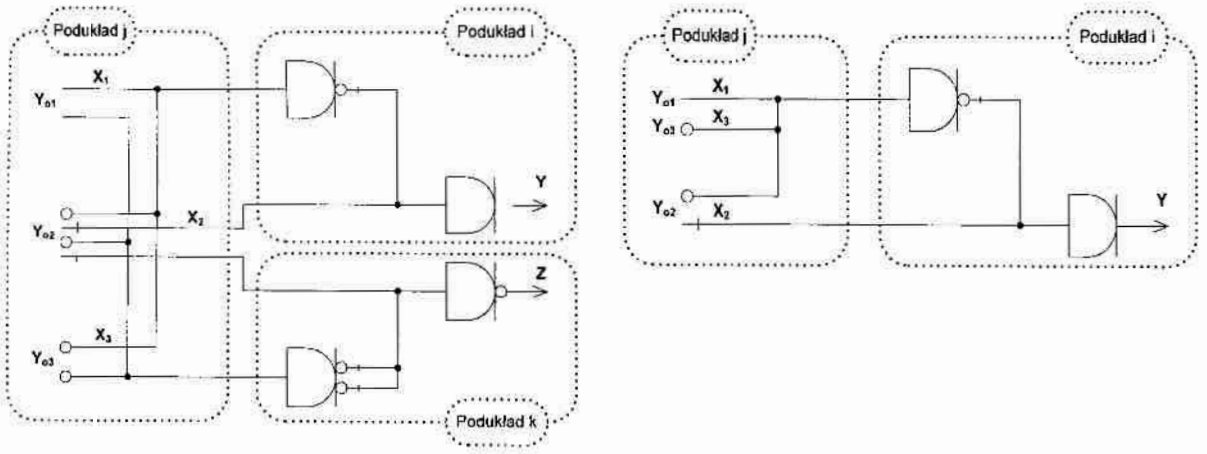
Tab. 1.5. Tablica prawdy układu z rys. 1.7

X_1	X_2	X_3	a	Y
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	-1	0
1	0	1	-1	0
1	1	0	0	0
1	1	1	-1	0

Złożoność tego układu prądowego pod względem liczby tranzystorów oblicza się następująco: układ składa się z 4 bramek, cztery układy wejściowe bramek zawierają 16 tranzystorów, 2 wyjścia bramek typu inwerter – 6 tranzystorów, 1 wyjście typu anty-inwerter – 2 tranzystory, 1 wyjście typu podwójny-inwerter – 3 tranzystory i 1 wyjście typu anty-podwójny-inwerter – 2 tranzystory. Dwie bramki zawierają wyjścia typu podwójny-inwerter i anty-podwójny-inwerter, na co dodatkowo potrzebne są 2 uproszczone układy inwersji SI, tj. 6 tranzystorów, co razem daje liczbę 35 tranzystorów. Należy zaznaczyć, że w przypadku, gdy przedstawiony układ nie reprezentuje całego systemu, a jest tylko jego częścią (*i*-tym podukładem większego systemu), liczba tranzystorów potrzebnych do jego realizacji może być mniejsza. Właśnie ten przypadek, który jest znacznie bardziej rozpowszechniony w praktyce, jest brany pod uwagę w niniejszej rozprawie podczas obliczania złożoności sprzętowej układów prądowych i napięciowych. W tym przypadku, jednak też można wyodrębnić dwie różne sytuacje.

Pierwszą reprezentuje rys. 1.8a, gdzie zakłada się, że źródłem sygnału wejściowego X_3 jest bramka należąca do innego *j*-tego podukładu systemu, która przekazuje ten sygnał również do innych podukładów systemu.

a) b)



Rys. 1.8. Ilustracja realizacji źródeł sygnałów wejściowych do i-tego podukładu, gdy jest on: jednym z podukładów (a) oraz jedynym podukładem (b) podłączonym do podukładu *j*-ego

W tej sytuacji bramka R1 na wejściu X_3 rozpatrywanego *i*-tego podukładu może zostać wyeliminowana, a sygnał zanegowany X_3 może być produkowany w *j*-tym podukładzie poprzez dołożenie do odpowiedniej bramki prądowej dodatkowego wyjścia typu inwerter (dołożenie modułu I). Z tego powodu liczba tranzystorów w rozpatrywanym układzie będzie pomniejszona o 4 tranzystory. Podobne zmniejszenie liczby tranzystorów można uzyskać dla

wejścia X_2 rozpatrywanego układu, ponieważ bramka R14 może być wyeliminowana, a moduły typu I i AI mogą być dołożone do bramki, należącej do j-tego podukładu. Pozwala to na zaoszczędzenie kolejnych 4 tranzystorów tworzących układ wejściowy bramki R14 oraz 3 tranzystorów potrzebnych do utworzenia uproszczonego układu inwersji (modułu SI). Ostatecznie liczba tranzystorów LT, niezbędna do realizacji tego układu, wynosi LT= 24.

Druga sytuacja jest przedstawiona na rys. 1.8b i zakłada, że j-ty podukład produkuje sygnał X_3 wyłącznie dla i-tego podukładu. W tym przypadku bramka R1 na wejściu X_3 rozpatrywanego i-tego podukładu może zostać wyeliminowana, a wyjście typu podwójny inwerter w bramce produkującej sygnał X_3 (w j-tym podukładzie) może być zamienione na wyjście typu inwerter. Z tego powodu liczba tranzystorów w rozpatrywanym i-tym podukładzie będzie pomniejszona o 7 tranzystorów (w porównaniu do układu z rys. 1.7), a w j-tym podukładzie – jeszcze o 3 tranzystory (ponieważ znika moduł SI w bramce produkującej sygnał X_3). Ostatecznie, minimalna liczba tranzystorów $minLT$, niezbędna do realizacji tego układu, wynosi $minLT$ = 18.

W dalszej części rozprawy przy ocenie złożoności sprzętowej zaprojektowanych układów prądowych będą podawane obie wartości liczby tranzystorów potrzebnych do realizacji układu: LT i $minLT$. Wartości te będą obliczane tak, jak zostało opisane powyżej.

1.2. Krótki przegląd algebry bramek prądowych

1.2.1. Podstawowe elementy i tożsamości algebry bramek prądowych

Specyficzne właściwości przedstawionych powyżej binarnych bramek prądowych oraz operacje, które one wykonują świadczą, że algebra bramek prądowych nie jest algebrą binarną. Z tego powodu istotnym stało się określenie zasad tworzenia układów cyfrowych z wykorzystaniem bramek prądowych oraz sformalizowanego sposobu zapisu wyrażeń logicznych powstających przy ich użyciu. Tak powstała tzw. algebra bramek prądowych lub w skrócie *algebra prądowa*, której podstawy zostały szczegółowo opisane w pracach [19, 26]. W niniejszym rozdziale przedstawiono wyłącznie te elementy algebry prądowej, które będą potrzebne do zrozumienia następnych rozdziałów rozprawy.

Zbiorem elementów w algebrze bramek prądowych jest zbiór liczb całkowitych, dodatkowo określono trzy elementy wyróżnione {-1, 0, 1}, które pojawiają się na wyjściach binarnych bramek prądowych. Należy zaznaczyć, że w układach cyfrowych złożonych z bramek binarnych na wejściu i na wyjściu mogą pojawiać się tylko wartości „0" i „1" (jak np. w układzie przedstawionym na rys. 1.7), natomiast wewnątrz układu mogą pojawiać

się inne wartości całkowite (poziomy logiczne), np. „–1", „2" itd. (co reprezentuje tab. 1.5). Określone zostały podstawowe operacje logiczne jakie wykonują poszczególne bramki (operacje jednoargumentowe, wymienione w punkcie 1.1) i wieloargumentowa operacja sumy algebraicznej. Określono sposób zapisu wyrażeń w algebrze bramek prądowych. Przyjęto podstawowe aksjomaty (1.5) – (1.7) stwierdzające, że:

- operacja sumy algebraicznej jest działaniem przemiennym:

$$a+b=b+a \tag{1.5}$$

- operacja sumy algebraicznej jest działaniem łącznym:

$$(a+b)+c=b+(a+c) \tag{1.6}$$

- elementem neutralnym jest element wyróżniony „0":

$$a+0=a \tag{1.7}$$

Określono starszeństwo operacji: w przypadku dowolnego wyrażenia najpierw wykonuje się operację dolną, a następnie górną. Przykładowe wyrażenie złożone (1.8) należy zatem interpretować w następujący sposób:

$$f=\overline{\bar{a}+\hat{b}} \tag{1.8}$$

1) $f_1=\bar{a}$

2) $f_2=\hat{b}$

3) $f_3=f_1+f_2$

4) $f_4=\overline{f_3}$

Ponadto, w oparciu aksjomaty (1.5) – (1.7), dla dowolnych zmiennych a, b oraz a_i (gdzie $i=1\ldots n$) określono podstawowe tożsamości algebry bramek prądowych:

$$\bar{\bar{a}}=\bar{a} \tag{1.9}$$

$$\hat{\hat{a}}=\hat{a} \tag{1.10}$$

$$\overline{a_1-a_2}=\overline{a_2-a_1} \tag{1.11}$$

$$\hat{a}_1+\hat{a}_2+\ldots+\hat{a}_n=0 \tag{1.12}$$

$$\overline{\bar{a}_1+\bar{a}_2+\ldots+\bar{a}_n}=\overline{\bar{a}_1+\hat{a}_2+\ldots+\hat{a}_n} \tag{1.13}$$

$$\overline{a_1+a_2+\ldots+a_n}=\bar{a}_1-a_2-\ldots-a_n \tag{1.14}$$

$$\overline{a_1+a_2+\ldots+a_n}=\bar{a}_1+\hat{a}_2+\ldots+\hat{a}_n \tag{1.15}$$

Tożsamości (1.9) i (1.10) stwierdzają, że podwójną negację można pominąć i funkcja logiczna nie zmieni wartości. Wyrażenie (1.11) wskazuje na swoistą przemienność inwersji i różnicy dwóch zmiennych. Wyrażenia (1.12) – (1.15) tworzą tożsamości związane z sumą

argumentów. Wyrażenie (1.12) stwierdza, że w wyniku anty-inwersji (lub jej sumy) zawsze otrzymamy wartość „0". Wyrażenia (1.13) – (1.15) wskazują na związki pomiędzy sumą inwersji i różnicą (anty-inwersją) podwójnej inwersji.Należy zaznaczyć, że w przypadku, gdy zmienne wejściowe a, b oraz a_i są zmiennymi binarnymi, zbiór tożsamości (1.9) – (1.15) można poszerzyć o dodatkowe tożsamości (1.16) – (1.22), które mogą zostać wykorzystane przy minimalizacji binarnych funkcji logicznych:

$$\bar{a} = \hat{a} + 1 \tag{1.16}$$

$$\hat{a} = \bar{a} + (-1) \tag{1.17}$$

$$\overline{a + a + \ldots + a} = \bar{a} \tag{1.18}$$

$$\widehat{a + a + \ldots + a} = \hat{a} \tag{1.19}$$

$$\overline{\bar{a}_1 + \bar{a}_2 + \ldots + \bar{a}_n} = \overline{1 + \hat{a}_1 + \hat{a}_2 + \ldots + \hat{a}_n} \tag{1.20}$$

$$a + \bar{a} = 1 \tag{1.21}$$

$$\overline{\hat{a}_1 + \hat{a}_2 + \ldots + \hat{a}_n} = 1 \tag{1.22}$$

W pracy [19] udowodniono, że każde wyrażenie boolowskie może być doprowadzone do odpowiedniego wyrażenia akceptowanego w algebrze bramek prądowych (tj. wyrażenia zawierającego operacje dodawania, odejmowania i inwersji (1.1) – (1.4)), dzięki czemu dowolna funkcja binarna może być zrealizowana przez układ zbudowany z binarnych bramek prądowych. Podstawą w/w przekształceń wyrażeń boolowskich są wyrażenia (1.23) – (1.26), reprezentujące zamianę podstawowych operacji boolowskich na odpowiednie operacje algebry bramek prądowych. Należy zaznaczyć, że symbole „$\cdot$", „$\vee$" i „$\bar{}$" (po lewej stronie znaku równości) reprezentują odpowiednio operacje boolowskie AND, OR i NOT, natomiast symbole „+" i „$\bar{}$" (w prawej części) reprezentują operacje sumy i inwersji w algebrze bramek prądowych. Analiza przedstawionych wyrażeń konwersji świadczy o tym, że liczba bramek niezbędnych do realizacji sprzętowej dowolnego z wyrażeń (1.23) – (1.26) jest jednakowa w obu technologiach – zarówno napięciowej, jak i prądowej.

$$a_1 \cdot a_2 \cdot \ldots \cdot a_n = \overline{\bar{a}_1 + \bar{a}_2 + \ldots + \bar{a}_n} \tag{1.23}$$

$$\overline{a_1 \cdot a_2 \cdot \ldots \cdot a_n} = \overline{\bar{a}_1 + \bar{a}_2 + \ldots + \bar{a}_n} \tag{1.24}$$

$$a_1 \vee a_2 \vee \ldots \vee a_n = \overline{\bar{a}_1 + \bar{a}_2 + \ldots + \bar{a}_n} \tag{1.25}$$

$$\overline{a_1 \vee a_2 \vee \ldots \vee a_n} = \overline{a_1 + a_2 + \ldots + a_n} \tag{1.26}$$

1.2.2. Minimalizacja funkcji binarnych w algebrze bramek prądowych

Najważniejszą częścią etapu projektowania układu cyfrowego (jak również projektowania logicznego systemu cyfrowego) jest minimalizacja opisów wszystkich funkcji logicznych, które układ powinien realizować. Minimalizacja opisu funkcji logicznej (w skrócie *minimalizacja funkcji logicznej*) oznacza proces otrzymywania jej reprezentacji w postaci wyrażenia zawierającego jak najmniejszą liczbę argumentów i operacji, co pozwala zmniejszyć złożoność sprzętową projektowanego układu. Ze względu na specyficzne cechy algebry bramek prądowych, znane metody minimalizacji binarnych funkcji logicznych (np. Veitcha-Karnaugha, Quine'a-McCluskey'a, Espresso [4, 25, 32, 34]) nie mogą być stosowane bezpośrednio w nowej algebrze. Z tego powodu przez kilka lat zespół badawczy prowadził prace powiązane z opracowaniem metod i sposobów minimalizacji funkcji logicznych (binarnych i wielowartościowych) przeznaczonych do realizacji na bramkach prądowych. Jeden z najbardziej prostych sposobów minimalizacji funkcji binarnych (tzw. *sposób 1*) oparty jest o wyrażenia konwersji (1.23) – (1.26). Zgodnie z tym sposobem, minimalizację funkcji dokonuje się w algebrze Boole'a w oparciu o dowolną znaną metodę (np. jedną z wyżej wymienionych). Po uzyskaniu zminimalizowanego wyrażenia boolowskiego następuje jego konwersja na odpowiednie wyrażenie algebry prądowej za pomocą wyrażeń (1.23) – (1.26). Dzięki temu wynikowe wyrażenie bezpośrednio nadaje się do realizacji w postaci prądowego układu kombinacyjnego, przy czym liczba bramek w tym układzie będzie jednakowa z liczbą bramek w odpowiednim układzie napięciowym. Potwierdzeniem tej tezy są rys. 1.9 - rys. 1.12, które reprezentują przykłady realizacji w technologiach prądowej i napięciowej podstawowych 2-argumentowych funkcji boolowskich. Tablice prawdy w/w funkcji przedstawiono w tab. 1.6.

Tab. 1.6. Tablice prawdy prostych dwuargumentowych funkcji binarnych

x1	*x2*	*and2*	*nand2*	*or2*	*nor2*	*x2n1*	*nx2n1*	*xor*
0	**0**	0	1	0	1	1	0	0
0	**1**	0	1	1	0	1	0	1
1	**0**	0	1	1	0	0	1	1
1	**1**	1	0	1	0	1	0	0

Należy zaznaczyć, że właściwości logiczne technologii prądowej, mianowicie bardzo prosta realizacja operacji dodawania i odejmowania, zwykle umożliwiają dalszą minimalizację funkcji logicznych. Na przykład, jeśli wyrażenie boolowskie opisujące zadaną

funkcję po minimalizacji przedstawia sobą (lub zawicra) *n*–elementową dysjunkcję $a_1 \vee a_2 \vee \ldots \vee a_n$, przy czym spełniony jest warunek:

$$a_1 \cdot a_2 \cdot \ldots \cdot a_n = 0 \tag{1.27}$$

to do przekształcenia tego wyrażenia w odpowiednie wyrażenie algebry bramek prądowych zamiast wyrażenia konwersji (1.25) można stosować wyrażenie uproszczone (1.28).

W przypadku minimalizacji funkcji logicznych za pomocą diagramów Veitcha-Karnaugha spełnienie warunku (1.27) oznacza utworzenie takich bloków jedynek (lub zer), które nie mają wspólnych kratek. Warunek ten pozwala również uprościć wzór (1.24) do postaci (1.29). Dzięki temu, np. realizacja funkcji boolowskiej $Y = X_1 xor X_2 = X_1 \oplus X_2 = \overline{\overline{X_1} \cdot X_2} \cdot \overline{\overline{X_2} \cdot X_1}$ wymaga wykorzystania trzech 2-wejściowych bramek NAND w technologii napięciowej (tj. 12 tranzystorów), ale dwóch bramek inwerterów w technologii prądowej (co przedstawia rys. 1.15) i wymaga *LTmin*=20 tranzystorów).

$$a_1 \vee a_2 \vee \ldots \vee a_n = a_1 + a_2 + \ldots + a_n \tag{1.28}$$

$$\overline{a_1 \cdot a_2 \cdot \ldots \cdot a_n} = \overline{a_1} + \overline{a_2} + \ldots + \overline{a_n} \tag{1.29}$$

a) b)

Rys. 1.9. Realizacja funkcji *and*2 na bramkach prądowych (a) i napięciowych (b)

a) b)

Rys. 1.10. Realizacja funkcji *nand*2 na bramkach prądowych (a) i napięciowych (b)

a) b)

Rys. 1.11. Realizacja funkcji *or*2 na bramkach prądowych (a) i napięciowych (b)

a) b)

Rys. 1.12. Realizacja funkcji *nor*2 na bramkach prądowych (a) i napięciowych (b)

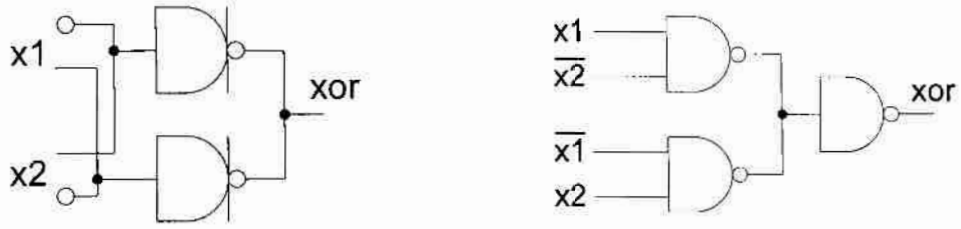
a) b)

Rys. 1.13. Realizacja funkcji *x2n*1 na bramkach prądowych (a) i napięciowych (b)

a) b)

Rys. 1.14. Realizacja funkcji *nx2n*1 na bramkach prądowych (a) i napięciowych (b)

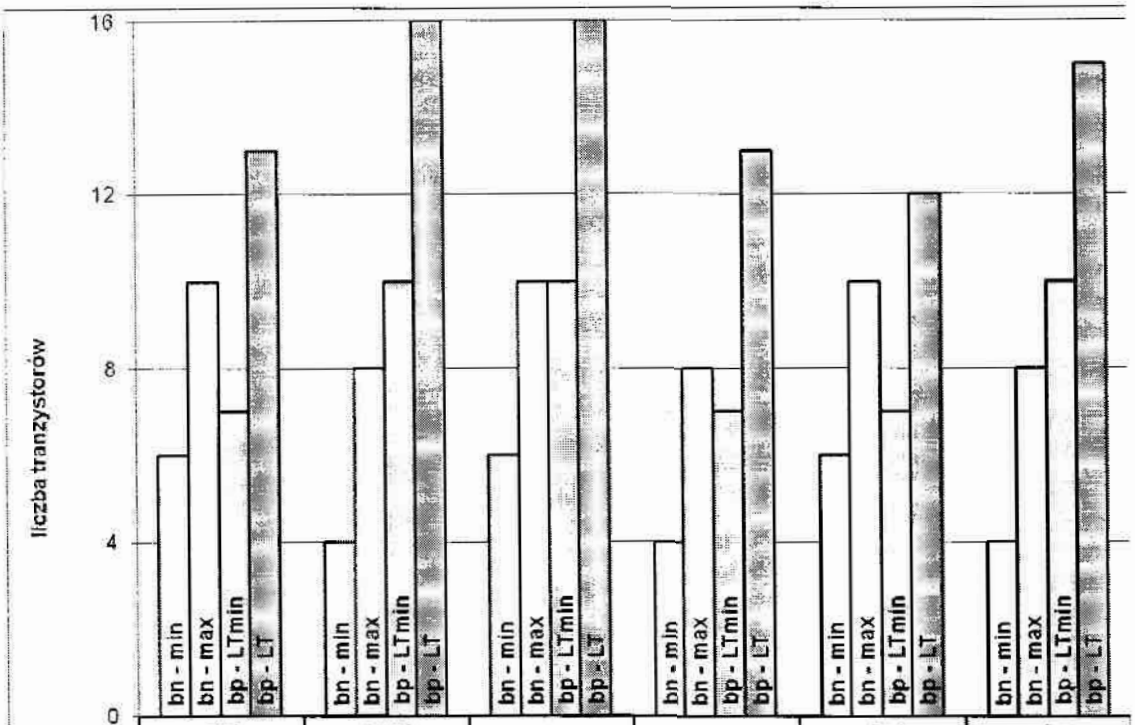
a) b)



Rys. 1.15. Realizacja funkcji *xor* na bramkach prądowych (a) i napięciowych (b)

Jako przykład, na rys. 1.16 przedstawiono dane o liczbach tranzystorów, jakie są potrzebne do realizacji dwuargumentowych funkcji logicznych przedstawionych w tab. 1.6. Zmienne *LT* i *LTmin* na tym rysunku reprezentują liczby tranzystorów w układach prądowych, a zmienne *min* i *max* – liczby tranzystorów w odpowiednich układach napięciowych (przy obliczaniu wartości *max* dodane są ewentualne bramki NOT, które mogą pojawić się na wejściach układu). Z rysunku tego wynika, że układy prądowe realizujące podstawowe dwuargumentowe funkcje boolowskie zwykle składają się z większej liczby tranzystorów.

Jednak przy zwiększeniu liczby argumentów w niektórych z w/w funkcji sytuacja się zmienia na korzyść technologii prądowej.



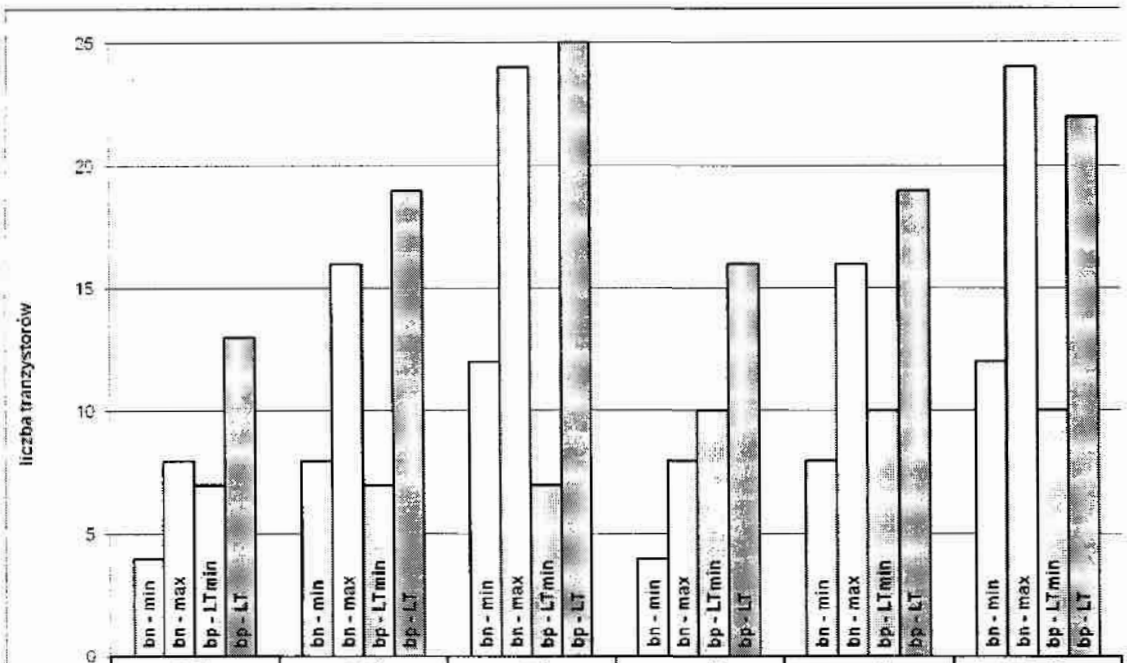
	and	nand	or	nor	x2n1	nx2n1
bn - min	6	4	6	4	6	4
bn - max	10	8	10	8	10	8
bp - LTmin	7	10	10	7	7	10
bp - LT	13	16	16	13	12	15

Rys. 1.16. Porównanie ilości tranzystorów potrzebnych do realizacji wybranych 2-argumentowych funkcji logicznych

Na przykład, na rys. 1.18 zamieszczone zostały dane o liczbach tranzystorów potrzebnych do realizacji dwóch podstawowych wieloargumentowych funkcji boolowskich w obu technologiach – NAND i NOR. Z danych tych wynika, że w technologii prądowej realizacja *k*-argumentowej bramki NOR wymaga mniejszej liczby tranzystorów, niż jej realizacja w technologii napięciowej już dla k>3. Przykładowa realizacja 4-argumentowej funkcji NOR na bramkach prądowych przedstawiona jest na rys. 1.17.

a) b)

Rys. 1.17. Realizacja 4-argumentowej funkcji NOR na bramkach prądowych (a) i napięciowych (b)



	nor2	nor4	nor6	nand2	nand4	nand6
bn - min	4	8	12	4	8	12
bn - max	8	16	24	8	16	24
bp - LTmin	7	7	7	10	10	10
bp - LT	13	19	25	16	19	22

Rys. 1.18. Porównanie ilości tranzystorów potrzebnych do realizacji wybranych wieloargumentowych funkcji logicznych

Podsumowując, główną zaletą opracowanego sposobu jest jego uniwersalność – *sposób 1* nadaje się do minimalizacji dowolnej funkcji binarnej. Dodatkowymi zaletami *sposobu 1* jest to, że:

- jego złożoność obliczeniowa jest równa złożoności obliczeniowej wykorzystanej w nim znanej metody minimalizacji;
- nadaje się on do bezpośredniej realizacji komputerowej;
- nie wymaga od projektanta głębokiej wiedzy o algebrze bramek prądowych.

Jednak sposób ten nie wykorzystuje w pełni właściwości logicznych algebry prądowej, dzięki czemu złożoność sprzętowa otrzymywanych układów prądowych pod względem liczby tranzystorów jest zwykle większa od złożoności odpowiednich układów napięciowych. Dlatego aktualnym i ważnym problemem jest rozwój sposobów minimalizacji funkcji binarnych przeznaczonych do realizacji na bramkach prądowych ukierunkowany na zmniejszenie złożoności sprzętowej otrzymanych układów pod względem zajmowanej powierzchni, tj. właśnie pod względem liczby wykorzystanych tranzystorów oraz ewentualnie pod względem liczby połączeń w układzie. W związku z tym, w rozdziale 2 rozprawy podjęto próbę rozwiązania tego problemu poprzez modyfikację *sposobu 1*.

W większości projektów rzeczywistych układów i systemów cyfrowych wymagane jest wykorzystanie wielowyjściowych układów kombinacyjnych, których projektowanie sprowadza się do minimalizacji nie jednej, lecz kilku funkcji logicznych $Y_1, Y_2, \ldots, Y_k$. Dzięki temu zwykle pojawia się możliwość wykorzystania jednej funkcji logicznej, np. Y_i do realizacji innej (lub innych) funkcji, np. Y_j lub odnalezienie w całym zbiorze funkcji wejściowych pewnego zbioru prostszych podfunkcji $a_1, a_2, \ldots, a_n$ (tzw. *funkcji bazowych*), z których wszystkie wymagane funkcje $Y_1, Y_2, \ldots, Y_k$ mogą być złożone. Algebra bramek prądowych jest dobrze dostosowana do realizacji w/w możliwości, ponieważ udowodniono, że dowolna funkcja logiczna w algebrze prądowej może być przedstawiona jako suma algebraiczna kilku innych funkcji logicznych [19]. W oparciu o to twierdzenie członkowie zespołu opracowali *sposób 2* minimalizacji funkcji logicznych, który wykorzystuje operacje arytmetycznego dodawania i odejmowania (najprościej realizowane operacje w algebrze bramek prądowych). Zgodnie ze *sposobem 2* przy minimalizacji funkcji logicznej $Y=f(X_1, X_2, \ldots, X_m)$ lub zbioru takich funkcji należy dążyć do otrzymania jej opisu w postaci sumy algebraicznej

$$Y = a_1 + a_2 + \ldots + a_n \tag{1.30}$$

kilku prostszych funkcji $a_1, a_2, \ldots, a_n$ o tych samych argumentach, dla których spełniony jest warunek $a_1 \cap a_2 \cap \ldots \cap a_n = \varnothing$. Należy zaznaczyć, że w przypadku minimalizacji funkcji za pomocą diagramów Veitcha-Karnaugha spełnienie tego warunku oznacza utworzenie takich bloków jedynek logicznych (lub zer), które nie mają wspólnych kratek.

Niestety, *sposób 2* jest sposobem heurystycznym, dlatego złożoność sprzętowa otrzymywanych za jego pomocą układów prądowych w dużym stopniu zależy od doświadczenia projektanta. Mimo to, stosowanie *sposobu 2* pozwala w niektórych przypadkach otrzymać cyfrowe układy prądowe prostsze od odpowiednich układów napięciowych nawet pod względem liczby wykorzystanych tranzystorów. Na przykład dla funkcji C_{out} przeniesienia wyjściowego pełnego sumatora jednobitowego FA (*ang.* full-adder), która może być przedstawiona za pomocą wyrażenia boolowskiego $C_{out} = A_i \cdot B_i \vee A_i \cdot C_{in} \vee B_i \cdot C_{in}$ (gdzie C_{in} reprezentuje przeniesienie wejściowe), jako funkcje bazowe $a_1, a_2, \ldots, a_n$ wybrane zostały trzy funkcje $a_1=\overline{A}_i$, $a_2=\overline{B}_i$ i $a_3=\hat{C}_{in}$, a ostateczne wyrażenie dla C_{out} zostało otrzymane jako zanegowana suma algebraiczna funkcji bazowych a_1, a_2, a_3, tj. $C_{out} = \overline{\overline{A}_i + \overline{B}_i + \hat{C}_{in}}$. Ponadto funkcja sumy S_i sumatora FA została otrzymana jako suma algebraiczna jego wszystkich argumentów minus podwojona wartość przeniesienia

wyjściowego, tj. $S_i = A_i + B_i + C_{in} - (C_{out} + C_{out})$. Bardziej szczegółowo proces otrzymania układu jednobitowego sumatora prądowego prostszego pod względem liczby tranzystorów i liczby połączeń od klasycznego sumatora FA zbudowanego z 9 bramek napięciowych [41], jak również od zoptymalizowanego układu FA zbudowanego bezpośrednio z 28 tranzystorów CMOS [29] przedstawiono w podrozdziale 2.3.1. Schemat takiego sumatora prądowego jest przedstawiony na rys. 2.33a.

Należy zaznaczyć, że oba opisane powyżej sposoby minimalizacji funkcji binarnych w algebrze prądowej były wielokrotnie wykorzystywane przez członków zespołu badawczego podczas opracowania projektów prądowych standardowych układów cyfrowych, np. różnego rodzaju dekoderów, sumatorów, multiplekserów, przerzutników, jednostek arytmetyczno-logicznych, a nawet podstawowych elementów komórek układów reprogramowalnych FPGA rodzin Virtex i Spartan2 (bloków LUT i SLICE) [A2-A7, A11, 10, 13, 17- 22].

1.2.3. Bramki prądowe dla logiki wielowartościowej

Doświadczenie zespołu w projektowaniu cyfrowych układów prądowych w oparciu o przedstawione w poprzednim podrozdziale sposoby minimalizacji wykazało, że wewnątrz układu kombinacyjnego realizującego funkcję binarną $Y=f(X_1, X_2, \ldots, X_m)$ (tj. na wejściach niektórych bramek układu) mogą się pojawiać poziomy logiczne różne od „0" i „1". Oznacza to, że w celu minimalizacji funkcji binarnej Y w oparciu o funkcje bazowe $a_1, a_2, \ldots, a_n$ (*sposób 2*) tak aby warunek (1.27) był spełniony, funkcje te można wybierać nie koniecznie ze zbioru funkcji binarnych. Inaczej mówiąc, funkcje bazowe $a_1, a_2, \ldots, a_n$, mimo że mają argumenty binarne, wcale nie muszą być funkcjami binarnymi, a układy realizujące funkcje bazowe działają (w przypadku ogólnym) w logice wielowartościowej MVL. Ten wniosek zachęcił członków zespołu do badań nad możliwościami zastosowania bramek prądowych w układach i systemach cyfrowych działających w logice MVL, i w szczególności w arytmetykach N-wartościowych, *modulo* N i resztowych.

Zaletami stosowania w cyfrowych systemach jednoukładowych logiki wielowartościowej z podstawą $N > 2$ są znaczne zmniejszenie liczby połączeń wewnętrznych w systemie oraz skrócenie czasu wykonywania operacji w szeregowych jednostkach operacyjnych, np. w szeregowych sumatorach, blokach mnożenia itd. [19]. Poza tym, układy działające w systemie wielowartościowym z podstawą N zwykle można łatwo przekształcić w odpowiednie układy działające w arytmetyce resztowej RNS, np. poprzez usunięcie bloków przeniesienia z układów sumujących. Dzięki głównej właściwości systemu RNS, mianowicie

braku przeniesienia między poszczególnymi cyframi wyniku podczas wykonania podstawowych operacji arytmetycznych, można osiągnąć dodatkowe zmniejszenie złożoności sprzętowej jednostek operacyjnych, jak i czasu wykonania operacji dodawania, odejmowania i mnożenia.

Niestety, mimo intensywnie prowadzonych w ostatnich latach badań w dziedzinie projektowania jednostek przetwarzających działających w w/w systemach liczbowych (o czym świadczy duża liczba publikacji, a nawet konferencji poświęconych tematyce MVL i RNS [43-48]), w praktyce na razie nie udaje się w pełni wykorzystać wszystkich wymienionych zalet logiki wielowartościowej i arytmetyki resztowej. Powiązane to jest głównie z tym, że w praktyce N-wartościowa zmienna logiczna X zwykle jest reprezentowana przez jej odpowiednik przedstawiony w kodzie dwójkowym $(X)_2$. W takim przypadku do przechowania i przekształcenia zmiennej X w systemach MVL i RNS potrzebnych jest $\lceil \log_2 N \rceil$ bitów w poszczególnych blokach jednostek operacyjnych (sumatorach, rejestrach i in.), a do przekazania takiej zmiennej między blokami potrzebna jest magistrala o szerokości $\lceil \log_2 N \rceil$ bitów (gdzie $\lceil p \rceil$ oznacza najmniejszą liczbę całkowitą, równą lub większą od p). Inny sposób konstruowania układów działających w systemach MVL i RNS oparty jest na wykorzystaniu nowych, niestandardowych typów bramek, np. wielowartościowych inwerterów, bramek typu MIN, MAX, bramek-multiplekserów i in. [15, 38, 39]. Jednak złożoność takich bramek pod względem liczby wykorzystanych tranzystorów szybko rośnie przy zwiększeniu podstawy N (zwykle zależność liczby tranzystorów od liczby N jest prawie kwadratowa), co powoduje, że powierzchnia zajmowana przez układ oraz czas propagacji sygnału są podobne jak dla układu, w którym wielowartościowa zmienna logiczna jest reprezentowana przez jej odpowiednik dwójkowy. W związku z tym, jednym z głównych celów autora niniejszej rozprawy, była weryfikacja wyżej wymienionych zalet stosowania w systemach jednoukładowych bloków operacyjnych działających w systemach MVL i RNS zbudowanych z bramek prądowych.

Wykonana w trakcie badań analiza wykazała możliwość zastosowania istniejących typów bramek prądowych: inwertera, anty-inwertera, podwójnego-inwertera oraz anty-podwójnego-inwertera w jednostkach operacyjnych działających w logikach wielowartościowych MVL z podstawą $N > 2$ i/lub w arytmetyce resztowej RNS (opartej o arytmetykę modulo N). Jednak efektywność zastosowania istniejących bramek w w/w jednostkach okazała się niezbyt wysoka, co się wiąże z tym, że bramki te przeznaczone są do działania w systemie binarnym (N=2): poziomy logiczne na wyjściu

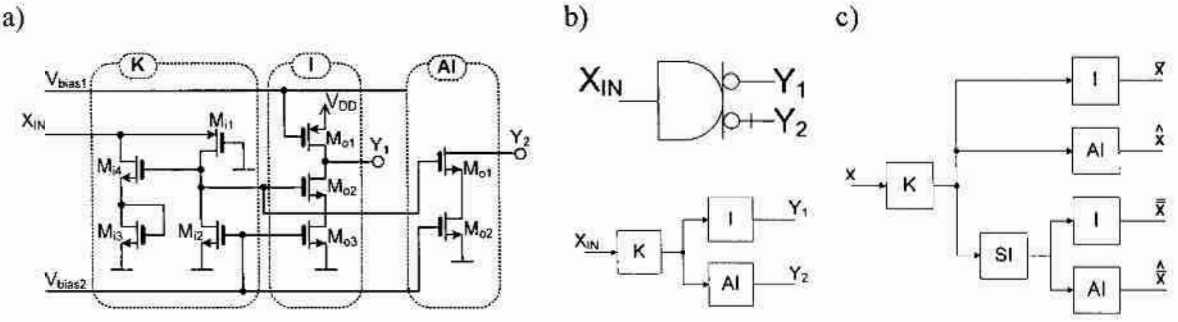
dowolnej z wyżej wymienionych bramek prądowych należą do zbioru {-1, 0} lub {0, 1}. Dlatego autor rozprawy wraz z innymi członkami zespołu badawczego (głownie dr inż. P. Pawłowskim) skupili się nad opracowaniem koncepcji nowych bramek prądowych, przeznaczonych do działania w systemach liczbowych z podstawą $N > 2$, tj. takich bramek inwertera, anty-inwertera, podwójnego-inwertera oraz anty-podwójnego-inwertera, na których wyjściach mogą się pojawiać poziomy logiczne ze zbioru $\{-N,..., 0\}$ lub $\{0,..., N\}$ (w zależności od typu bramki). Projektowanie nowej koncepcji bramki było oparte o opracowaną przez zespół badawczy tablicę prawdy bramek wielowartościowych wszystkich czterech typów. Fragment tej tablicy dla kilku różnych podstaw N jest przedstawiony w tab. 1.7 (dla porównania, w tabeli tej znalazły się również tablice prawdy bramek binarnych). Ponadto, ponieważ wartość podstawy N systemu w jednostkach arytmetycznych RNS i MVL może się zmieniać w trakcie eksploatacji jednostki (w zależności od realizowanych przez nią zadań), podstawowymi kryteriami przy opracowaniu koncepcji i możliwych realizacji tranzystorowych nowych bramek były regularność i modułowość ich budowy wewnętrznej. Te kryteria oraz zasady działania bramek wielowartościowych (przedstawione w tab. 1.7) gwarantują, że schematy klasycznych układów cyfrowych, np. sumatorów, przerzutników, itd. opracowane na poziomie bramek (nie tranzystorów) nie będą się zmieniały (lub będą się zmieniały w sposób regularny) przy zmianie wartości podstawy N.

Tab. 1.7. Tablice prawdy bramek wielowartościowych dla kilku różnych podstaw N

x	N=2				N=3				N=5				N=7			
	$\bar{x}$	$\hat{x}$	$\bar{\bar{x}}$	$\hat{\hat{x}}$	$\bar{x}$	$\hat{x}$	$\bar{\bar{x}}$	$\hat{\hat{x}}$	$\bar{x}$	$\hat{x}$	$\bar{\bar{x}}$	$\hat{\hat{x}}$	$\bar{x}$	$\hat{x}$	$\bar{\bar{x}}$	$\hat{\hat{x}}$
-7	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
-6	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
-5	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
-4	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
-3	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
-2	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
-1	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
0	1	0	0	-1	2	0	0	-2	4	0	0	-4	6	0	0	-6
1	0	-1	1	0	1	-1	1	-1	3	-1	1	-3	5	-1	1	-5
2	0	-1	1	0	0	-2	2	0	2	-2	2	-2	4	-2	2	-4
3	0	-1	1	0	0	-2	2	0	1	-3	3	-1	3	-3	3	-3
4	0	-1	1	0	0	-2	2	0	0	-4	4	0	2	-4	4	-2
5	0	-1	1	0	0	-2	2	0	0	-4	4	0	1	-5	5	-1
6	0	-1	1	0	0	-2	2	0	0	-4	4	0	0	-6	6	0
7	0	-1	1	0	0	-2	2	0	0	-4	4	0	0	-6	6	0

W wyniku badań opracowano nową koncepcję bramki prądowej przeznaczonej do działania w systemach z podstawą $N > 2$, która składa się z bloków wejściowych typu K (komparator) oraz z bloków wyjściowych typu I (inwerter) i/lub AI (anty-inwerter). Przykładowe realizacje tranzystorowe wyżej wymienionych bloków są przedstawione na rys. 1.19a, który przedstawia równocześnie realizację tranzystorową dwuwyjściowej binarnej bramki prądowej z wyjściami typu inwerter (Y_1) i anty-inwerter (Y_2). rys. 1.19b reprezentuje budowę tej bramki na poziomie bloków K, I, AI i SI, natomiast rys. 1.19c reprezentuje schemat 4-wyjściowej bramki binarnej z wyjściami różnych typów z rys. 1.6, też na poziomie bloków K, I, AI i SI.

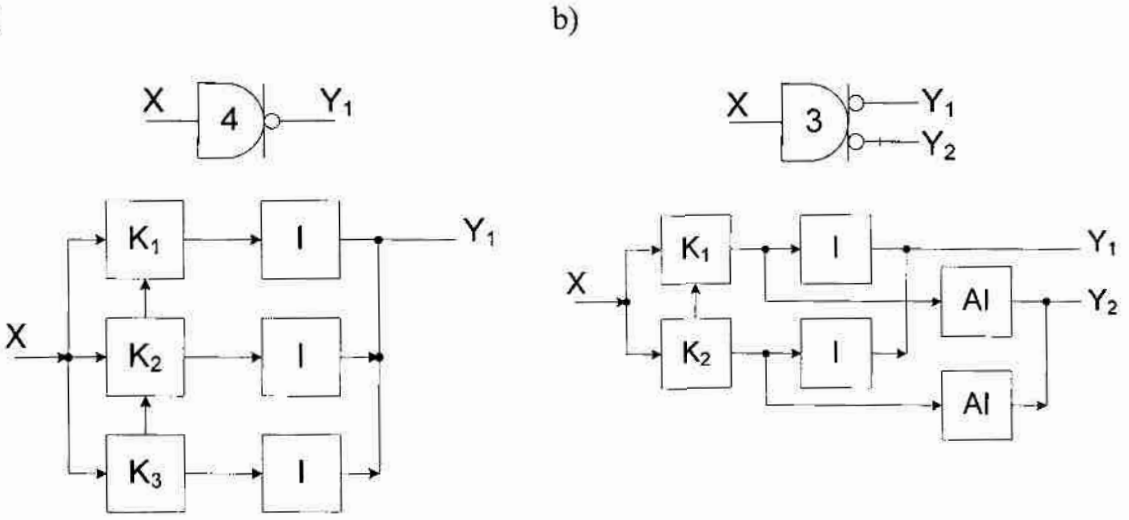
Obwód wejściowy nowej bramki prądowej działającej w systemie MVL z podstawą N składa się z $(N - 1)$ szeregowo połączonych bloków (komparatorów) K. Natomiast realizacja obwodu wyjściowego np. jednowyjściowej bramki inwertera wymaga podłączenia osobnego bloku I do wyjścia każdego komparatora K. Wyjścia wszystkich $(N\text{-}1)$ bloków I są połączone w jeden węzeł, który jest wyjściem bramki. Budowę takiej bramki dla systemu MVL z podstawą $N = 4$ ilustruje rys. 1.20a, natomiast budowę bramki dwuwyjściowej z wyjściami typu inwerter i anty-inwerter reprezentuje rys. 1.20b (dla wartości $N = 3$).



Rys. 1.19. Realizacja tranzystorowa bloków K, I i AI (a), budowa binarnych bramek prądowych na poziomie bloków K, I i AI: 2-wyjściowej (b) i 4-wyjściowej (c)

Badania symulacyjne bloków K, I, AI i SI oraz zbudowanych z nich kilku nowych bramek prądowych przeznaczonych do wykorzystania w systemach liczbowych z podstawą N aż do wartości 19 były przeprowadzone w środowisku SPICE przy użyciu zaawansowanych modeli tranzystorów BSIM3v3 [27]. Wyniki testów wskazują, że zaprojektowane bramki działają prawidłowo. Poza tym, przeprowadzone symulacje wykazały, że nowe bramki cechują się porównywalnymi (z istniejącymi bramkami prądowymi) parametrami statycznymi i dynamicznymi (dotyczy to m.in. liczby tranzystorów, wartości pobieranego prądu i częstotliwości działania bramki). Ponadto, potwierdzona została liniowa zależność wzrostu

czasu opóźnienia bramki wielowartościowej i wartości pobieranego przez nią prądu przy wzroście wartości podstawy N.



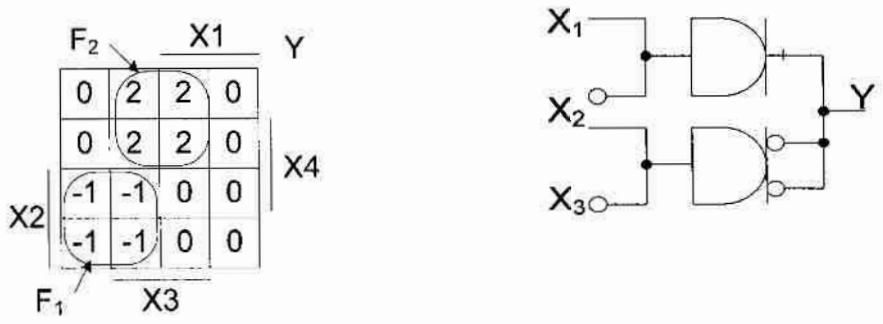
Rys. 1.20. Budowa wielowartościowych bramek prądowych na poziomie bloków K, I i AI: 1-wyjściowej (a) i 2-wyjściowej (b)

1.2.4. Minimalizacja funkcji *N*-wartościowych w algebrze bramek prądowych

W podrozdziale 1.2.2 opisany został *sposób 2* minimalizacji funkcji binarnych w algebrze bramek prądowych, który zakłada, że przy minimalizacji funkcji logicznej Y należy dążyć do otrzymania jej opisu w postaci sumy algebraicznej $Y = a_1 + a_2 + \ldots + a_n$ kilku prostszych funkcji bazowych ($a_1, a_2, \ldots, a_n$), które nie muszą być funkcjami binarnymi.

W związku z tym sposób ten może być stosowany do minimalizacji funkcji N-wartościowych, których argumenty są zmiennymi binarnymi. W takim przypadku można skorzystać z diagramów Veitcha-Karnaugha, które wypełniamy w sposób standardowy (jak w algebrze Boole'a), wpisując poszczególne wartości funkcji w odpowiednie kratki diagramu. Następnie w sposób standardowy łączymy w bloki kratki diagramu zawierające takie same wartości funkcji. Przykładowo dla 4-argumentowej funkcji $Y(X_1,X_2,X_3,X_4)$, której diagram Veitcha przedstawia rys. 1.21a wydzielić można dwa bloki 4-kratkowe, które będą określały funkcje bazowe odpowiednio F_1 i F_2: pierwszy dla wartości funkcji Y=-1 (zakłada się, że jest to funkcja bazowa $-F_1$), drugi dla wartości funkcji $Y = 2$ (zakłada się, że jest to dwukrotność funkcji bazowej F_2). W algebrze Boole'a zapis funkcji logicznych F_1 i F_2 przedstavia wyrażenie (1.31) i (1.32), które po konwersji do algebry prądowej przyjmą postać odpowiednio wyrażeń (1.33) i (1.34).

a) b)



Rys. 1.21. Diagram Veitcha (a) i realizacja na bramkach prądowych (b) trójargumentowej funkcji logicznej *Y*

$$F_1 = \overline{X_1} \cdot X_2 \tag{1.31}$$

$$F_2 = \overline{X_2} \cdot X_3 \tag{1.32}$$

$$F_1 = \overline{X_1 + \overline{X_2}} \tag{1.33}$$

$$F_2 = \overline{X_2 + \overline{X_3}} \tag{1.34}$$

W algebrze prądowej opis całej funkcji będzie wyglądał następująco (1.35):

$$Y = -F_1 + 2 \cdot F_2 = -\left(\overline{X_1 + \overline{X_2}}\right) + 2 \cdot \left(\overline{X_2 + \overline{X_3}}\right) = \left(\overline{X_1 + \overline{X_2}}\right) + 2 \cdot \left(\overline{X_2 + \overline{X_3}}\right) \tag{1.35}$$

a realizacja sprzętowa funkcji *Y* jest przedstawiona na rys. 1.21b.

Należy zaznaczyć, że metoda Veitcha-Karnaugha została w tym przykładzie przedstawiona tylko w celu zapewnienia klarowności omawianego zagadnienia. Opracowany sposób minimalizacji (*sposób 2*) może być oparty o dowolną inną metodę minimalizacji funkcji binarnych, np. Quine'a-McCluskey'a. Ponadto, w tej postaci już nadaje się on do realizacji komputerowej, ponieważ funkcje bazowe w przypadku minimalizacji funkcji *N*-wartościowych nie są wybierane heurystycznie. Jeśli wejściowa funkcja logiczna *Y* przyjmuje *K* różnych wartości ($K<N$), to będzie utworzonych co najmniej (*K*-1) funkcji bazowych (co najmniej po jednej funkcji bazowej dla każdej różnej od zera wartości funkcji *Y*). Wynika to z tego, że tablica prawdy funkcji wejściowej *Y* zostaje podzielona na *K* niezależnych fragmentów w ten sposób, że do *i*-tego fragmentu wpisywane są wszystkie te wiersze z tablicy prawdy funkcji *Y*, gdzie funkcja *Y* przyjmuje tę samą *i*-tą i niezerową wartość $W_i<N$ ($W_i \neq 0$). Następnie osobno dla każdego fragmentu stosuje się wybraną metodę minimalizacji, np. Quine'a-McCluskey'a, a wynik minimalizacji zostaje przekształcony do odpowiedniego wyrażenia algebry prądowej. Ostateczne wyrażenie dla funkcji *Y* formuje się jako sumę algebraiczną otrzymanych funkcji bazowych powielonych odpowiednią liczbę razy

(tj. pomnożone przez wartość W_i). Należy jednak zaznaczyć, że brak możliwości wyboru funkcji bazowych przez projektanta powoduje obniżenie efektywności otrzymywanych rozwiązań, szczególnie dla projektanta z doświadczeniem. Z tego powodu, mimo, że zaletą *sposobu 2* jest możliwość minimalizacji zarówno funkcji binarnych, jak i wielowartościowych, wciąż aktualnym i ważnym zadaniem jest zwiększenie jego efektywności oraz wykorzystanie do zaprojektowania różnego stopnia złożoności układów prądowych przeznaczonych do działania w arytmetykach binarnej, *N*-wartościowej, *modulo N* i RNS. W rozdziale 2 rozprawy podjęta będzie przez autora próba rozwiązania tych zadań.

1.2.5. Narzędzia programowe do weryfikacji cyfrowych układów prądowych na poziomie logicznym

Dodatkowym, ale również ważnym kierunkiem prac zespołu było opracowanie narzędzi programowych umożliwiających zautomatyzowanie procesu projektowania i weryfikacji cyfrowych układów prądowych. Główną przesłanką tego było przyspieszenie procesów tworzenia i weryfikacji układów prądowych jak i również umożliwienie projektowania układów cyfrowych osobom, które nie muszą dokładnie znać algebry i technologii bramek prądowych. Do opisów projektów układów cyfrowych oraz sprawdzenia poprawności ich działania na poziomie logicznym został wykorzystany język opisu sprzętu VHDL, który pozwala na przedstawienie układów cyfrowych w sposób strukturalny i funkcjonalny.

Język VHDL został stworzony do opisu i symulacji cyfrowych układów napięciowych. W celu opisania prądowych układów cyfrowych w zespole stworzono bibliotekę *nstd_logic* odpowiadająca standardowej bibliotece IEEE1164 wykorzystywanej w środowisku programowym (symulatorze VHDL) Active – HDL firmy Aldec [9, 33]. Definiuje ona podstawowe poziomy logiczne oraz tablice rezolucji sygnałów z uwzględnieniem algebry prądowej. Dla potrzeb modelowania stworzonych zostało 11 następujących poziomów logicznych:

'U' – poziom logiczny niezainicjowany;
'E' – poziom logiczny sygnalizujący błąd;
'C' – poziom logiczny minus „3";
'B' – poziom logiczny minus „2";
'A' – poziom logiczny minus „1";
'0' – poziom logiczny „0";

'1' – poziom logiczny „1";

'2' – poziom logiczny „2";

'3' – poziom logiczny „3";

'4' – poziom logiczny „4";

'5' – poziom logiczny „5"

przy czym poziomy od minus „3" do „5" określają wartość i kierunek natężenia prądu przepływającego przez węzeł. Modelując układy cyfrowe przełączane prądem, należy pamiętać o tym, że każda bramka prądowa może posiadać od jednego do ośmiu wyjść realizujących różne operacje bazowe. W bibliotece umieszczono czasy propagacji sygnału przez daną bramkę, który zależy liniowo od ilości wyjść (tab. 1.8).

Ze względu na wykonywaną operację bazową wyjściom bramek prądowych zostały przydzielone następujące symbole (nazwy skrótowe):

R1 – inwerter;

R2 – anty-inwerter;

R3 – podwójny-inwerter;

R4 – anty-podwójny-inwerter.

Tab. 1.8. Opóźnienia binarnych bramek prądowych różnej liczbie wyjść

liczba wyjść bramki	Opóźnienie bramki
1	0.856
2	1.175
3	1.410
4	1.722
5	1.973
6	2.218
7	2.536
8	2.837

Na podstawie tej biblioteki można tworzyć opisy funkcjonalnie (modele) dowolnych bramek, o dowolnej liczbie wyjść (nie większej jak 8). Bramki te umieszczane są w projekcie jako jeden z dodatkowych plików. Po utworzeniu zbioru opisów wszystkich potrzebnych bramek można tworzyć model układu strukturalnie, składający się z odpowiednich bramek i sumatorów (węzłów).

Niestety, opracowane przez zespół rozwiązanie posiadało szereg wad:

- brak w bibliotece najczęściej używanych bramek prądowych;
- niepotrzebne użycie sumatorów jako węzłów;

- nieujednolicony zapis funkcjonalny poszczególnych bramek;
- mała liczba poziomów logicznych (o ile dla prostych modeli nie pojawiały się inne wartości logiczne, o tyle dla większych układów lista była zbyt krótka);
- brak graficznego przedstawienia opracowanych modeli.

Z tego powodu w celu umożliwienia opisu, wizualizacji i weryfikacji złożonych układów prądowych autor w rozprawie podjął próbę przezwyciężenia w/w wad, wciąż opierając się o język VHDL i środowisko Active-HDL. Jednak dodatkowym celem badań autora było opracowanie specjalistycznych narzędzi programowych umożliwiających łatwe konstruowanie prostych układów prądowych w edytorze graficznym oraz przetestowanie ich na poziomie logicznym bez wykorzystania języka VHDL. Opis otrzymanych przez autora wyników w tym zakresie przedstawiono w rozdziale 4 niniejszej rozprawy.

2. Rozwój sposobów minimalizacji funkcji logicznych w algebrze bramek prądowych i projekty prądowych jednostek operacyjnych

W rozdziale 1 przedstawiono dwa sposoby minimalizacji funkcji binarnych przeznaczonych do realizacji na bramkach prądowych. Pierwszy z nich (*sposób 1*) pozwala na stosowanie znanych metod minimalizacji (np. Quine'a-McCluskey'a, Veitcha-Karnaugha, Espresso i in. [4, 14, 24, 25, 32, 34]) i gwarantuje, że dowolna funkcja binarna może być zrealizowana w postaci prądowego układu kombinacyjnego, którego złożoność sprzętowa pod względem liczby wykorzystanych bramek będzie jednakowa ze złożonością sprzętową układu zbudowanego z klasycznych (napięciowych) bramek CMOS. Jednak pod względem liczby tranzystorów układ prądowy zwykle jest bardziej złożony, ponieważ bramka prądowa składa się z większej liczby tranzystorów, niż klasyczna bramka CMOS. Z tego powodu w podrozdziałach 2.1 i 2.2 przedstawiono wyniki badań autora nad modyfikacją *sposobu 1*, ukierunkowaną na zmniejszenie złożoności otrzymywanych układów prądowych właśnie pod względem liczby wykorzystanych tranzystorów. Modyfikacja ta jest oparta m.in. o możliwość bardzo prostej realizacji, w technologii bramek prądowych, operacji arytmetycznego dodawania i odejmowania, i polega na odnalezieniu takich funkcji binarnych, tzw. funkcji wzorcowych, których realizacja w technologii prądowej jest prostsza od realizacji w technologii napięciowej (na klasycznych bramkach CMOS) właśnie pod względem liczby tranzystorów i liczby połączeń w układzie. Idea zmodyfikowanego *sposobu 1* minimalizacji polegałaby na wyszukiwaniu, np. w tablicy prawdy funkcji wejściowej Y, określonych funkcji wzorcowych, wpisywaniu do wyrażenia wynikowego, reprezentującego zminimalizowaną postać funkcji Y, gotowych wzorów opisujących znalezione funkcje wzorcowe, a następnie wyrzuceniu znalezionych fragmentów z tablicy prawdy funkcji wejściowej Y. W związku z tym, w podrozdziałach 2.1.1 i 2.1.2 rozprawy autor przedstawia dwie odnalezione przez siebie funkcje wzorcowe (tzw. funkcje typu T i typu XOR), jak również przedstawia algorytmy pozwalające na odnalezienie n-argumentowych funkcji typu T w tablicy prawdy zadanej N-argumentowej funkcji wejściowej ($N \geq n$) lub w zadanym zbiorze jej implikantów prostych. Ponadto, autor wyprowadza wyrażenia opisujące w algebrze prądowej funkcję boolowską XOR dla różnej liczby argumentów, których realizacja pozwala na uproszczenie odpowiednich układów prądowych realizujących te funkcje. Następnie, w podrozdziale 2.2

przedstawiono zmodyfikowany *sposób 1* oparty o wykorzystanie funkcji wzorcowych typu T i typu XOR, a wykorzystanie nowego sposobu minimalizacji funkcji binarnych przedstawiono w rozdziale 2.3 na przykładach projektowania kilku różnych układów cyfrowych w oparciu o bramki prądowe.

Drugi sposób minimalizacji (*sposób 2*), w przypadku minimalizacji funkcji binarnych, nie nadaje się do realizacji komputerowej, ponieważ wybór funkcji bazowych dokonuje się przez projektanta układu. Z kolei w przypadku minimalizacji funkcji wielowartościowych o argumentach binarnych sposób ten traci swoją efektywność, ponieważ wybór funkcji bazowych jest w dużym stopniu „narzucony z góry", a liczba funkcji bazowych może być duża (przynajmniej nie mniejsza, niż liczba różnych wartości funkcji wejściowej). Z tego powodu w podrozdziale 2.2 autor przedstawia modyfikację tego sposobu przeznaczoną do minimalizacji wielowartościowych funkcji logicznych o argumentach binarnych. Wprowadzona modyfikacja pozwala zmniejszyć liczbę wykorzystywanych funkcji bazowych poprzez zmniejszenie liczby K fragmentów, na które jest dzielona tabela prawdy funkcji wejściowej Y, i opiera się o warunek, pozwalający włączyć do i-tego fragmentu tablicy prawdy funkcji Y te wiersze, dla których przyjmuje ona nie tylko wartość W_i, lecz również inne wartości, np. W_j. W przypadku wykorzystania do minimalizacji funkcji Y diagramów Veitha-Karnaugha modyfikacja ta odpowiada możliwości łączenia w blok nawet tych kratek diagramu, które zawierają różne wartości funkcji Y. Efektywność wykorzystania zmodyfikowanego *sposobu 2* minimalizacji funkcji N-wartościowych przedstawiono w rozdziale 3 rozprawy na przykładach projektowania kilku różnych układów prądowych działających w arytmetykach N-wartościowej, *modulo* N i resztowej.

Należy zaznaczyć, że w celu klarowności przedstawienia materiału w tym rozdziale opracowane algorytmy poszukiwania funkcji wzorcowych i sposoby minimalizacji w większości pokazane są w oparciu o diagramy Veitcha-Karnaugha.

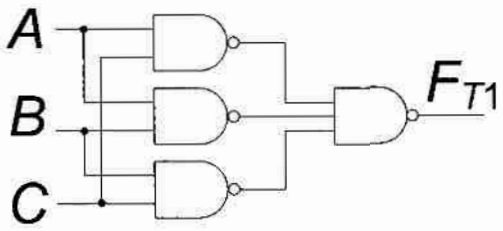
2.1. Minimalizacja funkcji binarnych

2.1.1. Funkcja wzorcowa typu T (TBlok)

Przykład 2.1

W tab. 2.1 przedstawiono tabelę prawdy, na rys. 2.2a diagram Veitcha przykładowej 3-argumentowej funkcji $F_{T1}(A,B,C)$, której opis w algebrze Boole'a przedstawia wyrażenie (2.1), a realizację sprzętową na bramkach NAND przedstawiono na rys. 2.1.

$$F_{T1} = A \cdot B + A \cdot C + B \cdot C = \overline{\overline{A \cdot B} \cdot \overline{A \cdot C} \cdot \overline{B \cdot C}} \tag{2.1}$$



Rys. 2.1. Realizacja sprzętowa funkcji F_{T1} na bramkach napięciowych

Wybór tej właśnie funkcji podyktowany był głównie dwoma powodami (funkcja ta znana jest pod angielską nazwą *majority function)*:

- jest trudna do zminimalizowania w algebrze Boole'a (mimo, że wszystkie wartości logiczne „1" w diagramie Veitcha można połączyć w bloki);
- jest często spotykana, ponieważ jest to funkcja przeniesienia wyjściowego sumatora jednobitowego FA.

Tab. 2.1. Tabela prawdy funkcji F_{T1} i funkcji bazowych ilustrujących sposób jej minimalizacji

A	B	C	F_{T1}	$F_{T1SUM}=A+B+C$	$F_{T1A}=\hat{\overline{A}}+B+C$	$F_{T1B}=A+\hat{\overline{B}}+C$	$F_{T1C}=A+B+\hat{\overline{C}}$	F_{T1SUM2}
0	0	0	0	0	-1	-1	-1	0
0	0	1	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	1	2	1	1	1	1
1	0	0	0	1	0	0	0	0
1	0	1	1	2	1	1	1	1
1	1	0	1	2	1	1	1	1
1	1	1	1	3	2	2	2	1

Na diagramach Veitcha przedstawionych na rys. 2.2a i rys. 2.2b pokazano dwa różne sposoby łączenia jedynek w bloki. Pierwszy z nich powoduje powstanie opisu funkcji F_{T1} bardziej dopasowanego do realizacji w technologii napięciowej, a drugi – do realizacji w technologii prądowej. Po konwersji (w oparciu o wzory 1.23-1.26 i 1.28-1.29) opisy te mogą być przedstawione w postaci wyrażeń (2.2) i (2.3).

$$F_{T1} = \overline{\overline{\overline{A} + \overline{B}} + \overline{\overline{A} + \overline{C}} + \overline{\overline{B} + \overline{C}}} \tag{2.2}$$

$$F_{T1} = \overline{\overline{A} + \overline{C}} + \overline{A + \overline{B} + \overline{C}} + \overline{\overline{A} + \overline{B} + C} \tag{2.3}$$

a)

C \ A			B	F_{T1}
	1	1	1	0
	0	1	0	0

b)

C \ A			B	F_{T1}
	1	1	1	0
	0	1	0	0

c)

C \ A			B	F_{T1SUM}
	2	3	2	1
	1	2	1	0

d)

C \ A			B	F_{T1SUM1}
	1	2	1	0
	0	1	0	-1

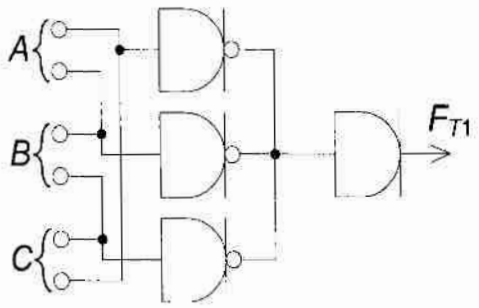
e)

C \ A			B	F_{T1SUM2}
	1	1	1	0
	0	1	0	0

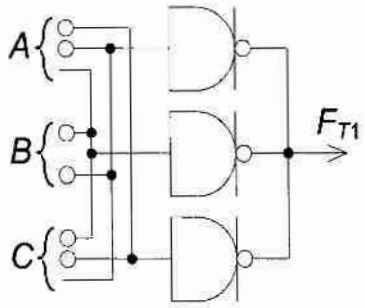
Rys. 2.2. Diagramy Veitcha funkcji F_{T1} (a-b), F_{T1SUM} (c), F_{T1SUM1} (d), F_{T1SUM2} (e)

Realizacja sprzętowa obu wyrażeń przedstawiona jest na rys. 2.3a i rys. 2.3b.

a)



b)



Rys. 2.3. Realizacja sprzętowa funkcji F_{T1} dla wyrażenia (2.2) (a) i (2.3) (b)

Schemat z rys. 2.3a składa się z 4 bramek prądowych o ogólnej liczbie wyjść 10 i wymaga LT=49 tranzystorów do jego realizacji. Schemat z rys. 2.3b składa się z 3 bramek o ogólnej liczbie wyjść 11 i wymaga LT=45 tranzystorów do jego realizacji. Opóźnienie wynosi dwie bramki w przypadku układu z rys. 2.3a i jedną w przypadku układu z rys. 2.3b.

Doświadczenie autora zdobyte przy projektowaniu różnych układów prądowych (szczególnie sumatorów [A3-A7]) doprowadziło do rozważenia innego podejścia do minimalizacji rozpatrywanej funkcji, opartego o wykorzystanie funkcji bazowych (w tym funkcji wielowartościowych). W tab. 2.1 pokazano funkcje F_{T1SUM}, która odpowiada algebraicznej sumie argumentów wejściowych A, B i C. Diagram Veitcha funkcji F_{T1SUM} przedstawiono na rys. 2.2c.

$$F_{T1SUM} = A + B + C \tag{2.4}$$

Zauważono, że:

- funkcja F_{T1SUM} nie jest binarna, jednak jej argumenty są binarne;
- funkcja F_{T1} przybiera wartość „1", wtedy, gdy $F_{T1SUM} > 1$.

Na podstawie tych przesłanek autor wykonał na otrzymanej funkcji F_{T1SUM} dwa następujące działania (dążąc do doprowadzenia F_{T1SUM} do F_{T1}):

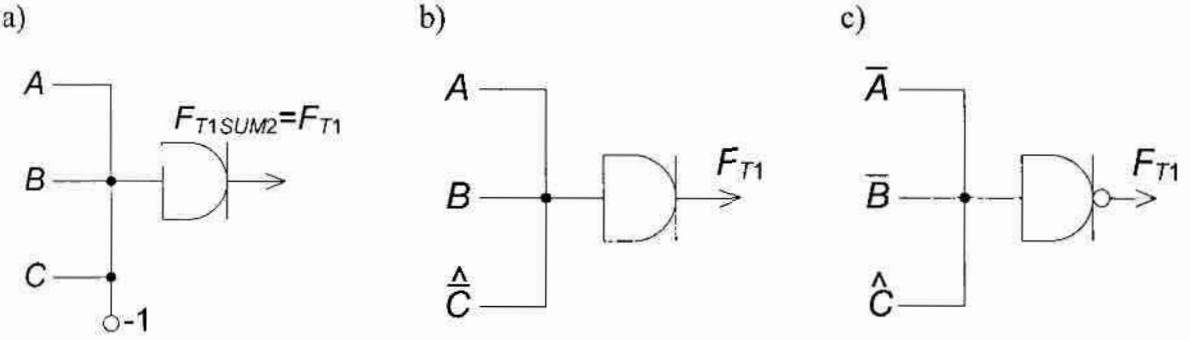
- odejmowania stałej wartości „1" (2.5):

$$F_{T1SUM1} = A + B + C - 1 \tag{2.5}$$

- podwójnej-inwersji na otrzymanej funkcji F_{T1SUM1} (2.6):

$$F_{T1SUM2} = \overline{\overline{A + B + C - 1}} \tag{2.6}$$

Diagramy Veitcha funkcji F_{T1SUM1} i F_{T1SUM2} przedstawiono odpowiednio na rys. 2.2d i rys. 2.2e, przy czym wynika z nich, że funkcje F_{T1SUM2} i F_{T1} są jednakowe. Do realizacji funkcji F_{T1} wystarczy jedna bramka - podwójny inwerter – rys. 2.4a. Układ ten zawiera LT=21 tranzystorów czyli ponad 2 razy mniej niż układ realizujący wyrażenie (2.2).



Rys. 2.4. Realizacja sprzętowa funkcji F_{T1SUM2} (a), F_{T1} (b-c)

Kolejne i już ostateczne uproszczenie polega na usunięciu z wyrażenia (2.6) wartości logicznej „-1" (w oparciu o tożsamość (2.17)). W wyniku wyrażenie (2.6) przekształcono w następujące wyrażenie (2.7):

$$F_{T1SUM2} = F_{T1} = \overline{\overline{A + B + C - 1}} = \overline{\overline{A + B + \hat{\bar{C}}}} \tag{2.7}$$

Realizację sprzętową tego wyrażenia pokazano na rys. 2.4b. Do realizacji funkcji F_{T1} (2.7) wystarczy jedna bramka – podwójny inwerter, a cały układ można zbudować z LT=18 lub nawet $LTmin$=10 tranzystorów.

Należy zaznaczyć, że wartość „-1" w wyrażeniu (2.6) można połączyć z dowolnym argumentem i w efekcie uzyskać inne opisy (2.8) i (2.9) tej samej funkcji F_{T1} o identycznych tablicach prawdy.

$$F_{T1} = \overline{\overline{A+B+C-1}} = \overline{\overline{A-1+B+C}} = \overline{\overline{\hat{\bar{A}}+B+C}} \tag{2.8}$$

$$F_{T1} = \overline{\overline{A+B+C-1}} = \overline{\overline{A+B-1+C}} = \overline{\overline{A+\hat{\bar{B}}+C}} \tag{2.9}$$

Budowa bramek prądowych świadczy, że prostsze pod względem liczby tranzystorów są bramki z wyjściami typu inwerter i anty-inwerter. Dlatego, opisując funkcje przeznaczone do realizacji na bramkach prądowych, należy dążyć do zamiany operacji podwójnej-inwersji na operacje inwersji i anty-inwersji. Przykładowo wyrażenie (2.7) można przekształcić do postaci (2.10). Realizację sprzętową wyrażenia (2.10) przedstawiono na rys. 2.4c.

$$F_{T1} = \overline{\overline{A+B+\hat{\bar{C}}}} = \overline{\hat{A}+\bar{B}+\hat{C}} \tag{2.10}$$

Ostatecznie układ realizujący funkcję F_{T1} składa się z 1 bramki z wyjściem typu inwerter o ogólnej liczbie wyjść 4 i można go zbudować z LT=15 lub $LTmin$=7 tranzystorów. Warto wspomnieć, że odpowiednik takiego układu zbudowany z bramek NAND (jest to najprostsza postać) składa się z 4 bramek o ogólnej liczbie wejść 9 i można go zbudować minimum z 18 tranzystorów (rys. 2.1).

Powyższą funkcję nazwano 3-argumentową funkcją wzorcową typu T (skrótowo TBlokiem), a bardziej szczegółowo TBlokiem(3). Należy zaznaczyć, że istnieje dokładnie 8 różnych wariantów 3-argumentowej funkcji wzorcowej typu T, które różnią się tylko zanegowanymi wartościami (jednej lub kilku) jej argumentów A, B, C. Warianty te są przedstawione na rys. 2.5a-h w postaci diagramów Veitcha oraz odpowiednich wyrażeń opisujących te funkcje (F_{TB1} - F_{TB8}) w algebrze bramek prądowych.

Ponadto badania wykazały, że istnieją 2-, 4- i więcej argumentowe funkcje wzorcowe typu T, których realizacja na bramkach prądowych jest prostsza (pod względem liczby tranzystorów) od analogicznych układów napięciowych. Na przykład, 4-argumentową funkcję typu T reprezentuje następujące wyrażenia boolowskie:

$$TBlok(4) = A \cdot B \cdot C + A \cdot B \cdot D + A \cdot C \cdot D + B \cdot C \cdot D \tag{2.11}$$

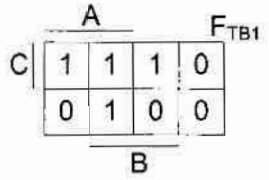
którego realizacja (po uproszczeniu) wymaga stosowania 5 bramek NAND (tj. minimum 32 tranzystorów). W algebrze prądowej funkcja ta jest reprezentowana za pomocą wyrażenia (2.12) i może być ona realizowana w jedno-bramkowym układzie o ogólnej liczbie LT=18 lub $LTmin$=7 tranzystorów.

$$TBlok(4) = \overline{\overline{A} + \overline{B} + \overline{C} + \hat{D}} \tag{2.12}$$

Z tego powodu autor proponuje opis ogólny *n*-argumentowej funkcji wzorcowej typu T (w skrócie TBlok(*n*)) przedstawiony za pomocą wyrażenia (2.13):

$$TBlok(n) = \overline{\overline{X}_1 + \overline{X}_2 + \ldots + \overline{X}_{n-1} + \hat{X}_n} \tag{2.13}$$

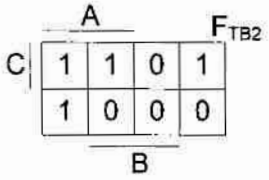
a)



$F_{TB1} = \overline{\overline{A} + \overline{B} + \hat{C}}$

K	1	2	3
F_{TB1}	*A*	*B*	*C*
	0	1	1
	1	0	1
	1	1	0
	1	1	1
S[K]	**3**	**3**	**3**

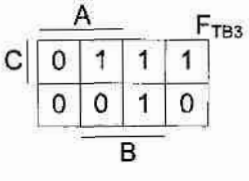
b)



$F_{TB2} = \overline{\overline{A} + B + \hat{C}}$

K	1	2	3
F_{TB2}	*A*	*B*	*C*
	0	0	1
	1	0	0
	1	0	1
	1	1	1
S[K]	**3**	**1**	**3**

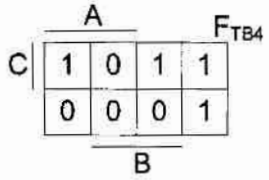
c)



$F_{TB3} = \overline{A + \overline{B} + \hat{C}}$

K	1	2	3
F_{TB3}	*A*	*B*	*C*
	0	0	1
	0	1	0
	0	1	1
	1	1	1
S[K]	**1**	**3**	**3**

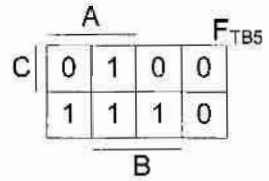
d)



$F_{TB4} = \overline{A + B + \hat{C}}$

K	1	2	3
F_{TB4}	*A*	*B*	*C*
	0	0	0
	0	0	1
	0	1	1
	1	0	1
S[K]	**1**	**1**	**3**

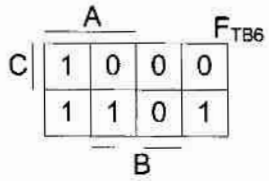
e)



$F_{TB5} = \overline{\overline{A} + B + \hat{\hat{C}}}$

K	1	2	3
F_{TB5}	*A*	*B*	*C*
	0	1	0
	1	0	0
	1	1	0
	1	1	1
S[K]	**3**	**3**	**1**

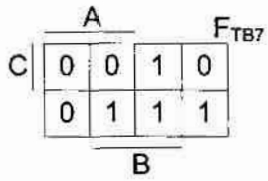
f)



$F_{TB6} = \overline{\overline{A} + B + \hat{\hat{C}}}$

K	1	2	3
F_{TB6}	*A*	*B*	*C*
	0	0	0
	1	0	0
	1	0	1
	1	1	0
S[K]	**3**	**1**	**1**

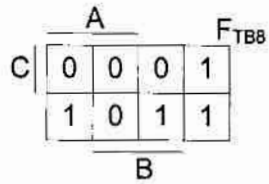
g)



$F_{TB7} = \overline{A + \overline{B} + \hat{\hat{C}}}$

K	1	2	3
F_{TB7}	*A*	*B*	*C*
	0	0	0
	0	1	0
	0	1	1
	1	1	0
S[K]	**1**	**3**	**1**

h)



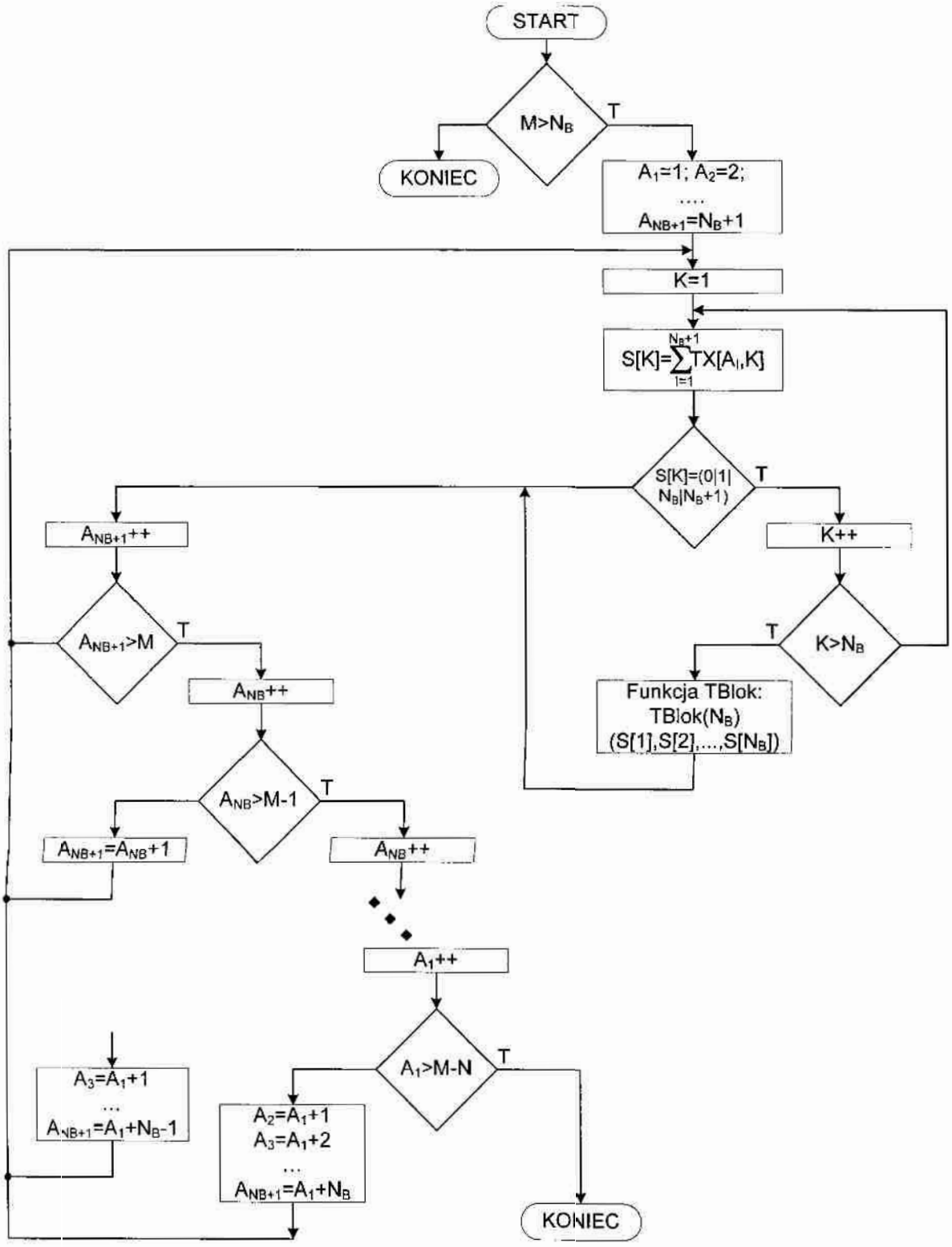
$F_{TB8} = \overline{A + B + \hat{\hat{C}}}$

K	1	2	3
F_{TB8}	*A*	*B*	*C*
	0	0	0
	0	0	1
	0	1	0
	1	0	0
S[K]	**1**	**1**	**1**

Rys. 2.5. Diagramy Veitcha, opisy i skrócone tabele prawdy różnych wariantów funkcji TBlok(3) (a)-(h)

Każdy TBlok(*n*) oprócz wersji podstawowej (2.13) posiada 2^n wariantów, których opis jest podobny do opisu podstawowego, a różni się od niego jedynie brakiem negacji we wzorze podstawowym (dla jednego lub kilku argumentów).

Autor opracował warunek pozwalający stwierdzić, że w tablicy prawdy zadanej funkcji logicznej znajduje się funkcja TBlok(N), oraz oparty o ten warunek algorytm (rys. 2.6) do wyszukiwania funkcji TBlok(N) w tablicy prawdy zadanej N-argumentowej funkcji Y.



Rys. 2.6. Schemat blokowy algorytmu wyszukiwania w funkcji N-argumentowej funkcji TBlok(N)

Podstawą algorytmu jest tablica TX o rozmiarze MxN, która reprezentuje te wiersze tablicy prawdy funkcji Y, w których Y=1 (bardziej szczegółowo M oznacza liczbę implikantów pierwotnych funkcji Y). TBlok(N) istnieje wtedy, gdy suma wartości $S[K]$ we wszystkich K kolumnach (N+1) wierszy (A_1, A_2, ..., A_{N+1},) tablicy TX daje wartość 1 lub N. Argumenty X_K funkcji Y, które w ostatecznym opisie znalezionego TBloku zostaną zanegowane, a które nie, uzależnione są od wartości $S[K]$ otrzymanych przy sumowaniu poszczególnych kolumn (K=1,...,N): jeżeli obliczona wartość $S[K]$=N, to we wzorze należy zanegować argument X_K (odpowiadający tej kolumnie), natomiast jeśli $S[K]$=1, to argument X_K pozostaje bez negacji.

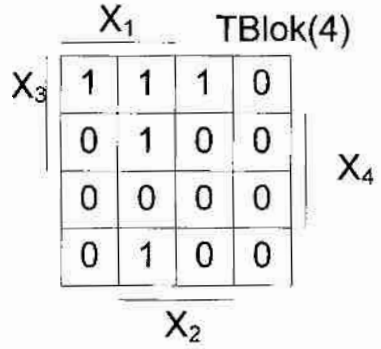
W algorytmie tym na początku sprawdzane jest, czy liczba wierszy M w tablicy TX (liczba implikantów pierwotnych) jest większa lub równa od liczby N_B = (N+1), gdyż TBlok(N_B) wymaga istnienia co najmniej (N+1) jedynek w tablicy prawdy funkcji wejściowej. Jeżeli tak, to zmienne A_1, A_2, ..., A_{NB+1} przyjmują kolejne wartości poczynając od 1 do N_B+1 i określają wiersze, które w danym kroku są sprawdzane (pod względem, czy tworzą one TBlok(N)). Sprawdzanie to polega na obliczaniu sumy $S[K]$ (rys. 2.7) w N_B+1 wierszach tablicy TX (K=1,...,N).

Jeżeli którakolwiek z obliczonych sum jest różna od 1 lub N, to dane wiersze nie tworzą funkcji TBlok(N). W takim przypadku należy zwiększyć o 1 wartość zmiennej A_{NB+1}, umieszczając w ten sposób w tablicy TX kolejny zestaw wierszy do sprawdzenia istnienia funkcji TBlok(N). W przypadku, gdy dla każdej z kolumn suma $S[K]$ daje wartość 1 lub N, numery wierszy, które określa tablica A tworzą TBlok(N), a odpowiednia suma $S[K]$ w kolumnie K jest podstawą do stworzenia opisu funkcji TBlok(N) tak, jak to było opisane powyżej. Złożoność obliczeniowa algorytmu jest rzędu $O(C_M^N)$ operacji porównania.

Na rys. 2.7 przedstawiono diagramy Veitcha dla przykładowych funkcji TBlok(4), TBlok(5) i TBlok(6) wraz z fragmentami tablic prawdy (tablicami TX) i wartościami $S[K]$, ilustrującymi zachowanie opracowanego algorytmu. Zamieszczono tu również opisy odnalezionych funkcji TBlok w algebrze bramek prądowych.

Należy zaznaczyć, że opisany powyżej algorytm po lekkiej modyfikacji pozwala również na odnalezienie n-argumentowych funkcji wzorcowych typu T w tabeli prawdy funkcji N-argumentowych, gdzie N>n. Jedną z takich sytuacji przedstawiono w kolejnym przykładzie.

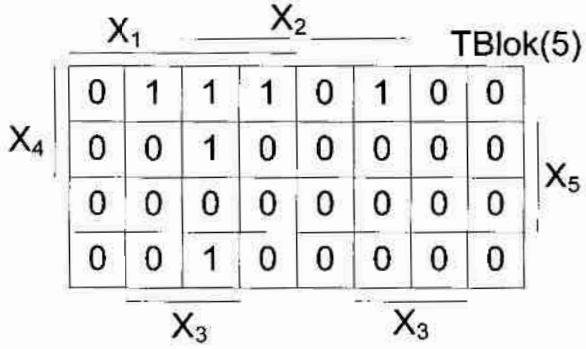
a)



$$TBlok(4) = \overline{\overline{X}_1 + \overline{X}_2 + \overline{X}_3 + \hat{\overline{X}}_4}$$

K	1	2	3	4
TBlok(4)	X1	X2	X3	X4
	0	1	1	0
	1	0	1	0
	1	1	0	0
	1	1	1	0
	1	1	1	1
S[K]	4	4	4	1

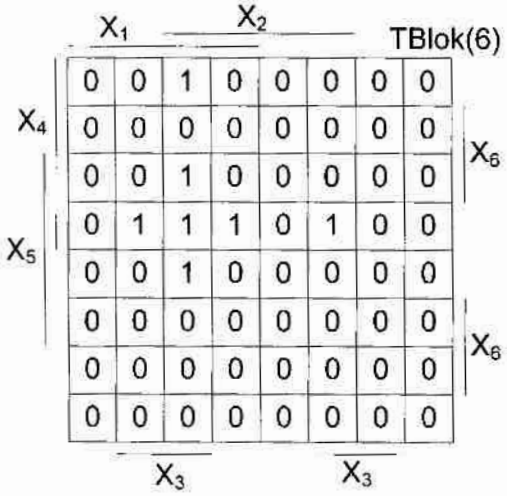
b)



$$TBlok(5) = \overline{\overline{X}_1 + \overline{X}_2 + \overline{X}_3 + \overline{X}_4 + \hat{\overline{X}}_5}$$

K	1	2	3	4	5
TBlok(5)	X1	X2	X3	X4	X5
	0	1	1	1	0
	1	0	1	1	0
	1	1	0	1	0
	1	1	1	0	0
	1	1	1	1	0
	1	1	1	1	1
S[K]	5	5	5	5	1

c)



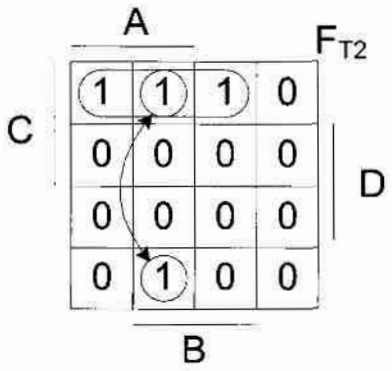
K	1	2	3	4	5	6
TBlok(6)	X1	X2	X3	X4	X5	X6
	0	1	1	1	1	0
	1	0	1	1	1	0
	1	1	0	1	1	0
	1	1	1	0	1	0
	1	1	1	1	0	0
	1	1	1	1	1	0
	1	1	1	1	1	1
S[K]	6	6	6	6	6	1

$$TBlok(6) = \overline{\overline{X}_1 + \overline{X}_2 + \overline{X}_3 + \overline{X}_4 + \overline{X}_5 + \hat{\overline{X}}_6}$$

Rys. 2.7. Diagramy Veitcha i opisy przykładowych funkcji TBlok(n) dla n=4 (a), n=5 (b) i n=6 (c)

Przykład 2.2

Poniżej zaprezentowano przykład 4-argumentowej funkcji $F_{T2}(A,B,C,D)$, która zawiera TBlok(3). Tabela prawdy funkcji F_{T2} przedstawiona jest w tab. 2.2, a jej diagram Veitcha - na rys. 2.8.



Rys. 2.8. Diagram Veitcha funkcji F_{T2}

Opis funkcji F_{T2} otrzymany w oparciu o bloki jedynek przedstawione na rys. 2.8 reprezentuje wyrażenie (2.14):

$$F_{T2} = \overline{\overline{A} + \overline{B} + D} + \overline{\overline{A} + \overline{C} + D} + \overline{\overline{B} + \overline{C} + D} \tag{2.14}$$

które może być uproszczone do postaci wyrażenia (2.15):

$$F_{T2} = \overline{\overline{\overline{A} + \overline{B}} + \overline{\overline{A} + \overline{C}} + \overline{\overline{B} + \overline{C}} + D} = \overline{F_{T2X} + D} \tag{2.15}$$

Tab. 2.2. Tabela prawdy funkcji F_{T2}

A	B	C	D	F_{T2}
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

W wyrażeniu (2.15) wyłoniono funkcję F_{T2X} (2.16), której tabela prawdy przedstawiona jest w tab. 2.3.

Tab. 2.3. Tabela prawdy funkcji F_{T2X}

A	B	C	F_{T2X}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$$F_{T2X} = \overline{\overline{A} + \overline{B} + \overline{A} + \overline{C} + \overline{B} + \overline{C}} \tag{2.16}$$

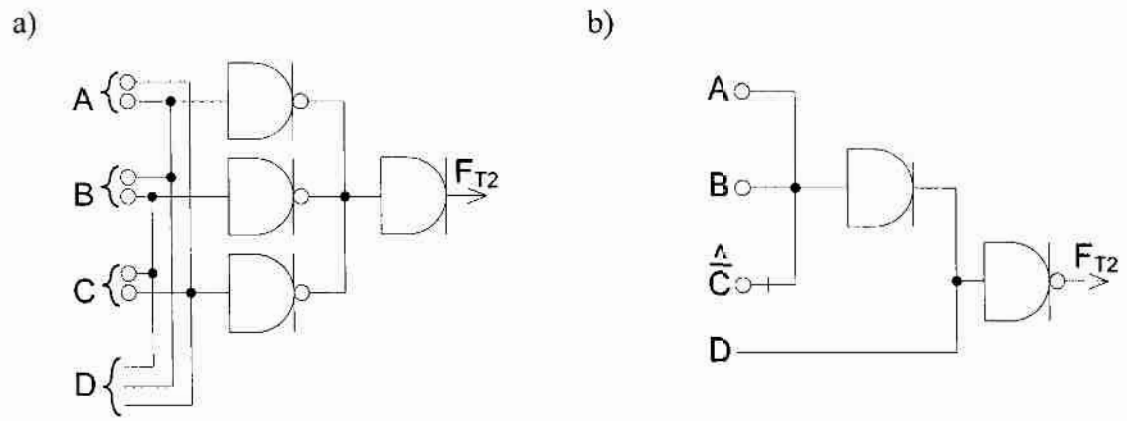
Porównując tabelę prawdy funkcji F_{T1} z *przykładu 2.1* i funkcji F_{T2X} zauważono związek zachodzący pomiędzy nimi (2.17) (funkcja F_{T2X} jest negacją funkcji F_{T1}):

$$F_{T2X} = \overline{F_{T1}} \tag{2.17}$$

Zatem opis funkcji F_{T2} można przedstawić za pomocą wyrażenia (2.18):

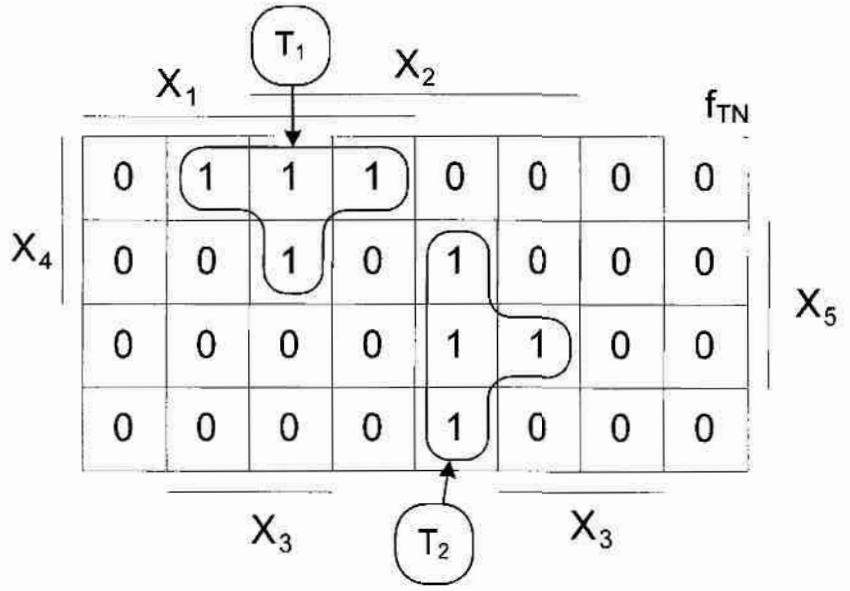
$$F_{T2} = \overline{\overline{F_{T1}} + D} = \overline{\overline{\hat{A} + \overline{B} + \overline{C}} + D} \tag{2.18}$$

Do realizacji sprzętowej funkcji F_{T2} za pomocą wyrażenia (2.14) potrzeba 4 bramek prądowych – trzy bramki z wyjściem typu inwerter i jedna z wyjściem typu podwójny inwerter o ogólnej liczbie wyjść 13 i do jego budowy potrzeba LT=58 tranzystorów (rys. 2.9a). Realizując wyrażenie (2.18), wykorzystamy tylko 2 bramki (wyjścia typu inwerter i podwójny-inwerter) o ogólnej liczbie wyjść równej 6, a do budowy tego układu potrzeba LT=28 lub $LTmin$=17 tranzystorów (rys. 2.9b). Warto zauważyć, że do realizacji funkcji F_{T2} za pomocą bramek napięciowych NAND trzeba użyć 4 bramek, a cały układ zawierać będzie 24 tranzystory.



Rys. 2.9. Realizacja sprzętowa funkcji F_{T2} w oparciu o wyrażenia (2.14) (a) i (2.18) (b)

Z *przykładu 2.2* wynika, że binarna funkcja N-argumentowa $Y=f(X_1, X_2,..., X_N)$ może zawierać funkcję wzorcową (jedną lub więcej) TBlok(n) o dowolnych rozmiarach n, gdzie $N \geq n > 1$. W przypadku $N > n$ dokładnie n argumentów X_i funkcji Y formuje TBlok(n), przy czym są to te argumenty X_i, dla których zmienna $S(K)$ przyjmuje wartość $S(K)=1$ lub $S(K)=n$. Pozostałe (N-n) argumentów wskazują miejsce w tabeli prawdy lub diagramie Veitcha funkcji Y, gdzie ten TBlok(n) się znajduje. Dla tych argumentów zmienna $S(K)$ przyjmuje wartość $S(K)=0$ lub $S(K)=n+1$. Na rys. 2.10 jako przykład przedstawiono 5-argumentową funkcję $f_{TN}(X_1, X_2, X_3, X_4, X_5)$.



K	1	2	3	4	5
T_1	X_1	X_2	X_3	X_4	X_5
	1	0	1	1	0
	1	1	0	1	0
	1	1	1	1	0
	1	1	1	1	1
$S(K)$	4	3	3	4	1

$$T_1 = \overline{X}_2 + \overline{X}_3 + \hat{\overline{X}}_5 + \overline{X}_1 + \overline{X}_4$$

K	1	2	3	4	5
T_2	X_1	X_2	X_3	X_4	X_5
	0	1	0	0	0
	0	1	0	0	1
	0	1	0	1	1
	0	1	1	0	1
S(K)	0	4	1	1	3

$$T_2 = X_3 + X_4 + \hat{X}_5 + X_1 + \overline{X}_2$$

Rys. 2.10. Diagram Veitcha i opisy dwóch funkcji TBlok(3) znalezionych w 5-argumentowej funkcji f_{TN}

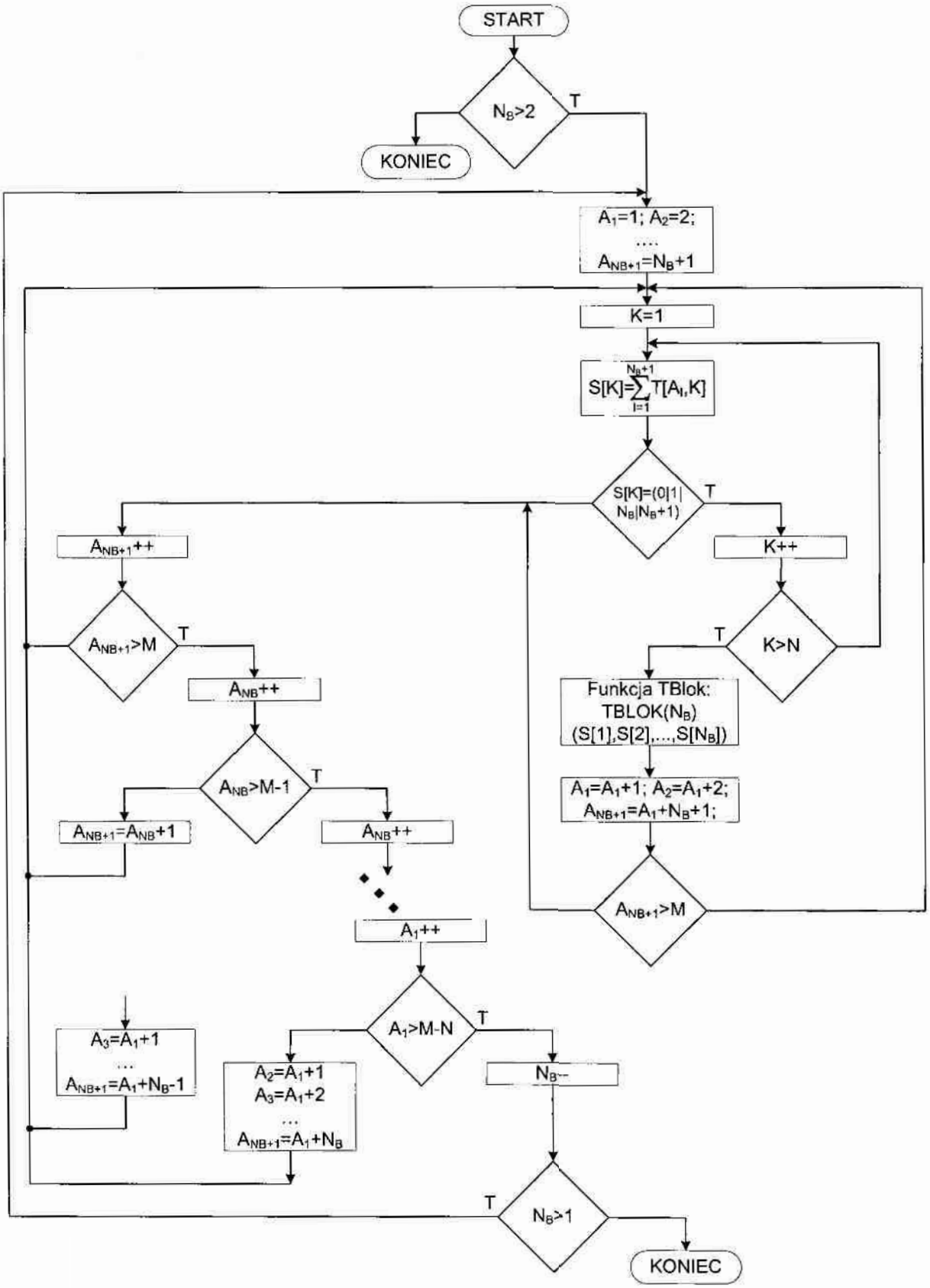
Po sprawdzeniu wszystkich kombinacji 4-wierszowych (wyszukiwanie funkcji wzorcowych TBlok(3)) okazało się, że funkcja f_{TN} zawiera dwa bloki TBlok(3) - T_1 i T_2. Wartości $S(K)$ reprezentujące sumę wartości argumentów w poszczególnych kolumnach obu tablic przyjmują cztery różne wartości zmiennej: 0, 1, n, n+1. W przypadku bloku T_1 wartości $S(2)= S(3)= n= 3$ oraz $S(5)= 1$ odpowiadają argumentom X_2, X_3 i X_5

i formują TBlok(3). Wartości $S(1)= S(4)= 4$ odpowiadają argumentom X_1 i X_4, i wskazują miejsce, gdzie dany TBlok(3) się znajduje. W przypadku bloku T_2 wartości $S(3)= S(4)= 1$ oraz $S(5)= n= 1$ odpowiadają argumentom X_3, X_4 i X_5, dlatego właśnie te argumenty formują opis odpowiedniej funkcji TBlok(3). Wartości $S(1)= 0$ i $S(2)= n+1= 4$ odpowiadają argumentom X_1 i X_2, dlatego te argumenty wskazują miejsce, gdzie dany TBlok(3) się znajduje.

Autor opracował algorytm wyszukiwania funkcji wzorcowych TBlok(n) w funkcji N-argumentowej, gdzie n jest dowolną wartością z zakresu od 2 do N. Podstawą algorytmu jest tablica TX o rozmiarze MxN, gdzie N oznacza liczbę argumentów, a M liczbę jedynek w diagramie Veitcha-Karnaugha danej funkcji. W poszczególnych wierszach umieszczone są wartości argumentów, dla których funkcja przybiera wartość 1. W ten sposób tablica TX przedstawia sobą listę implikantów pierwotnych. Algorytm sprawdza istnienie funkcji wzorcowych TBlok począwszy od największego możliwego TBloku(N_B), gdzie $N_B=MIN(M-1,N)$, a funkcja MIN oznacza wartość mniejszą. Aby wewnątrz wejściowej funkcji N-argumentowej mógł istnieć największy możliwy TBlok(N), tablica TX musi zawierać przynajmniej $(N+1)$ wierszy. Wektor $A(N_B+1)$ na początku zawiera kolejne liczby od 1 do N_B+1, które oznaczają numery wierszy tablicy TX. Te wiersze są w danym kroku algorytmu sprawdzane pod kątem istnienia funkcji wzorcowej TBlok(N_B). Obliczana jest suma wartości $S[K]$ we wskazanych przez wektor A wierszach osobno dla każdej kolumny K. Jeśli uzyskano N_B sum o wartościach 1 lub N_B i $(N-N_B)$ sum o wartościach równych 0 lub (N_B+1), to wiersze wskazane przez wektor A tworzą TBlok(N_B). Po sprawdzeniu wszystkich możliwych kombinacji dla (N_B+1) wierszy, w kolejnym kroku należy sprawdzić istnienie TBloków(N_B-1), postępując podobnie jak dla TBloków(N_B), i każdym kolejnym kroku zmniejszać rozmiar poszukiwanego bloku o 1. Ostatnim wyszukiwanym jest TBlok(2). Schemat blokowy opisanego algorytmu jest przedstawiony na rys. 2.11.

Zaletą opracowanego algorytmu jest to, że nadaje się on do realizacji komputerowej i nie ma ograniczeń na liczbę argumentów N funkcji wejściowych jak również na liczbę n argumentów funkcji wzorcowych. Jedynym, ale poważnym ograniczeniem tego algorytmu jest jego duża złożoność obliczeniowa, która wynosi $O(N \cdot C_M^N)$ operacji porównania. Uniemożliwia ona stosowanie algorytmu dla funkcji o liczbie argumentów $N > 10$ (dla takich funkcji po prostu nie da się uzyskać rozwiązania w rozsądnym czasie).

Z tego powodu autor podjął próbę optymalizacji opracowanego algorytmu wyszukiwania funkcji TBlok(n) o różnej liczbie argumentów n wewnątrz N-argumentowej funkcji binarnej Y $(N \geq n)$.



Rys. 2.11. Schemat blokowy algorytmu odnajdującego funkcje TBlok(n) (dla n=N,...,2) w tablicy prawdy N–argumentowej funkcji logicznej

Optymalizacja była skierowana na zmniejszenie jego złożoności obliczeniowej, głównie dzięki wykorzystaniu w większym stopniu znanych metod minimalizacji. Oznacza to, że zmodyfikowany algorytm już nie operuje na tablicy prawdy funkcji Y lub na jej liście implikantów pierwotnych, operuje natomiast na liście implikantów prostych zakładając, że lista ta została otrzymana za pomocą dowolnej ze znanych metod minimalizacji (np. Quine'a-McCluskey'a, Espresso i in.). Jako przykład tab. 2.4 przedstawia listę implikantów pierwotnych 6-argumentowej ($N = 6$) funkcji $f1(X_1,X_2,X_3,X_4,X_5,X_6)$ realizowanej w pierwszym S-bloku algorytmu kryptograficznego DES [36], a tab.2.5 zawiera listę implikantów prostych tej funkcji otrzymanych za pomocą metody Quine'a-McCluskey'a.

Należy zaznaczyć, że w ostatniej tablicy symbole „*" oznaczają argumenty X_i, które zostały wyeliminowane po wykonaniu operacji sklejenia w trakcie działania metody Quine'a-McCluskey'a, przy czym wszystkie implikanty zostały posortowane pod względem liczby znaków „*", tak że poszczególne kolumny tab. 2.5 zawierają odpowiednio N, (N-1) i (N-2)-argumentowe implikanty.

Tab. 2.4. Lista implikantów pierwotnych funkcji $f1$ S-bloku $S1$ w algorytmie DES

dec	X_1	X_2	X_3	X_4	X_5	X_6
0	0	0	0	0	0	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
9	0	0	1	0	0	1
10	0	0	1	0	1	0
12	0	0	1	1	0	0
13	0	0	1	1	0	1
14	0	0	1	1	1	0
17	0	1	0	0	0	1
18	0	1	0	0	1	0
21	0	1	0	1	0	1

dec	X_1	X_2	X_3	X_4	X_5	X_6
22	0	1	0	1	1	0
23	0	1	0	1	1	1
25	0	1	1	0	0	1
26	0	1	1	0	1	0
31	0	1	1	1	1	1
33	1	0	0	0	0	1
35	1	0	0	0	1	1
36	1	0	0	1	0	0
37	1	0	0	1	0	1
38	1	0	0	1	1	0
40	1	0	1	0	0	0

dec	X_1	X_2	X_3	X_4	X_5	X_6
43	1	0	1	0	1	1
46	1	0	1	1	1	0
48	1	1	0	0	0	0
50	1	1	0	0	1	0
51	1	1	0	0	1	1
52	1	1	0	1	0	0
55	1	1	0	1	1	1
57	1	1	1	0	0	1
58	1	1	1	0	1	0
63	1	1	1	1	1	1

Autor zwrócił uwagę na to, że funkcje wzorcowe TBlok(n) zawsze składają się wyłącznie z (N-1)-argumentowych implikantów niezależnie od wartości n i N. Znane metody minimalizacji funkcji binarnych już nie są w stanie bardziej uprościć lub skrócić zbiór tych implikantów. Natomiast w algebrze prądowej można będzie to zrobić, jeśli w zbiorze (N-1)-argumentowych implikantów odnalezione zostaną funkcje TBlok(n).

Tab. 2.5. Lista implikantów prostych funkcji $f1$ S-bloku $S1$ w algorytmie DES

X_1	X_2	X_3	X_4	X_5	X_6
1	0	1	0	0	0

Nazwa	X_1	X_2	X_3	X_4	X_5	X_6
A_1	0	0	0	*	0	0
A_2	0	0	*	1	0	0
A_3	0	0	1	*	1	0
A_4	0	1	0	*	1	0
A_5	*	0	0	0	1	1
A_6	0	0	1	*	0	1
A_7	0	1	0	*	0	1
A_8	1	0	0	1	*	0
A_9	1	*	0	1	0	0
A_{10}	1	0	0	1	0	*
A_{11}	1	1	0	0	*	0
A_{12}	1	0	0	0	*	1
A_{13}	*	0	1	1	1	0
A_{14}	*	1	1	0	0	1
A_{15}	1	1	0	0	1	*
A_{16}	1	0	*	0	1	1

X_1	X_2	X_3	X_4	X_5	X_6
*	1	*	0	1	0
*	1	*	1	1	1

Z diagramów Veitcha reprezentujących przykładowe funkcje TBlok(n) przedstawionych na rys. 2.7, rys. 2.8 i rys. 2.10 widać, że wszystkie one mają jedną wspólną jedynkę (tj. wszystkie dwu-kratkowe bloki mają wspólną kratkę). Oznacza to, że wśród danej listy zawierającej L prostych (N-1)-argumentowych implikantów funkcji $f1$ trzeba wyszukać właśnie implikanty zawierające wspólny implikant pierwotny(nazwany przez autora *implikantem głównym*). Implikanty takie autor nazywa *implikantami podobnymi*, a warunek podobieństwa jest następujący: dwa implikanty są podobne, jeśli wartość K-tego bitu w obu implikantach jest jednakowa, lub jest zaznaczona przez symbol „*" w jednym z implikantów ($K = 1,..., N$). Na przykład, w drugiej kolumnie tab. 2.5 implikant A_1 jest podobny do implikantu A_2, ale nie jest podobny do implikantu A_3. Liczba n odnalezionych podobnych implikantów określa rozmiar odnalezionej funkcji TBlok(n). Na przykład implikanty A_5, A_{12} i A_{16} są podobne i razem formują TBlok(3).

Opisane wyżej warunek istnienia i idea poszukiwania funkcji TBlok(n) formują następujący dwuetapowy algorytm do ich odnalezienia (*algorytm zmodyfikowany*). Z danej listy zawierającej L prostych (N-1)-argumentowych implikantów funkcji Y wybierany jest pierwszy implikant A_1, który jest rozwijany na 2 implikanty N-argumentowe A_{11} i A_{12} (symbol „*" jest zamieniany w jednym przypadku na „0", a w drugim – na „1"). Oba implikanty mogą

być *implikantami głównymi* dla jakiejś funkcji TBlok. Zatem należy utworzyć dwie tablice TX_{A11} dla pierwszego z nich i TX_{A12} dla drugiego i w tablicy TX_{A11} umieścić implikant A_{12}, a w TX_{A12} implikant A_{11}. Następnie z tabeli z implikantami prostymi wybieramy kolejny implikant A_2, rozwijamy go na dwa implikanty pierwotne A_{21} i A_{22} i jeśli żaden z nich nie jest identyczny z implikantem A_{11} lub A_{12} to dla nich tworzone są kolejne tablice TX_{A21} i/lub TX_{A22} i podobnie jak to wskazano powyżej umieszczane są w nich odpowiednie implikanty (A_{21} w TX_{A22} i A_{21} w TX_{A22}). Jeśli natomiast, którykolwiek z implikantów rozwiniętych wystąpił wcześniej to w podstawieniu uczestniczy tablica TX_{Aij}, która dla niego została wcześniej utworzona. Po wykonaniu L kroków algorytmu sformowane zostaje nie więcej jak L tablic TX, przy czym liczba n_i implikatów w tabeli TX_i ($i = 1,..., L$), określa rozmiar funkcji TBlok(n_i). Na tym etap pierwszy (etap formowania tablic TX z implikantami podobnymi) się kończy i zaczyna się etap drugi – odnalezienia wszystkich różnych funkcji TBlok(n_i) i formowania wyrażeń je opisujących. Odbywa się to w następujący sposób:

Ze wszystkich tablic TX wybierana jest tablica TX_i o maksymalnej liczbie wierszy (implikantów) n_i, ($n_i > 1$), i formowane jest wyrażenie opisujące odnalezioną funkcję TBlok(n_i). Wyrażenie to formowane jest w sposób pokazany na rys. 2.10. Należy zaznaczyć, że opisana struktura wyrażenia wynikowego zawsze jest stała – jest to zanegowana suma, zawierająca wewnątrz jeszcze jedną podwójnie zanegowaną sumę. Upraszcza to realizację komputerową opracowanego sposobu.Następnie ze wszystkich pozostałych tablic TX znowu wybierana jest tablica TX_j o maksymalnej liczbie wierszy (implikantów) n_j, ($n_j > 1$) i formowane jest wyrażenie opisujące odnalezioną funkcję TBlok(n_j), itd.

Dzięki radykalnemu zmniejszeniu długości listy danych wejściowych w porównaniu z podstawowym algorytmem z wartości M (gdzie M - to jest liczba pierwotnych implikantów funkcji wejściowej) do wartości L (gdzie L – to jest liczba (N-1)-argumentowych implikantów prostych funkcji wejściowej) oraz bardziej efektywnej organizacji procesu poszukiwania, udało się zmniejszyć złożoność obliczeniową w algorytmie zmodyfikowanym do wartości $O(2 \cdot L^2)$ operacji podstawienia. Tak niska złożoność zmodyfikowanego algorytmu wyszukiwania funkcji wzorcowych w porównaniu do złożoności znanych metod minimalizacji pozwala na jego stosowanie w kombinacji z dowolną z nich w większości praktycznych zastosowań, bez widocznego wydłużenia czasu działania całego procesu minimalizacji.

Należy zaznaczyć, że opisany powyżej *algorytm zmodyfikowany* może być bezpośrednio zastosowany również do pełnej listy k-argumentowych implikantów ($k < N$)

zadanej funkcji Y, które mogą być otrzymane np. po wykonaniu pierwszego etapu algorytmu Quine'a-McCluskey'a (tj. po wykonaniu wszystkich możliwych operacji sklejenia implikantów pierwotnych) i następnie wyrzuceniu implikantów N-argumentowych. W tym przypadku, dzięki wydłużeniu listy implikantów, czas wykonania algorytmu też się wydłuża, nawet kilkukrotnie. Jednak pojawia się potencjalna możliwość odnalezienia większej liczby funkcji wzorcowych typu T dzięki temu, że niektóre implikanty N-argumentowe, które wchodzą w skład implikantów k-argumentowych ($k < N$-1), mogą być ponownie wykorzystane, teraz jako składowe implikantów (N-1)-argumentowych formujących funkcje wzorcowe. Problemem otwartym w tej sytuacji jest jednak problem odnalezienia „optymalnego pokrycia", tj. formowanie listy implikantów prostych (poprzez wyrzucenie wszystkich zbędnych implikantów) w ten sposób, żeby maksymalnie uprościć opis wynikowy funkcji. Problem ten jest NP-trudny (ang. nondeterministic polynomial) [6] i nienajlepiej nadaje się do realizacji komputerowej. Z tego powodu autor zakłada, że podstawowym przypadkiem zastosowania opracowanego algorytmu (w przypadku jego realizacji komputerowej) jest przypadek, kiedy funkcja wejściowa jest zadana właśnie za pomocą listy implikantów prostych. Natomiast w celu ewentualnego zwiększenia liczby odnalezionych funkcji wzorcowych typu T, każdy (N-1)-argumentowy implikant z tej listy zostaje rozwinięty w dwa implikanty N-argumentowe (symbol „*" w implikancie jest zamieniany w jednym przypadku na „0", a w drugim – na „1"). W ten sposób tworzona jest lista zawierająca część N-argumentowych implikantów pierwotnych funkcji wejściowej Y. Następnie z tej listy wyrzucone zostają wszystkie implikanty powtarzające się, a następnie wykonywane są wszystkie możliwe operacje sklejania i otrzymywana jest nowa lista implikantów (N-1)-argumentowych $B_1, B_2,..., B_P$, której długość P jest zwykle większa od L. Następne działania są identyczne z opisaną powyżej listą implikantów $A_1, A_2,..., A_L$, tj. wyszukiwane są implikanty podobne, które zostają zachowane w tablicach TX_1, $TX_2,..., TX_P$. W każdej i-tej tablicy TX_i, gdzie ($i = 1,..., P$ oraz $n_i > 1$), jest prowadzona operacja obliczenia wartości sum $S[K]$ i formowane jest ostateczne wyrażenie opisujące znaleziony TBlok(n_i). Bardziej szczegółowo różne strategie postępowania w procesie minimalizacji funkcji logicznych z wykorzystaniem funkcji wzorcowych typu T są opisane w podrozdziale 2.1.3. Przykład zastosowania zmodyfikowanego algorytmu do odnalezienia funkcji wzorcowych TBlok(2), TBlok(3) i TBlok(4) w zadanej 6-argumentowej funkcji algorytmu kryptograficznego DES znajduje się w rozdziale 2.3.3 niniejszej rozprawy.

W tab. 2.6 - tab. 2.11 umieszczono podstawowe parametry realizacji funkcji TBlok(n) dla kilku n w różnych technologiach (prądowej i napięciowej). Realizację na bramkach prądowych wykonano przy użyciu *sposobu 1* i *zmodyfikowanego sposobu 1*.

Tab. 2.6. Podstawowe parametry układów prądowych realizujących funkcję TBlok(n) (*sposób 1*)

Liczba argumentów n	3		4		6		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	1		*1*		1		1	
Ogólna liczba wyjść	3		*4*		6		7	
Liczba tranzystorów	15	7	*18*	*7*	24	7	27	7

Tab. 2.7. Podstawowe parametry układów prądowych realizujących funkcję TBlok(n) (*zmodyfikowany sposób 1*)

Liczba argumentów n	3		4		6		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	4		5		7		8	
Ogólna liczba wyjść	9		16		36		49	
Liczba tranzystorów	40	49	62	74	124	142	164	185

Tab. 2.8. Podstawowe parametry układów napięciowych realizujących funkcję TBlok(n)

Liczba argumentów n	3		4		6		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	4		5		7		8	
Ogólna liczba wyjść	9		16		36		49	
Liczba tranzystorów	18	24	32	40	72	84	98	112

Tab. 2.9. Podstawowe parametry układów prądowych realizujących funkcję TBlok(*3*) (*sposób 1*)

Liczba argumentów N	3		4		6		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	1		1		1		1	
Ogólna liczba wyjść	3		4		6		7	
Liczba tranzystorów	15	7	18	7	24	7	27	7

Tab. 2.10. Podstawowe parametry układów prądowych realizujących funkcję TBlok(3) (*zmodyfikowany sposób 1*)

Liczba argumentów N	3		4		6		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	4		5		7		8	
Ogólna liczba wyjść	9		16		36		49	
Liczba tranzystorów	40	49	62	74	124	142	164	185

Tab. 2.11. Podstawowe parametry układów napięciowych realizujących funkcję TBlok(3)

Liczba argumentów N	3		4		6		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	4		5		7		8	
Ogólna liczba wyjść	9		16		36		49	
Liczba tranzystorów	18	24	32	40	72	84	98	112

2.1.2. Funkcja wzorcowa typu XOR (XBlok)

Drugą funkcją, która została wybrana przez autora rozprawy jako funkcja wzorcowa, jest funkcja logiczna XOR (ang. eXclusive OR). Wybór tej właśnie funkcji podyktowany był kilkoma powodami:

- jest często spotykana, ponieważ jest wykorzystywana jako funkcja sumy w sumatorach jednobitowych;
- w przypadku wykorzystania klasycznych metod minimalizacji jest ona trudna do zminimalizowania, ponieważ nie da się wykonać żadnej operacji sklejenia implikantów pierwotnych (w przypadku wykorzystania diagramów Veitcha-Karnaugha jedynek w diagramie nie da się połączyć w bloki). Oznacza to, że podczas realizacji sprzętowej dla każdej dwuargumentowej funkcji XOR trzeba będzie wykorzystać trzy 2-wejściowe bramki NAND;
- jeśli stosowana metoda minimalizacji pozwala na odnalezienie, w tablicy prawdy lub wśród implikantów zadanej funkcji wejściowej, funkcji typu XOR dla dowolnej liczby argumentów [40], to „z góry" zakłada się (narzuca się) wykorzystanie w układzie bramek typu XOR, które są bardziej złożone w porównaniu do bramek NAND i NOR pod względem liczby tranzystorów (nawet trzykrotnie);
- w technologii prądowej realizacja tej funkcji wymaga stosowania minimum dwóch, a niekiedy trzech bramek (jeśli był wykorzystany *sposób 1* minimalizacji).

Z tego powodu poniżej autor zastosował *sposób 2* minimalizacji funkcji logicznych w algebrze prądowej w celu uproszczenia wyrażeń opisujących 3- i więcej argumentowe funkcje XOR, dążąc do uzyskania wyrażenia ogólnego, opisującego n-argumentową funkcję XOR. Pozwoli to zmniejszyć złożoność sprzętową układów prądowych realizujących te funkcje.

Przykład 2.3

W tab. 2.12 przedstawiono tabelę prawdy, a na rys. 2.12a diagram Veitcha funkcji $F_{X1}(A, B, C)=A$ *xor* B *xor* C. Takie położenie jedynek w diagramie powoduje, że żadnej z nich nie można połączyć z inną wykorzystując klasyczne metody minimalizacji (np. Veitcha-Karnaugha lub Quine'a-McCluskeya). Opis funkcji F_{X1} z diagramu na rys. 2.12a otrzymany w oparciu o *sposób 2* minimalizacji przedstawiono poniżej:

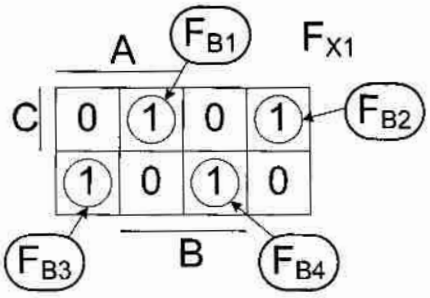
$$F_{X1} = F_{B1} + F_{B2} + F_{B3} + F_{B4} = \overline{\overline{A} + \overline{B} + \overline{C}} + \overline{A + B + \overline{C}} + \overline{\overline{A} + B + C} + \overline{A + \overline{B} + C} \qquad (2.19)$$

Realizację sprzętową wyrażenia (2.19) w technologii bramek prądowych przedstawiono na rys. 2.13a. Układ z rys. 2.13a składa się z 4 bramek o ogólnej liczbie wyjść 16 i zawiera LT=64 tranzystorów.

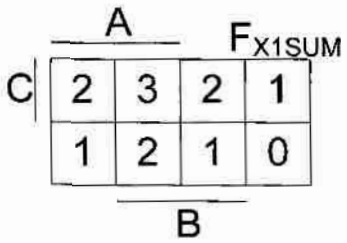
Tab. 2.12. Tabela prawdy funkcji F_{X1}

A	B	C	F_{X1}	$F_{X1SUM}=A+B+C$	F_{X-2}	F'_{X1}
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	1	1	0	1
0	1	1	0	2	-2	0
1	0	0	1	1	0	1
1	0	1	0	2	-2	0
1	1	0	0	2	-2	0
1	1	1	1	3	-2	1

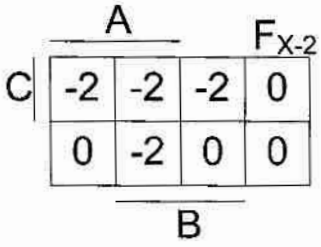
a)



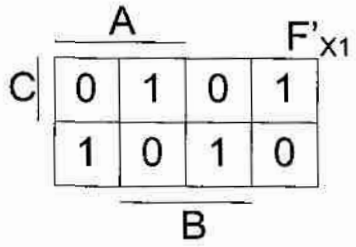
b)



c)



d)



Rys. 2.12. Diagramy Veitcha funkcji F_{X1} (a), F_{X1SUM} (b), F_{X1SUM1} (c), F'_{X1} (d)

W celu zmniejszenia złożoności sprzętowej tego układu określono następujący zbiór funkcji bazowych (znajdujących się w tab. 2.12):

- funkcja F_{X1SUM} (algebraiczna suma wartości logicznych argumentów wejściowych), której diagram Veitcha pokazano na rys. 2.12b (opis tej funkcji przedstawia wyrażenie (2.20)):

$$F_{X1SUM} = A + B + C \tag{2.20}$$

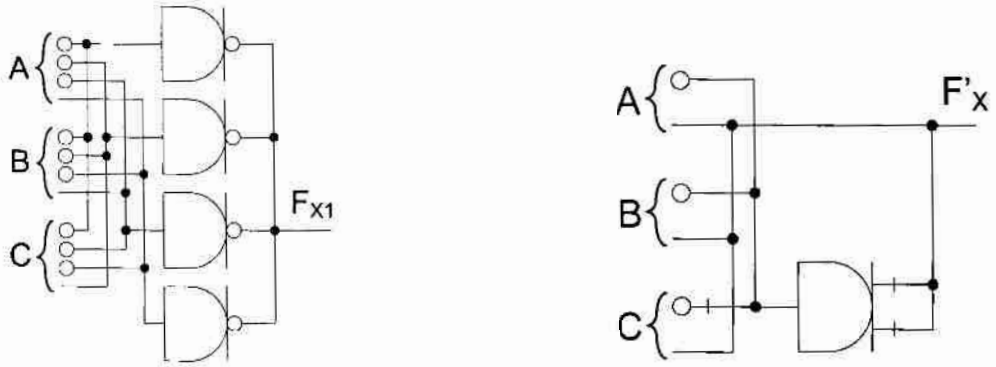
- funkcja F_{X-2}, której diagram Veitcha przedstawiono na rys. 2.12c. Funkcja F_{X-2} jest podobna do funkcji TBlok(3) (F_{T1}), gdzie zamiast wartości logicznych „1" znajdują się wartości logiczne „-2". Opis tej funkcji przedstawia wyrażenie (2.21):

$$F_{X-2} = -2 \cdot (F_{T1}) = -2 \cdot \overline{\overline{A} + \overline{B} + \hat{C}} = 2 \cdot \overline{\overline{A} + \overline{B} + \hat{C}} \tag{2.21}$$

- oraz funkcja F'_X, która jest sumą dwóch poprzednich funkcji, a jej opis przedstawia wyrażenie (2.22):

$$F'_X = F_{X1SUM} + F_{X-2} = A + B + C + 2 \cdot \overline{\overline{A} + \overline{B} + \hat{C}} \tag{2.22}$$

a) b)



Rys. 2.13. Realizacja sprzętowa funkcji F_{X1} w oparciu o wyrażenia (2.19) (a) i (2.22) (b)

Stwierdzono, że tablica prawdy funkcji F'_X jest identyczna z tablicą prawdy funkcji F_X, zatem jest to ta sama funkcja. Realizacja funkcji F'_X w oparciu o bramki prądowe jest prostsza (rys. 2.13b) w porównaniu do jej realizacji na bramkach napięciowych: układ zawiera 1 bramkę i składa się z LT=28 lub nawet $LTmin$=19 tranzystorów. Opis tej funkcji jest również prostszy od opisu otrzymanego *sposobem 2*.

Porównując funkcje F_{X1} i F_{X1SUM} można zauważyć, że w tych wierszach tab. 2.12 gdzie funkcja F_{X1SUM} przybiera wartość nieparzystą, wartość funkcji F_{X1} jest równa „1". Innymi słowy funkcja F_{X1} wskazuje nieparzystość sumy wartości wszystkich argumentów A, B, C w wybranym wierszu tabeli prawdy (na wyjściu otrzymamy „1") lub jej parzystość

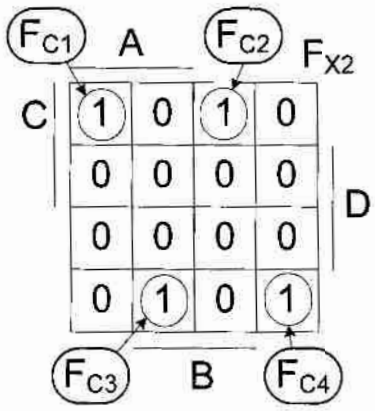
(na wyjściu uzyskujemy „0"). Powyższą funkcję nazwano 3-argumentową funkcją wzorcową typu XOR, a w skrócie XBlokiem(3).

Przykład 2.4

W niniejszym przykładzie przedstawiono 4-argumentową funkcję $F_{X2}(A, B, C, D)$, która zawiera 3-argumentową funkcję wzorcową typu XOR. Minimalizacja zostanie wykonana najpierw *sposobem 1*, a następnie *sposobem 2*. Tabela prawdy rozpatrywanej funkcji F_{X2} przedstawia tab. 2.13, a jej diagram Veitcha przedstawiony jest na rys. 2.14a.

Tab. 2.13. Tabela prawdy funkcji F_{X2}

A	B	C	D	F_{X2}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0



Rys. 2.14. Diagram Veitcha funkcji F_{X2}

Opis funkcji F_{X2} otrzymany w oparciu o diagram Veitcha z rys. 2.14 przedstawia wyrażenie (2.23):

$$F_{X2} = F_{C1} + F_{C2} + F_{C3} + F_{C4} = \overline{\overline{A+B+C+D} + \overline{\overline{A}+\overline{B}+C+D} + \overline{A+\overline{B}+\overline{C}+D} + \overline{\overline{A}+B+\overline{C}+D}} \tag{2.23}$$

Funkcja F_{X2} może posiadać wartości logiczne „1" tylko wtedy, gdy wejście D przyjmuje wartość „0", zatem może ona zostać opisana wyrażeniem (2.24). W wyrażeniu (2.24) wyeksponowano funkcję F_{X2X}, która jest podobna do 3-argumentowej funkcji XOR. Tabela prawdy tej funkcji przedstawiona jest w tab. 2.14 (wyrażenie (2.25)).

$$F_{X2} = \overline{\overline{A+B+C} + \overline{\overline{A}+\overline{B}+C} + \overline{A+\overline{B}+\overline{C}} + \overline{\overline{A}+B+\overline{C}} + D} = \overline{F_{X2X} + D}, \tag{2.24}$$

$$\text{gdzie } F_{X2X} = \overline{\overline{A+B+C} + \overline{\overline{A}+\overline{B}+C} + \overline{A+\overline{B}+\overline{C}} + \overline{\overline{A}+B+\overline{C}}} \tag{2.25}$$

Tab. 2.14. Tabela prawdy funkcji F_{X2X}

A	B	C	F_{X2X}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Porównując tabelę prawdy funkcji F_{X1} z przykładu 2.3 i funkcji F_{X2X} zauważono związek zachodzący pomiędzy nimi (2.26):

$$F_{T2X} = F_{X1} \tag{2.26}$$

Zatem opis funkcji F_{X2} możemy przedstawić za pomocą wyrażenia (2.27):

$$F_{X2} = \overline{F_{X1} + D} = A+B+C+2\cdot\overline{\overline{A}+\overline{B}+\hat{C}}+D \tag{2.27}$$

Do realizacji sprzętowej funkcji F_{X2} za pomocą wyrażenia (2.24) potrzebnych jest 6 bramek prądowych z wyjściem typu inwerter o ogólnej liczbie wyjść 18. Korzystając z wyrażenia (2.27) do realizacji układu wykorzystamy tylko 2 bramki – jeden inwerter i jeden anty-podwójny-inwerter, a ogólna liczba wyjść wynosi 8. Należy zaznaczyć, że tak prostego opisu nie da się uzyskać znanymi sposobami minimalizacji opisanymi w rozdziale 1.

Przykład 2.5

W przykładzie tym na podstawie minimalizacji prostej funkcji 2-argumentowej pokazano różne opisy tej samej funkcji i porównano liczbę tranzystorów potrzebnych do jej realizacji w zależności od typów użytych bramek prądowych. W tab. 2.15 przedstawiono tabelę prawdy funkcji $F_{X(2)}$, która realizuje funkcję XOR dwóch argumentów. Na rys. 2.15a przedstawiono diagram Veitcha funkcji $F_{X(2)}$ a jej opis przedstawia wyrażenie (2.28).

Tab. 2.15. Tabele prawdy funkcji $F_{X(2)}$, $F_{SUMX(2)}$, $F_{X(2)\text{-}2}$, $F'_{X(2)}$

A	B	$F_{X(2)}$	$F_{SUMX(2)}$	$F_{X(2)\text{-}2}$	$F'_{X(2)}$
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	1	0	1
1	1	0	2	-2	0

a)

B \ A	0	1
1	1	0

$F_{X(2)}$

b)

B \ A	2	1
1	1	0

$F_{SUMX(2)}$

c)

B \ A	-2	0
0	0	0

$F_{X(2)\text{-}2}$

Rys. 2.15. Diagramy Veitcha funkcji $F_{X(2)}$ (a), $F_{SUMX(2)}$ (b) $F_{X(2)\text{-}2}$ (c)

$$F_{X(2)} = \overline{\overline{A} + B} + \overline{A + \overline{B}} \tag{2.28}$$

Realizacja funkcji $F_{X(2)}$ w algebrze prądowej wymaga użycia dwóch bramek z wyjściem typu inwerter. Cały układ można zbudować przy użyciu 26 tranzystorów. Zminimalizowano również pozostałe funkcje umieszczone w tab. 2.15:

- $F_{SUMX(2)}$, której diagram Veitcha przedstawiony jest na rys. 2.15b. Opis tej funkcji przedstawia wyrażenie (2.29):

$$F_{SUMX(2)} = A + B \tag{2.29}$$

- $F_{X(2)\text{-}2}$, której diagram Veitcha przedstawiony jest na rys. 2.15c. Opis tej funkcji przedstawia wyrażenie (2.30):

$$F_{X(2)\text{-}2} = -2 \cdot \overline{\overline{A} + \overline{B}} = 2 \cdot \overline{\overline{A} + B} \tag{2.30}$$

- $F'_{X(2)}$, będącą algebraiczną sumą dwóch powyższych funkcji (2.31):

$$F'_{X(2)} = F_{SUMX(2)} + F_{X(2)\text{-}2} = A + B + 2 \cdot \overline{\overline{A} + \overline{B}} = F_{X(2)} \tag{2.31}$$

Tabela prawdy funkcji $F'_{X(2)}$ jest identyczna z tabelą prawdy funkcji $F_{X(2)}$, zatem opisy również są opisami tej samej funkcji logicznej. Funkcja $F'_{X(2)}$ potrzebuje do realizacji w technologii bramek prądowych jedną bramkę (anty-podwójny-inwerter). Do realizacji opisu funkcji (2.31) potrzebne są LT=23 tranzystory czyli o 3 mniej niż w przypadku realizacji wyrażenia (2.28).

Poniżej zaprezentowano kolejny przykład zastosowania *sposobu 2* minimalizacji, tym razem biorąc pod uwagę fakt, że operacje inwersji i anty-inwersji są prostsze w realizacji sprzętowej niż operacje podwójnej-inwersji i anty-podwójnej-inwersji (tab. 2.16).

Tab. 2.16. Tabele prawdy funkcji $F''_{X(2)}$, $F''_{SUMX(2)}$, $F''_{X(2)\text{-}2}$, $F'''_{X(2)}$

A	B	$\overline{A}$	$\overline{B}$	$\hat{B}$	$\overline{A}+\hat{B}$	$\hat{\overline{A}+\hat{B}}$	$F''_{X(2)}$	$F''_{SUMX(2)}$	$F''_{X(2)\text{-}2}$	$F'''_{X(2)}$
0	0	1	1	0	1	-1	0	2	-2	0
0	1	1	0	-1	0	0	1	1	0	1
1	0	0	1	0	0	0	1	1	0	1
1	1	0	0	-1	-1	0	0	0	0	0

Opis funkcji $F''_{SUMX(2)}$ przedstawia wyrażenie (2.32):

$$F''_{SUMX(2)} = \overline{A} + \overline{B} \tag{2.32}$$

Opis funkcji $F''_{X(2)\text{-}2}$ przedstawia wyrażenie (2.33):

$$F''_{X(2)\text{-}2} = -2 \cdot \overline{A + B} = 2 \cdot \hat{\overline{A + B}} \tag{2.33}$$

a)

B \ A	0	1
0	0	1
1	1	0

$F''_{X(2)}$

b)

B \ A	0	1
0	0	1
1	1	2

$F''_{SUMX(2)}$

c)

B \ A	0	1
0	0	0
1	0	-2

$F''_{X(2)\text{-}2}$

Rys. 2.16. Diagramy Veitcha funkcji $F''_{X(2)}$ (a), $F''_{SUMX(2)}$ (b) $F''_{X(2)\text{-}2}$ (c)

Wykorzystując funkcję $\hat{\overline{A}+\hat{B}}$, której tabela prawdy przedstawiona jest w tab. 2.16 funkcję $F''_{X(2)\text{-}2}$ można przedstawić za pomocą wyrażenia (2.34), które będzie prostsze w realizacji sprzętowej.

$$F''_{X(2)\text{-}2} = 2 \cdot \hat{\overline{A}+\hat{B}} \tag{2.34}$$

Sumując obie funkcję otrzymano funkcję $F'''_{X(2)}$, której opis przedstawia wyrażenie (2.35).

$$F'''_{X(2)} = F''_{SUMX(2)} + F''_{X(2)-2} = \overline{A} + \overline{B} + 2 \cdot \overline{\hat{A}} + \hat{B} = F''_{X(2)} = F_{X(2)} \tag{2.35}$$

Realizacja sprzętowa wyrażenia (2.35) wymaga użycia 1 bramki, ogólna liczba wyjść wynosi 7, a liczba tranzystorów LT=17.

Dwu-argumentową funkcję XOR nazwano XBlokiem(2). Najprostszy opis tej funkcji pod względem liczby tranzystorów w układzie prądowym przedstawia wyrażenie (2.35).

Przykład 2.6

W tab. 2.17 przedstawiono tabelę prawdy, a na rys. 2.17a diagram Veitcha-Karnaugha funkcji $F_{X(4)}$. Funkcja ta jest 4-argumentową funkcją XOR (XBlok(4)), a jej opis uzyskany w oparciu o *sposób 1* przedstawia wyrażenie (2.36):

$$\begin{aligned} F_{X(4)} = {} & \overline{A + \overline{B} + \overline{C} + \overline{D}} + \overline{\overline{A} + B + \overline{C} + \overline{D}} + \overline{\overline{A} + \overline{B} + C + \overline{D}} + \overline{\overline{A} + \overline{B} + \overline{C} + D} + \\ & + \overline{\overline{A} + B + C + D} + \overline{A + \overline{B} + C + D} + \overline{A + B + \overline{C} + D} + \overline{A + B + C + \overline{D}} \end{aligned} \tag{2.36}$$

Tab. 2.17. Tabele prawdy funkcji $F_{X(4)}$, $F_{SUMX(4)}$, $F_{X(4)-2}$, $F'_{X(4)-4}$

A	B	C	D	$\overline{\hat{A}} + \overline{\hat{B}} + \overline{\hat{C}} + \hat{D}$	$F_{X(4)}$	$F_{SUMX(4)}$	$F_{X(4)-2}$	$F_{X(4)-4}$	$F'_{X(4)}$
0	0	0	0	-1	0	4	-2	-2	0
0	0	0	1	-1	1	3	-2	0	1
0	0	1	0	-1	1	3	-2	0	1
0	0	1	1	-1	0	2	-2	0	0
0	1	0	0	-1	1	3	-2	0	1
0	1	0	1	-1	0	2	-2	0	0
0	1	1	0	-1	0	2	-2	0	0
0	1	1	1	0	1	1	0	0	1
1	0	0	0	-1	1	3	-2	0	1
1	0	0	1	-1	0	2	-2	0	0
1	0	1	0	-1	0	2	-2	0	0
1	0	1	1	0	1	1	0	0	1
1	1	0	0	-1	0	2	-2	0	0
1	1	0	1	0	1	1	0	0	1
1	1	1	0	0	1	1	0	0	1
1	1	1	1	0	0	0	0	0	0

Realizacja funkcji $F_{X(4)}$ w technologii bramek prądowych wymaga użycia 8 bramek z wyjściem typu inwerter i zawierać będzie $LTmin$=140 tranzystorów.

Postępując analogicznie jak w opisanej w poprzednim przykładzie procedurze minimalizacji 3-argumentowej funkcji XOR, wybrano następujący zbiór funkcji bazowych (przedstawionych w tab. 2.17):

- funkcję $F_{SUMX(4)}$, której diagram Veitcha-Karnaugha przedstawiony jest na rys. 2.17b. Opis tej funkcji przedstawia wyrażenie (2.37):

$$F_{SUMX(4)} = \overline{A} + \overline{B} + \overline{C} + \overline{D} \tag{2.37}$$

- funkcję $F_{X(4)-2}$, której diagram Veitcha-Karnaugha przedstawiony jest na rys. 2.17c. Opis tej funkcji przedstawia wyrażenie (2.38):

$$F_{X(4)-2} = 2 \cdot \overline{A} + \overline{B} + \overline{C} + \overline{D} \tag{2.38}$$

- funkcję $F_{X(4)-4}$, której diagram Veitcha-Karnaugha przedstawiony jest na rys. 2.17d. Opis tej funkcji przedstawia wyrażenie (2.39):

$$F_{X(4)-4} = 2 \cdot \overline{A + B + C + D} \tag{2.39}$$

a)

C \ A,B	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$F_{X(4)}$, D

b)

C \ A,B	00	01	11	10
00	2	1	2	3
01	1	0	1	2
11	2	1	2	3
10	3	2	3	4

$F_{SUMX(4)}$, D

c)

C \ A,B	00	01	11	10
00	-2	0	-2	-2
01	0	0	0	-2
11	-2	0	-2	-2
10	-2	-2	-2	-2

$F_{X(4)-2}$, D

d)

C \ A,B	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	-2

$F_{X(4)-4}$, D

Rys. 2.17. Diagramy Veitcha funkcji $F_{X(4)}$ (a), $F_{SUMX(4)}$ (b), $F_{X(4)-2}$ (c), $F_{X(4)-4}$ (d)

Następnie utworzono funkcję $F'_{X(4)}$, będącą algebraiczną sumą trzech powyższych funkcji (2.40):

$$\begin{aligned} F'_{X(4)} &= F_{SUMX(4)} + F_{X(4)-2} + F_{X(4)-4} = \\ &= \overline{A} + \overline{B} + \overline{C} + \overline{D} + 2 \cdot \overline{A} + \overline{B} + \overline{C} + \hat{D} + 2 \cdot \overline{A + B + C + D} = F_{X(4)} \end{aligned} \tag{2.40}$$

Tabela prawdy funkcji $F'_{X(4)}$ jest identyczna z tabelą prawdy funkcji $F_{X(4)}$, zatem opisy również są opisami tej samej funkcji logicznej. Funkcja $F'_{X(4)}$ potrzebuje do realizacji w technologii bramek prądowych dwóch bramek (anty-inwerter i anty-podwójny-inwerter) i zawierać będzie LT=54 lub $LTmin$=42 tranzystorów, czyli blisko trzy razy mniej niż w przypadku wyrażenia (2.36).

Na rys. 2.18a i rys. 2.18b przedstawione są przykładowe realizacje funkcji wzorcowej XBlok(3) i XBlok(4) na bramkach prądowych, natomiast rys. 2.18c ilustruje przykład realizacji funkcji XBlok(3) znajdującej się w 5-argumentowej funkcji logicznej. Wynika z niego, że realizacja funkcji wzorcowych XBlok(n) znajdujących się w N-argumentowej funkcji logicznej, gdzie $n < N$ wymaga jedynie dołączenia dodatkowych N-n wejść argumentów i jednej bramki z wyjściem typu inwerter (tak jak jest to pokazane na rys. 2.18c). Przytoczone powyżej przykłady pozwoliły autorowi rozprawy na wprowadzenie n-argumentowej funkcji XBlok(n), która w algebrze Boole'a może być opisana za pomocą wyrażenia (2.41):

$$Y = X_1 \, xor \, X_2 \, xor \ldots xor \, X_n \tag{2.41}$$

Tablica prawdy funkcji XBlok(n) zawiera 2^{n-1} jedynek, na których nie da wykonać żadnej operacji sklejenia podczas minimalizacji znanymi metodami (np. Quine'a-McCluskeya lub Veitcha-Karnaugha). Taka funkcja zwykle jest realizowana w układach cyfrowych za pomocą (n-1) dwuwejściowych bramek XOR połączonych szeregowo lub tworzących drzewo zgodnie z wyrażeniem (2.42):

$$\begin{aligned} Y &= ((((X_1 \, xor \, X_2) \, xor \, X_3) \, xor \, X_4) \ldots xor \, X_n) = \\ &= ((X_1 \, xor \, X_2) \, xor \, (X_3 \, xor \, X_4)) \ldots xor \, (X_{n-1} xor \, X_n)) \end{aligned} \tag{2.42}$$

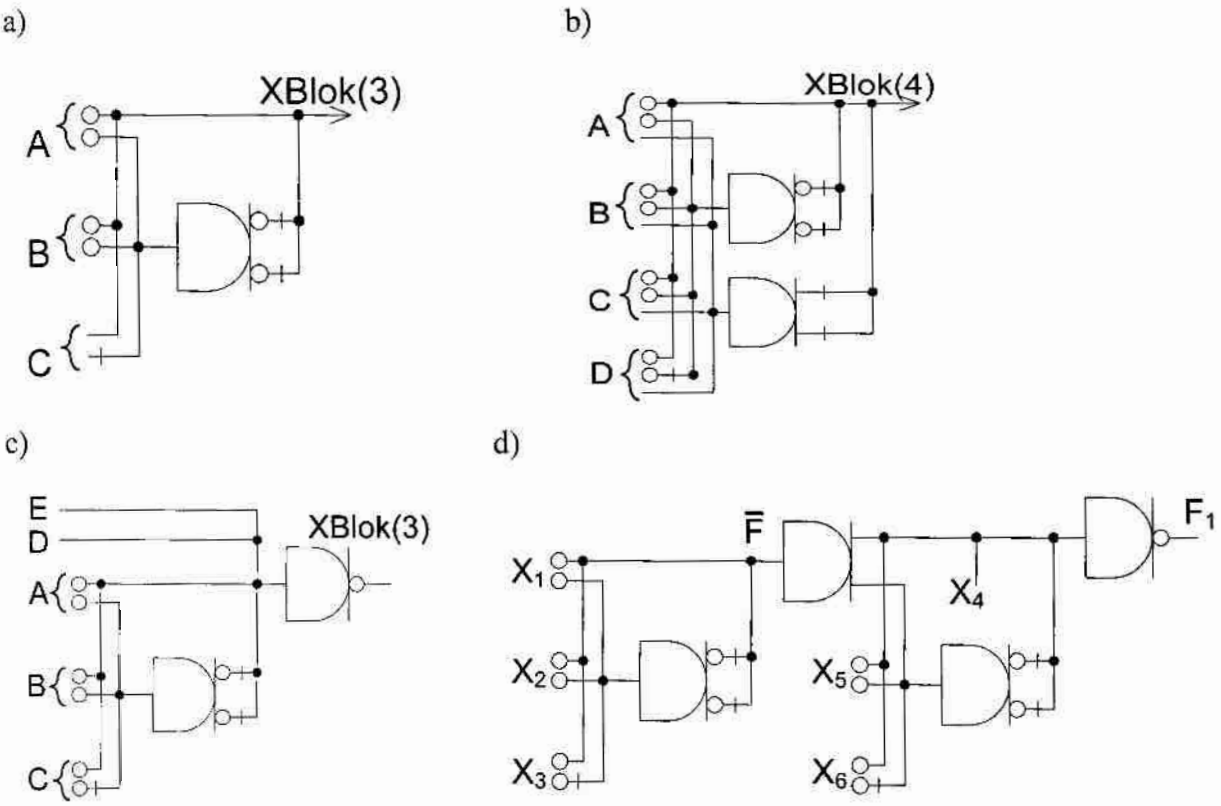
Dzieje się tak dlatego, że n-argumentowe funkcje XOR trudno zrealizować w postaci jednej bramki CMOS (n>2). W technologii bramek prądowych stosuje się prawie ten sam sposób, mianowicie, n-argumentowe funkcje XOR są realizowane w oparciu o układy realizujące funkcję XBlok(3) – też połączone szeregowo lub tworzące drzewo. Jedną z takich realizacji, w której wykorzystano dwie funkcje XBlok(3), przedstawia rys. 2.18d. Na rysunku tym

6-argumentowa funkcja wejściowa $F_1(X_1,X_2,X_3,X_4,X_5,X_6)$, zawiera w sobie m.in. 5-argumentową funkcję XOR (tj. funkcję XBlok(5)), zgodnie z wyrażeniem (2.46):

$$F_1 = (X_1\ xor\ X_2 xor\ X_3 xor\ X_5 xor\ X_6)\cdot \overline{X}_4 = \\ = ((X_1\ xor\ X_2 xor\ X_3) xor\ X_5 xor\ X_6)\cdot \overline{X}_4 \tag{2.43}$$

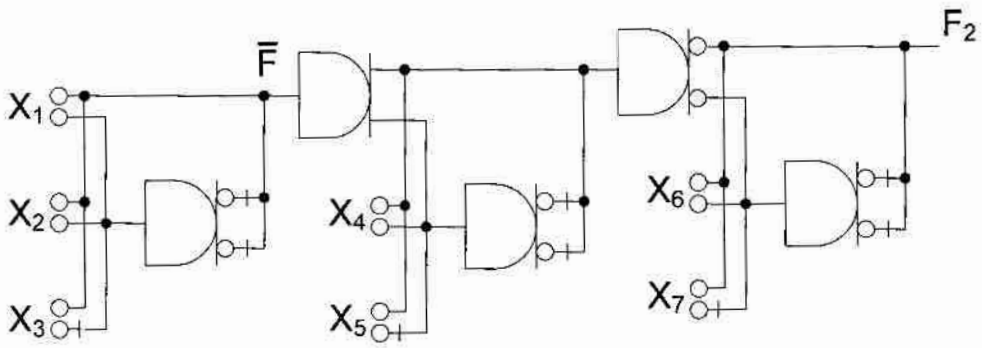
Przykładową realizację funkcji XOR dla przypadku n=7 argumentów (tj. XBlok(7)), przedstawia rys. 2.19.

Z rys. 2.18 i rys. 2.19 widać, że zaletą wykorzystania funkcji XBlok(3) jak „klocków" do realizacji *n*-argumentowych funkcji XOR jest to, że złożoność sprzętowa układu prądowego wzrasta prawie liniowo wraz ze wzrostem liczby *n*, a poza tym, nie zależy od liczby argumentów *N* wejściowej funkcji logicznej (w odróżnieniu od technologii napięciowej, gdzie potrzebna będzie dodatkowa bramka).



Rys. 2.18. Realizacja na bramkach prądowych XBloku(*N*) dla funkcji *N*-argumentowej dla *N*=3 (a), *N*=4 (b), XBloku(3) dla funkcji 5-argumentowej(c), XBloku(5) dla funkcji 6-argumentowej(d),

Np. układ z rys. 2.18d składa się z 4 bramek, a ogólna liczba tranzystorów wynosi *LTmin*=49. Dla porównania liczba tranzystorów w odpowiednim układzie napięciowym (złożonym z klasycznych bramek CMOS) wynosi 52 (zakładając, że jedna dwuwejściowa bramka XOR składa się z 12 tranzystorów [41]).



Rys. 2.19. Przykładowa realizacja 7-argumentowej funkcji XOR

Podsumowując, dowolna *N*-argumentowa funkcja binarna może zawierać XBloki(N_B), gdzie N_B = 2,...,*N*, i jest to najczęściej spotykany w praktyce przypadek. W związku z tym w tab. 2.18 - tab. 2.20 oraz na wykresach przedstawionych na rys. 2.20 i rys. 2.21 autor przedstawia porównanie złożoności sprzętowej układów realizujących *N*-argumentowe funkcje logiczne z udziałem XBlok(3) i zbudowanych w różnych technologiach.

Tab. 2.18. Podstawowe parametry realizacji funkcji XBlok(3)
dla różnej liczby *n* argumentów w technologii prądowej

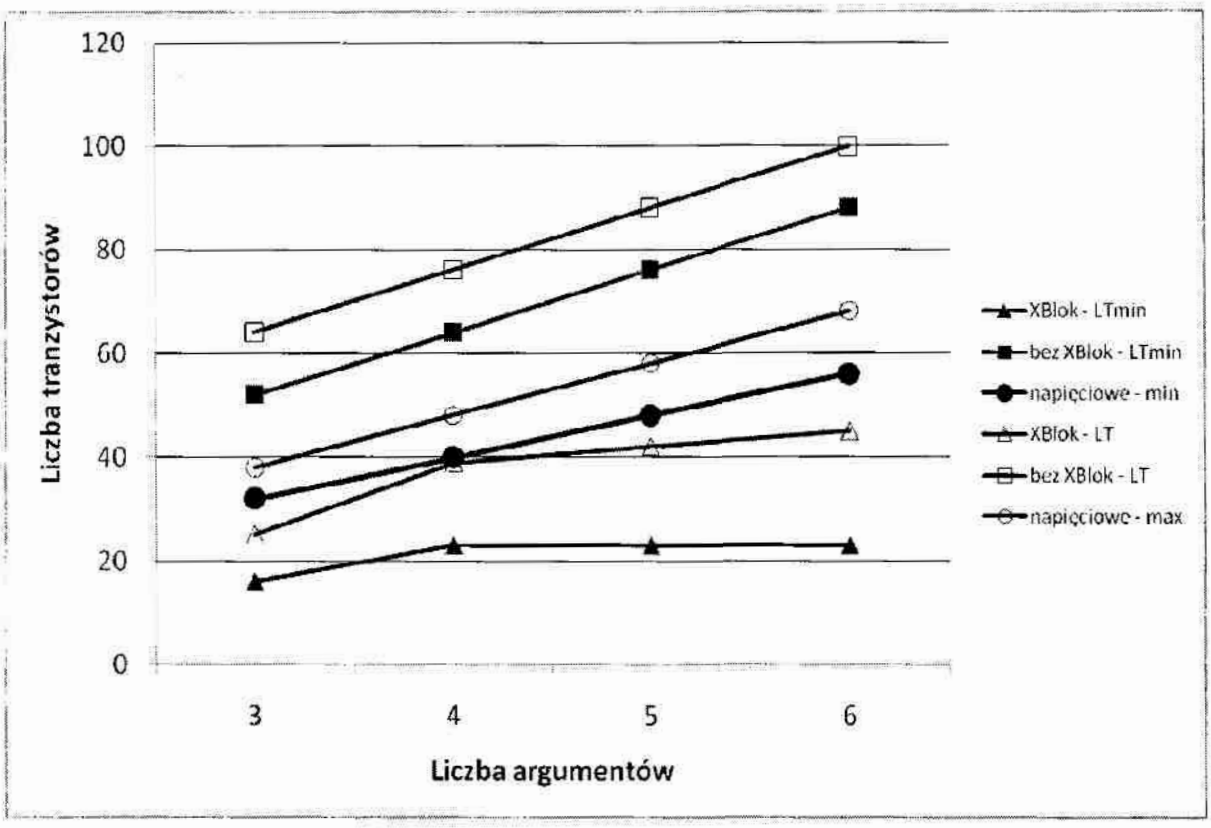
Liczba argumentów *N*	3		4		5		6	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	1		2		2		2	
Ogólna liczba wyjść	8		10		11		12	
Liczba tranzystorów	25	16	39	23	42	23	45	23

Tab. 2.19. Podstawowe parametry realizacji funkcji XBlok(*n*)
dla różnej liczby *n* argumentów w technologii prądowej

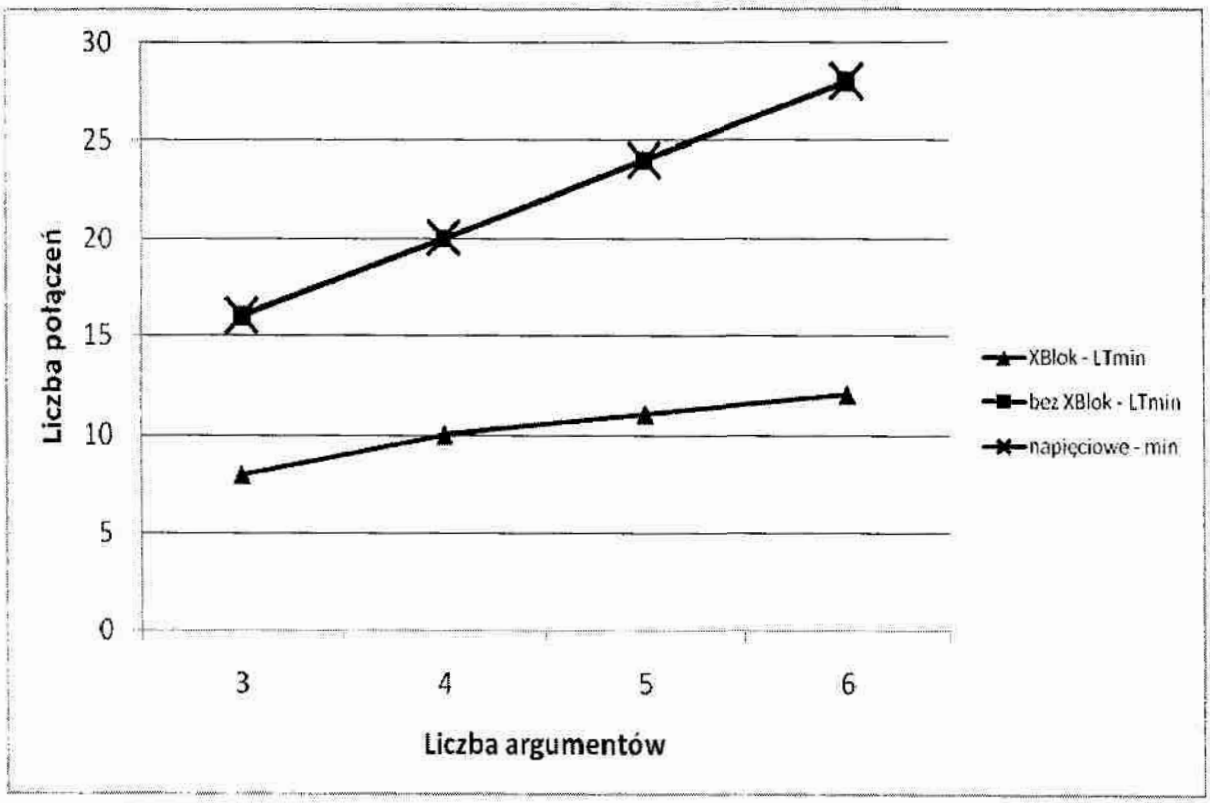
Liczba argumentów *N*	3		4		5		7	
	LT	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*	*LT*	*LTmin*
Liczba bramek	1		3		3		5	
Ogólna liczba wyjść	8		14		16		22	
Liczba tranzystorów	25	16	48	36	54	39	86	65

Tab. 2.20. Podstawowe parametry realizacji funkcji XBlok(3)
dla różnej liczby *n* argumentów w technologii napięciowej

Liczba argumentów *N*	3		4		5		6	
	max	*min*	*max*	*min*	*max*	*min*	*max*	*min*
Liczba bramek	5		5		5		5	
Ogólna liczba wejść	16		20		24		28	
Liczba tranzystorów	38	32	48	40	58	48	68	56



Rys. 2.20. Liczba tranzystorów potrzebna do realizacji
funkcji XBlok(3) dla różnych metod minimalizacji i technologii



Rys. 2.21. Liczba połączeń potrzebna do realizacji
funkcji XBlok(3) dla różnych metod minimalizacji i technologii

Analiza tab. 2.18 - tab. 2.20 świadczy o mniejszej (dla XBloku(3) około 25%) złożoności sprzętowej układów prądowych i napięciowych realizujących funkcje XBlok(n). W związku z tym, obie wprowadzone przez autora funkcje wzorcowe, typu T i typu XOR będą brane pod uwagę w następnym podrozdziale rozprawy przy opracowaniu sposobu minimalizacji binarnych funkcji logicznych w algebrze prądowej.

2.1.3. Sposób minimalizacji funkcji binarnych w algebrze prądowej w oparciu o funkcje wzorcowe

Ze względu na istnienie różnych algorytmów wyszukiwania funkcji wzorcowych typu T - podstawowego i zmodyfikowanego oraz ze względu na istnienie różnych metod minimalizacji funkcji binarnych, które umożliwiają odnalezienie funkcji typu XOR lub nie, nadają się do realizacji komputerowej lub nie, operują na liście implikantów funkcji wejściowej lub na jej tablicy prawdy, w tym rozdziale autor proponuje sposób minimalizacji uwzględniający większość z w/w sytuacji.

Przypadek 1

Jeśli N-argumentowa funkcja wejściowa jest zadana za pomocą listy implikantów prostych, należy najpierw wybrać z tej listy wszystkie implikanty (N-1)-argumentowe $A_1, A_2,..., A_L$. Wyrażenie wynikowe opisujące funkcję wejściową należy przedstawić jako sumę logiczną OR wszystkich pozostałych implikantów. Następnie, korzystając z wyrażeń (1.23) i (1.25) należy przekształcić otrzymane wyrażenie w odpowiednie w algebrze prądowej. Do listy (N-1)-argumentowych implikantów $A_1, A_2,..., A_L$ lub do listy rozwiniętej implikantów (N-1)-argumentowych $B_1, B_2,..., B_P$ (zgodnie z *algorytmem zmodyfikowanym*) należy zastosować *podstawowy* lub *zmodyfikowany algorytm* wyszukiwania funkcji wzorcowych typu T (funkcji TBlok(i), $i = N, (N\text{-}1),..., 2$) opisany w podrozdziale 2.1.1. Jeśli takie funkcje zostały odnalezione, należy do wyrażenia wynikowego dopisać sumę wszystkich wyrażeń opisujących odnalezione funkcje TBlok(i). Pozostałe (N-1)-argumentowe implikanty należy przekształcić w odpowiednie wyrażenia algebry prądowej w oparciu o wzory (1.23) i (1.27) oraz dopisać je do wyrażenia wynikowego. Na tym procedura minimalizacji funkcji wejściowej się kończy. Należy jednak zaznaczyć, że w przypadku, gdy oprócz listy implikantów prostych jako dane wejściowe zadany jest zbiór i-argumentowych funkcji typu XOR (funkcje XBlok(i), $i = N, (N\text{-}1),..., 2$), do wyrażenia wynikowego należy dopisać sumę wszystkich odpowiednio przekształconych (za pomocą wzorów (2.22), (2.40) oraz (2.43)) wyrażeń opisujących odnalezione funkcje XBlok(i).

Przypadek 2

Jeśli funkcja wejściowa jest zadana za pomocą listy implikantów pierwotnych, należy najpierw do tej listy zastosować dowolną znaną metodę minimalizacji przeznaczoną do realizacji komputerowej, np. klasyczną lub rozszerzoną (ang. extended) metodę Quine'a-McCluskey'a, metodę Espresso, itd. w celu uzyskania listy implikantów prostych zadanej funkcji wejściowej (oraz ewentualnie listę znalezionych funkcji XBlok(i), $i = N, (N\text{-}1),..., 2$). W ten sposób rozpatrywany przypadek jest doprowadzony do opisanego powyżej *przypadku 1* i dalej należy postępować tak, jak w *przypadku 1*. Należy zaznaczyć, że jeśli funkcja wejściowa jest zadana za pomocą tabeli prawdy, jej lista implikantów pierwotnych jest otrzymywana w sposób standardowy, poprzez wpisanie na listę tych kombinacji wartości wszystkich N argumentów dla których funkcja przyjmuje wartość „1".

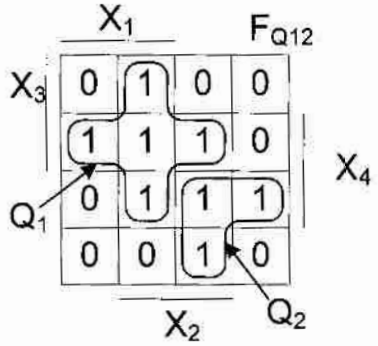
Przypadek 3

Ten przypadek odpowiada „ręcznej" minimalizacji N-argumentowych funkcji logicznych (zwykle $N \leq 6$), np. w oparciu o diagramy Veitcha-Karnaugha. W tym przypadku, po wypełnieniu w sposób standardowy diagramu wartościami funkcji wejściowej, następuje wyszukiwanie kratek zawierających wartości „1" i ewentualnie symbole „*", nadających się do łączenia w tzw. bloki, które w tym przypadku nazywane są przez autora rozprawy blokami standardowymi SBlok(n), gdzie n oznacza liczbę argumentów, tworzących ten blok ($n < N$). Inaczej mówiąc SBlok(n) reprezentuje blok zawierający 2^n kratek diagramu, w których w każdej jest wpisana „1" lub symbol „*". Na przykład blok 2-kratkowy oznaczany jest symbolem SBlok(1), blok 4-kratkowy oznaczany jest symbolem SBlok(2) itd. Kolejnym krokiem jest wyszukiwanie na diagramie funkcji SBlok, TBlok i XBlok różnego rozmiaru n. Funkcje SBlok, TBlok i XBlok należy wyszukiwać zaczynając od tej, która pokrywa jak największa ilość kratek z wartościami „1" lub „*", pamiętając, że XBlok(n) pokrywa 2^{n-1} takich kratek, a TBlok(n) - (n+1) takich kratek diagramu. Przykładowo, dla funkcji 6-argumentowej należy sprawdzić w podanej kolejności występowanie następujących funkcji wzorcowych: SBlok(6), SBlok(5), XBlok(6), SBlok(4), XBlok(5), SBlok(3), XBlok(4), TBlok(6), TBlok(5), TBlok(4), SBlok(2), TBlok (3), XBlok(3), TBlok(2), SBlok(1), XBlok(2), SBlok(0). Po odnalezieniu dowolnego z w/w bloków należy wstawić do wyrażenia wynikowego, reprezentującego zminimalizowaną funkcję, odpowiedni opis odnalezionego bloku i zamienić w diagramie wartości w odpowiednich kratkach z „1" na „*". Proces poszukiwania bloków kończy się, gdy diagram już nie zawiera żadnej kratki z wartością „1". Wynikowe wyrażenie przedstawia sobą sumę wszystkich odnalezionych funkcji SBlok,

TBlok i XBlok (zgodnie ze wzorem (2.27)), ale jeśli dwa lub więcej znalezionych bloków mają wspólne kratki, do sumy tych bloków należy zastosować operację podwójnej–inwersji zgodnie ze wzorem (2.25).

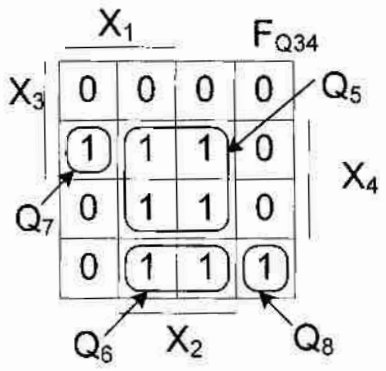
Należy zaznaczyć, że proponowana kolejność poszukiwania funkcji SBlok, TBlok i XBlok nie zawsze gwarantuje otrzymanie najprostszego opisu funkcji wejściowej. Dzieje się tak dlatego, że różne rodzaje bloków o różnych rozmiarach mogą pokrywać jednakową liczbę jednostkowych kratek diagramu: np. SBlok(2) i TBlok(3) pokrywają po 4 kratki jednostkowe każdy. Dlatego w zależności od liczby takich bloków w diagramie oraz od kolejności ich wyszukiwania można otrzymywać różne wyrażenia wynikowe opisujące zminimalizowaną funkcję wejściową. Problem najlepszego pokrycia jest wciąż otwartym, a autor rozprawy proponuje stosowanie algorytmów genetycznych w celu znalezienia takiego pokrycia diagramu, które pozwoli wybrać minimalną ilość funkcji wzorcowych. Niżej umieszczono dwa przykłady ilustrujące opisany problem.

Na rys. 2.22 pokazano przykładowy diagram Veitcha dla 4-argumentowej funkcji F_{Q12}. Korzystając z algorytmu opisanego powyżej (zgodnie z *przypadkiem 3*), znaleziony zostałby najpierw TBlok(4) (Q_1), a następnie TBlok(2) (Q_2), tak jak zaznaczono to na diagramie.

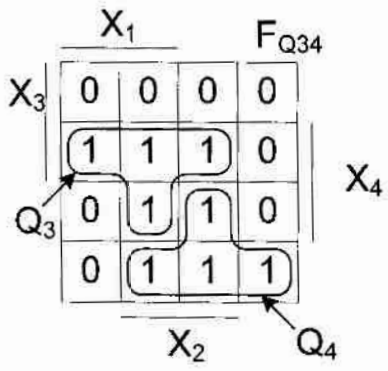


Rys. 2.22. Diagram Veitcha przykładowej funkcji F_{Q12}

a)



b)



Rys. 2.23. Diagram Veitcha przykładowej funkcji F_{Q34}

Na rys. 2.23a przedstawiono diagram Veitcha dla funkcji F_{Q34}. Korzystając z proponowanego sposobu minimalizacji (również *przypadek 3*), najpierw odnaleziona zostałaby funkcja Q_5 (SBlok (2)) a następnie funkcja Q_6 (SBlok(1)) i dwie funkcje SBlok(0) – Q7 i Q8.

Inna kolejność poszukiwania funkcji wzorcowych, np. najpierw TBlok(3), a potem SBlok(2) (rys. 2.23b), pozwala na odnalezienie dwóch funkcji TBlok(3) – Q3 i Q4, które dają znacznie prostsze wyrażenie końcowe.

2.2. Minimalizacja wielowartościowych funkcji logicznych o binarnych argumentach

W podrozdziale 1.2.2 zaprezentowano sposób minimalizacji wielowartościowych funkcji logicznych oparty o diagramy Veitcha-Karnaugha. Sposób ten polegał na łączeniu ze sobą kratek o takich samych wartościach logicznych, znajdujących się w określonych kratkach diagramów Veitcha-Karnaugha. W niniejszym podrozdziale zaprezentowano kilka przykładów, w których wykorzystano ten sposób do znalezienia opisów różnych funkcji. Zaproponowano również modyfikacje tego sposobu, który pozwala otrzymać opisy prostsze, łatwiejsze do realizacji sprzętowej.

Przykład 2.7

W przykładzie tym przedstawiono minimalizację funkcji $F_P = f(A, B, C, D)$, której tabela prawdy przedstawia tab. 2.21.

Zgodnie ze sposobem zaprezentowanym w podrozdziale 1.2.2 stworzono diagram Veitcha tej funkcji (rys. 2.24a), a następnie wybrano 4 funkcje bazowe F_{a1}, F_{a2}, F_{a3}, F_{a4} mające takie same argumenty i spełniające warunek: $F_{a1} \cap F_{a2} \cap F_{a3} \cap F_{a4} = \varnothing$. Sumując je ze sobą otrzymamy funkcję F_P. Wyrażenie (2.44) przedstawia tak otrzymaną funkcję F_P i jest najprostszym możliwym do uzyskania zaprezentowanym sposobem:

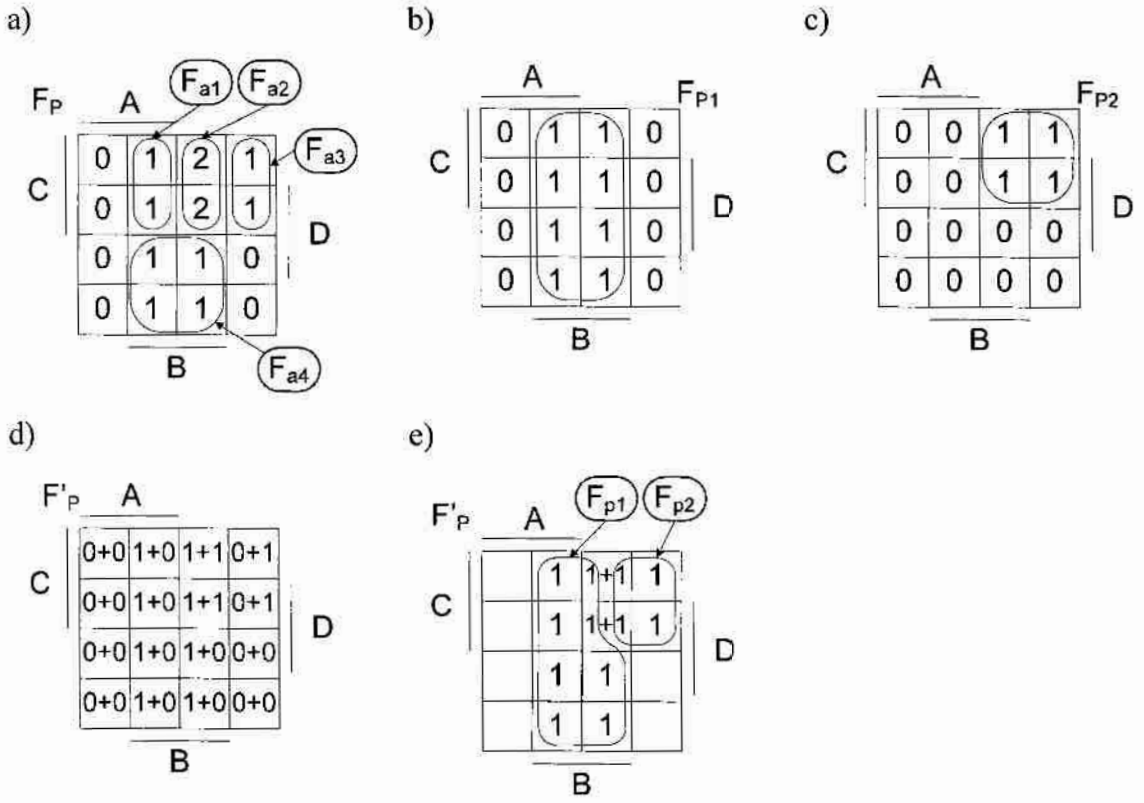
$$F_P = F_{a1} + F_{a2} + F_{a3} + F_{a4} = \overline{\overline{B} + C} + \overline{\overline{A} + \overline{B} + \overline{C}} + \overline{A + B + \overline{C}} + 2 \cdot \overline{A + \overline{B} + \overline{C}} \qquad (2.44)$$

Opis (2.44) funkcji F_P może być zrealizowany przez układ prądowy zawierający 4 bramki o ogólnej liczbie wyjść 16.

Ten sam sposób zastosowano do odnalezienia wyrażeń opisujących funkcje F_{P1} i F_{P2}, których tabele prawdy są przedstawione w tab. 2.21, a diagramy Veitcha reprezentują odpowiednio rys. 2.24b i rys. 2.24c. Otrzymane wyrażenia przedstawiają odpowiednio wzory (2.45) i (2.46).

Tab. 2.21. Tablica prawdy funkcji F_P, F_{P1}, F_{P2}, F_P'

dec	*A*	*B*	*C*	*D*	F_P	F_{P1}	F_{P2}	F_P'
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	1	0	1	1
3	0	0	1	1	1	0	1	1
4	0	1	0	0	1	1	0	1
5	0	1	0	1	1	1	0	1
6	0	1	1	0	2	1	1	2
7	0	1	1	1	2	1	1	2
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0
10	1	0	1	0	0	0	0	0
11	1	0	1	1	0	0	0	0
12	1	1	0	0	1	1	0	1
13	1	1	0	1	1	1	0	1
14	1	1	1	0	1	1	0	1
15	1	1	1	1	1	1	0	1



Rys. 2.24. Diagramy Veitcha funkcji: F_P (a), F_{P1} (b), F_{P2} (c) oraz F_{P1} i F_{P2} (d)

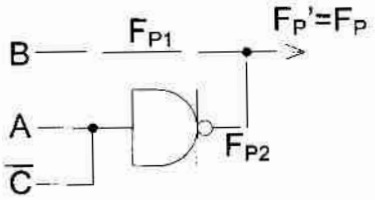
$$F_{P1} = B \tag{2.45}$$

$$F_{P2} = \overline{A + \overline{C}} \tag{2.46}$$

Wprowadzono również sumę F_p^* dunkcji bazowych F_{P1} i F_{P2}. Funkcja ta została przedstawiona na rys. 2.24d w postaci diagramu Veitcha-Karnaugha, gdzie po prawej stronie każdej kratki umieszczone są wartości logiczne dla funkcji F_{P1}, a po lewej F_{P2}. Dodatkowo rys. 2.24e przedstawia ten sam diagram, po usunięciu z niego wartości logicznego „0". Wyrażenie (2.47) przedstawia wzór otrzymanej w ten sposób funkcji F_P':

$$F_P' = F_{P1} + F_{P2} = B + \overline{A + \overline{C}} \tag{2.47}$$

Należy zauważyć, że sumując wartości w tych samych kratkach (wartość logicznego „0" nie wpływa na zmianę wartości sumy) otrzymamy diagram Veitcha identyczny jak dla funkcji F_P. Zatem diagramy te realizują tą samą funkcję. Otrzymany opis funkcji F_P' jest prostszym od wyrażenia (2.44). Realizacja praktyczna wyrażenia (2.47) na bramkach prądowych przedstawiona jest na rys. 2.25 (układ zawiera 1 bramkę, a ogólna liczba wyjść wynosi 4).



Rys. 2.25. Realizacja przykładowej funkcji F_P w technologii prądowej

Jak widać, wartość logiczną W z dowolnej kratki diagramu Veitcha-Karnaugha można przedstawić jako sumę N zmiennych $W_1, W_2, \ldots, W_N$ takich, że $W = \sum_{i=1}^{N} W_i$. Podczas minimalizacji wartości W_i uczestniczą w odnalezieniu funkcji wzorcowych, o których mowa w rozdziale 1.2.2, , przy czym wybór liczby N zmiennych W_i oraz określenie ich wartości jest w przypadku ogólnym działaniem heurystycznym, zależnym od doświadczenia projektanta. Następny przykład potwierdza tą tezę.

Przykład 2.8

Tab. 2.22 przedstawia tabelę prawdy funkcji $F_R(A, B, C, D)$, a na rys. 2.26a odpowiadający jej diagram Veitcha. Minimalizując funkcję F_R sposobem podanym w podrozdziale 1.2.2 wyróżniono funkcje bazowe F_{b1}, F_{b2}, F_{b3} i otrzymano wyrażenie (2.48) opisujące funkcję F_R:

$$F_R = F_{b1} + F_{b2} + F_{b3} = \overline{A + B + \overline{C}} + \overline{A + \overline{B} + D} + \overline{\overline{A} + \overline{B} + C} \tag{2.48}$$

Realizacja tego wyrażenia wymaga wykorzystania 3 bramek, a ogólna liczba wyjść w układzie wynosi 12.

Zauważono, że gdyby istniały dodatkowe dwie jedynki w diagramie Veitcha-Karnaugha funkcji F_R (blok F_{R3}), to można byłoby uzyskać opis prostszy, składający się z dwóch funkcji F_{R1} i F_{R2}, których diagramy Veitcha pokazane są odpowiednio na rys. 2.26b i rys. 2.26c, a opis ich przedstawiają wyrażenia (2.49) i (2.50):

$$F_{R1} = \overline{A + \overline{C}} \tag{2.49}$$

$$F_{R2} = \overline{\overline{B} + C} \tag{2.50}$$

Tab. 2.22. Tabele prawdy funkcji F_R, F_{R1}, F_{R2}, F_{R3} i F_R'

dec	*A*	*B*	*C*	*D*	F_R	F_{R1}	F_{R2}	F_{R3}	F_R'
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	1	1	0	0	1
3	0	0	1	1	1	1	0	0	1
4	0	1	0	0	1	0	1	0	1
5	0	1	0	1	0	0	1	-1	0
6	0	1	1	0	1	1	0	0	1
7	0	1	1	1	0	1	0	-1	0
8	1	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0
10	1	0	1	0	0	0	0	0	0
11	1	0	1	1	0	0	0	0	0
12	1	1	0	0	1	0	1	0	1
13	1	1	0	1	1	0	1	0	1
14	1	1	1	0	0	0	0	0	0
15	1	1	1	1	0	0	0	0	0

Aby otrzymać funkcję F_R wykorzystując funkcje F_{R1} i F_{R2} należy znaleźć taką trzecią funkcję bazową F_{R3} aby spełnione było wyrażenie (2.51):

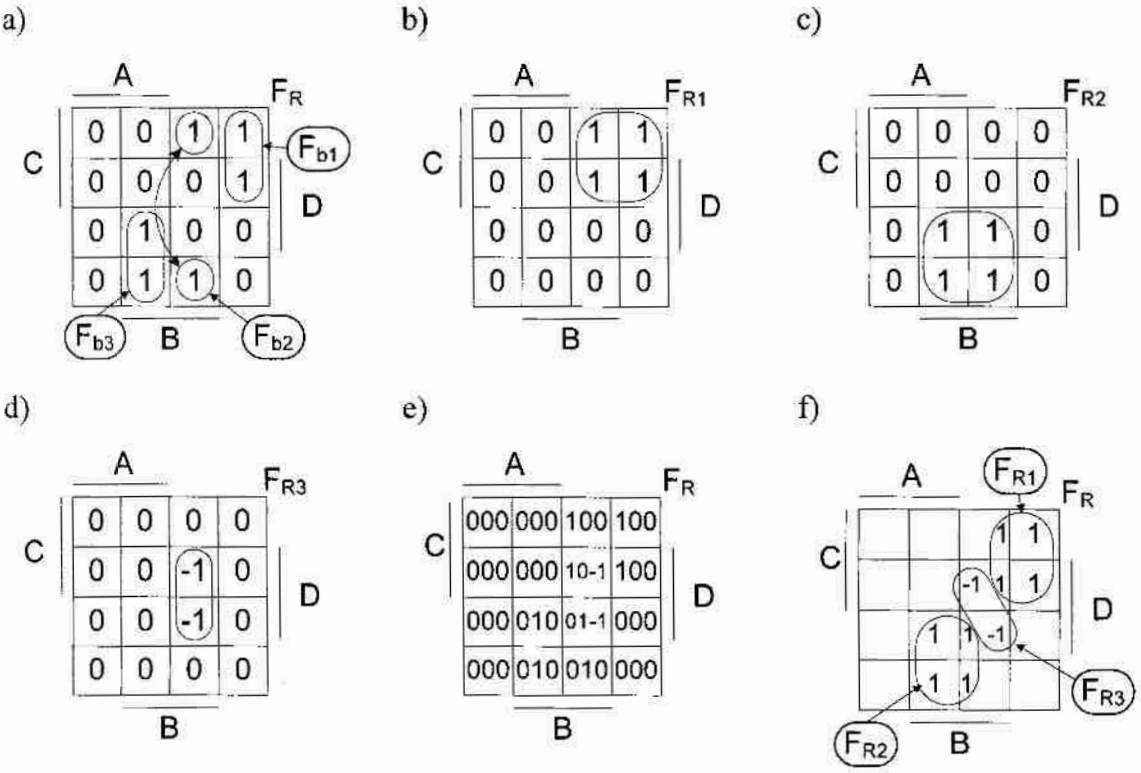
$$F_R = F_{R1} + F_{R2} + F_{R3} \tag{2.51}$$

Funkcja F_{R3} powinna wprowadzać dwie wartości „-1" w miejsce dodatkowo wstawionych jedynek. Diagram Veitcha dla tej funkcji przedstawiony jest na rys. 2.26d, a jej opis przedstawia wyrażenie (2.53).

Na rys. 2.26e przedstawione są funkcje bazowe F_{R1}, F_{R2}, F_{R3} (w jednym diagramie Veitcha) realizujące wyrażenie (2.51), gdzie poszczególne wartości w kratkach odpowiadają wartościom dla funkcji F_{R1} (z lewej), F_{R2} (w środku) i F_{R3} (z prawej). Dodatkowo rys. 2.26f przedstawia ten sam diagram, po usunięciu z niego wartości logicznych „0". Wyrażenie (2.52) reprezentuje w algebrze bramek prądowych opis funkcji F_R jako sumy funkcji bazowych F_{R1}, F_{R2}, F_{R3}:

$$F_R = F_{R1} + F_{R2} + F_{R3} = \overline{A + \overline{C}} + \overline{\overline{B} + C} + \overline{\overline{A + \overline{B} + \overline{D}}} \tag{2.52}$$

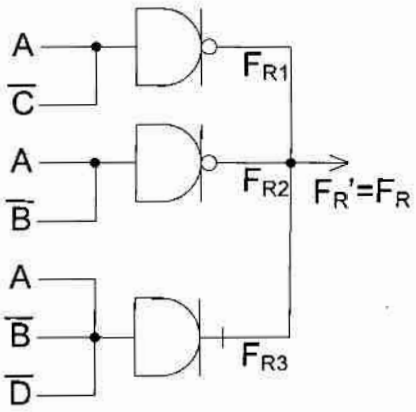
$$F_{R3} = -1 \cdot \overline{A + \overline{B} + \overline{D}} = \overline{\overline{A + \overline{B} + \overline{D}}} \tag{2.53}$$



Rys. 2.26. Diagramy Veitcha funkcji F_R, F_{R1}, F_{R2}, F_{R3}

Realizacje praktyczną funkcji F_R w oparciu o wyrażenie (2.52) na bramkach prądowych przedstawiono na rys. 2.27. Układ ten składa się z 3 bramek, a ogólna liczba wyjść w układzie wynosi 10.

Jak widać, technologia bramek prądowych umożliwia również wykonanie operacji sumy na ujemnych wartościach logicznych. Właściwość ta zwiększa możliwości poszukiwań zminimalizowanych opisów funkcji oraz pozwala wprowadzić pewną modyfikację stosowanego do tej pory *sposobu 2* minimalizacji funkcji logicznych w algebrze bramek prądowych opartego o wprowadzenie funkcji bazowych.



Rys. 2.27. Realizacja funkcji F_R (2.52) w technologii prądowej

Dowolna funkcja logiczna o binarnych argumentach $Y = F(A_1, A_2,\ldots, A_k)$, gdzie $A_i \in \{0,1\}$ może zostać przedstawiona w postaci tabeli prawdy zawierającej 2^k wierszy X_i, gdzie indeks i oznacza numer wiersza ($i = 0, 1,..., 2^k\text{-}1$). Każdy wiersz X_i w tej tablicy zawiera określone wartości (kombinacje) wszystkich argumentów $A_1, A_2,\ldots, A_k$ funkcji oraz odpowiednią dla tej kombinacji wartość funkcji W_i. Ponadto, zwykle wartości argumentów $A_1, A_2,\ldots, A_k$ reprezentują wartość liczby dziesiętnej i (tj. numeru wiersza) w kodzie binarnym. Wynika stąd, że pojedynczy wiersz tabeli prawdy może być przedstawiony za pomocą wyrażenia (2.54):

$$F[A_1, A_2,\ldots, A_N] = F[i] = W_i \tag{2.54}$$

Przyjęte założenia pozwalają sformułować w sposób formalny warunek, który powinien być brany pod uwagę przez projektanta przy wyborze funkcji bazowych w celu minimalizacji zadanej funkcji Y. Otóż funkcje bazowe F_j (j=1,...,K) należy wybierać tak, aby ich suma algebraiczna była równa funkcji Y zgodnie z następującym wyrażeniem (2.55):

$$F_1[i] + F_2[i] + \ldots + F_K[i] = F[i] \tag{2.55}$$

dla każdego $i = 0, 1,..., 2^k\text{-}1$.

W przypadku wykorzystania, do minimalizacji funkcji logicznych o argumentach binarnych, diagramów Veitcha-Karnaugha warunek ten oznacza, że każdy utworzony w diagramie blok określa jedną funkcję bazową F_j. Ile bloków projektant utworzy na diagramie Veitcha-Karnaugha, tyle będzie wynosiła liczba K różnych funkcji bazowych. Poza tym, warunek (2.55), który jest podstawą modyfikacji opisanego w podrozdziale 1.2.2 *sposobu 2*, pozwala na łączenie w jeden j-ty blok nawet tych kratek diagramu, które odpowiadają różnym wartościom funkcji Y. Należy zaznaczyć, że zmodyfikowany sposób minimalizacji (*zmodyfikowany sposób 2*) może być oparty o inną metodę minimalizacji funkcji binarnych – nie koniecznie to musi być metoda Veitcha-Karnaugha. Jednak w tej postaci raczej się on nie

nadaje do realizacji komputerowej, ponieważ pierwszy etap – wybór funkcji bazowych jest wykonywany w sposób heurystyczny. Ponadto w tym przypadku liczba K już nie zależy bezpośrednio od liczby wszystkich różnych od zera wartości funkcji Y. Jednak tak samo, jak w *sposobie 2*, w tym sposobie też dla każdego sformowanego bloku reprezentującego funkcję F_j, można zastosować dowolną znaną metodę minimalizacji, np. Quine'a-McCluskey'a, a wynik minimalizacji przekształcić w odpowiednie wyrażenie algebry prądowej. Ostateczne wyrażenie dla funkcji Y formuje się jako suma algebraiczna otrzymanych funkcji bazowych ewentualnie powielonych odpowiednią liczbę razy (tj. pomnożona przez wartość W_j, jeśli funkcja bazowa F_j jest funkcją binarną). Zaletą *zmodyfikowanego sposobu 2* jest to, że pozwala on (szczególnie przy minimalizacji funkcji arytmetyki N-wartosciowej oraz funkcji logicznych z małą liczbą argumentów w oparciu o diagramy Veitcha-Karnaugha) na otrzymanie rozwiązań znacznie lepszych, niż za pomocą innych metod minimalizacji. Niestety jakość otrzymanego rozwiązania w dużym stopniu zależy od doświadczenia projektanta. Z tego powodu, w następnym rozdziale pracy przedstawiono kilka projektów różnego stopnia złożoności układów prądowych przeznaczonych do działania w arytmetykach binarnej, N-wartosciowej, *modulo N* i RNS, i zaprojektowanych za pomocą *zmodyfikowanego sposobu 2*. Złożoność sprzętowa zaprojektowanych układów jest nawet kilkukrotnie mniejsza od złożoności sprzętowej ich odpowiedników napięciowych.

2.3. Wykorzystanie opracowanych sposobów minimalizacji do projektowania binarnych jednostek operacyjnych

2.3.1. Projektowanie sumatorów jednobitowych

Jednym z najczęściej wykorzystywanych układów cyfrowych jest układ sumatora jednobitowy, którego tabela prawdy oraz diagramy Veitcha dla funkcji sumy S i przeniesienia C_{out} przedstawione są na rys. 2.28. Wyrażenia (2.56) i (2.57) przedstawiają opis obu funkcji w algebrze Boole'a po minimalizacji:

$$S = (A \cdot B \cdot C_{in}) \vee (A \cdot \overline{B} \cdot \overline{C}_{in}) \vee (\overline{A} \cdot B \cdot \overline{C}_{in}) \vee (\overline{A} \cdot \overline{B} \cdot C_{in}) \tag{2.56}$$

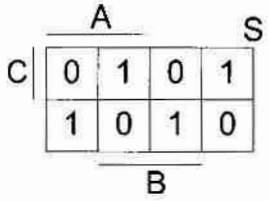
$$C_{out} = A \cdot B \vee A \cdot C_{in} \vee B \cdot C_{in} \tag{2.57}$$

Poniżej autor przedstawia proces minimalizacji funkcji S i C_{out}, korzystając z opisanych w podrozdziale 1.2.2 *sposobu 1* i *sposobu 2*, a następnie w oparciu o wprowadzone w podrozdziale 2.1 funkcje wzorcowe.

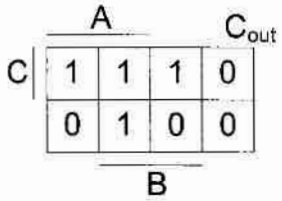
a)

C_{in}	A	B	S	C_{out}
0	0	0	0	0
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	1	1

b)



c)



Rys. 2.28. Tablice prawdy (a) i diagramy Veitcha sumy S (b) i przeniesienia C_{out} (c) dla pełnego sumatora jednobitowego

Rozwiązanie 1

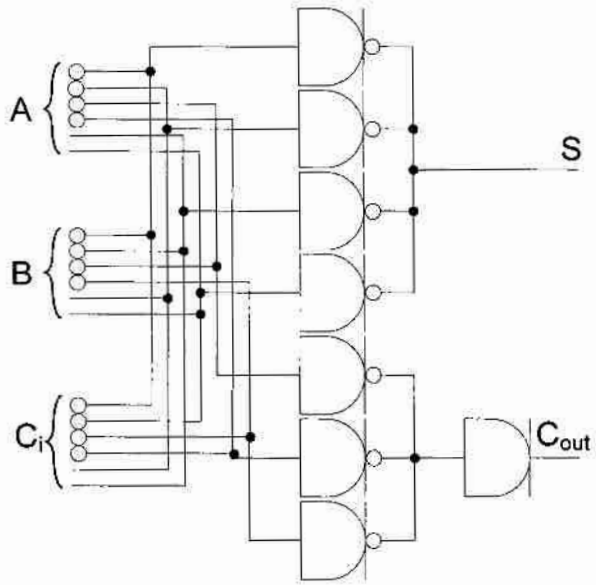
Powyższe opisy funkcji S i C_{out} służą podstawą do uzyskania opisów tych funkcji w algebrze bramek prądowych (zgodnie ze *sposobem 1*). Wyrażenie boolowskie opisujące funkcję sumy S po konwersji przedstawia sobą 4–elementową dysjunkcję implikantów prostych przy czym spełniony jest warunek (1.27) (poszczególne implikanty nie mają wspólnych kratek w diagramie Veitcha). Zatem, dokonując konwersji, można skorzystać z wyrażenia (1.29) i w rezultacie otrzymać wyrażenie (2.58) będące opisem funkcji S w algebrze bramek prądowych:

$$S = \overline{\overline{A} + \overline{B} + \overline{C_{in}}} + \overline{\overline{A} + B + C_{in}} + \overline{A + \overline{B} + C_{in}} + \overline{A + B + \overline{C_{in}}} \qquad (2.58)$$

Do konwersji funkcji C_{out} należy użyć wzoru (1.23) otrzymując wyrażenie (2.59), które opisuje tę funkcję w algebrze bramek prądowych:

$$C_{out} = \overline{\overline{\overline{A} + \overline{B}} + \overline{\overline{A} + \overline{C_{in}}} + \overline{\overline{B} + \overline{C_{in}}}} \qquad (2.59)$$

Realizacja sumatora jednobitowego na bramkach prądowych w oparciu o wyrażenia (2.58) i (2.59) pokazana jest na rys. 2.29. Układ składa się z 8 bramek o ogólnej liczbie wyjść 26, a jego realizacja wymaga $LTmin$=76 tranzystorów (w tab. 2.23 znajduje się porównanie ilościowe różnych realizacji sumatorów binarnych). Jak widać, otrzymane rozwiązanie jest dość złożone, szczególnie pod względem liczby tranzystorów.



Rys. 2.29. Jednobitowy sumator prądowy opisany za pomocą wyrażeń (2.58) i (2.59)

Rozwiązanie 2

Pełne sumatory jednobitowe mają dwie funkcje wyjściowe o wspólnych argumentach, warto więc wykorzystać ten fakt oraz *sposób 2* minimalizacji. Zauważono, że funkcja $C_{out,}$ może być przedstawiona jako zanegowana suma algebraiczna następujących trzech funkcji bazowych: $a_1=\overline{A}_i$, $a_2=\overline{B}_i$ i $a_3=\hat{C}_{in}$ (których diagramy Veitcha są przedstawione na rys. 2.30a-c). Sumę tę reprezentuje wyrażenie (2.60), natomiast na rys. 2.30d przedstawiona jest suma funkcji bazowych a_1, a_2, a_3 bez negacji:

$$C_{out} = \overline{\overline{A}_i + \overline{B}_i + \hat{C}_{in}} \tag{2.60}$$

a)

C \ A	$\overline{A}$	$\overline{A}$	A	A
C	0	0	1	1
$\overline{C}$	0	0	1	1
	$\overline{B}$			B

a_1

b)

C \ A	$\overline{A}$	$\overline{A}$	A	A
C	1	0	0	1
$\overline{C}$	1	0	0	1
	$\overline{B}$			B

a_2

c)

C \ A	$\overline{A}$	$\overline{A}$	A	A
C	-1	-1	-1	-1
$\overline{C}$	0	0	0	0
	$\overline{B}$			B

a_3

d)

C \ A	$\overline{A}$	$\overline{A}$	A	A
C	0	-1	0	1
$\overline{C}$	1	0	1	2
	$\overline{B}$			B

$a_{1\text{-}3}$

Rys. 2.30. Diagramy Veitcha funkcji bazowych wybranych do minimalizacji funkcji C_{out}

Podobnie dla funkcji sumy S wybrane zostały trzy funkcje $a_4 = \overline{C}_{out}$, $a_5 = \overline{\overline{A} + \overline{B} + \overline{C}_{in}}$ i $a_6 = \overline{\overline{A + B + C_{in}}}$ (rys. 2.31a-c), a ostateczne wyrażenie dla funkcji sumy S zostało otrzymane jako suma algebraiczna funkcji bazowych a_4, a_5, a_6 (2.61):

$$S = \overline{C}_{out} + \overline{\overline{A} + \overline{B} + \overline{C}_{in}} + \overline{\overline{A + B + C_{in}}} \tag{2.61}$$

a)

C \ A B	00	01	11	10
0	0	0	1	0
1	1	0	1	1

a_4

b)

C \ A B	00	01	11	10
0	0	1	0	0
1	0	0	0	0

a_5

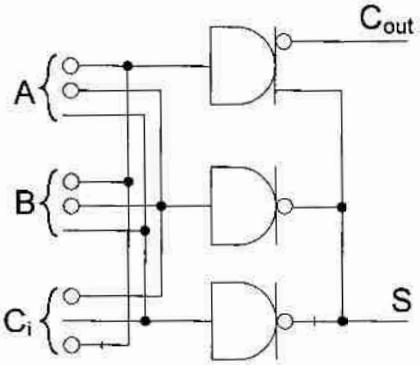
c)

C \ A B	00	01	11	10
0	0	0	0	0
1	0	0	0	-1

a_6

Rys. 2.31. Diagramy Veitcha funkcji bazowych wybranych do minimalizacji funkcji S

Na rys. 2.32 przedstawiony jest układ sumatora jednobitowego zrealizowany w oparciu o wyrażenia (2.60) i (2.61). Układ ten składa się z 3 bramek o ogólnej liczbie wyjść 13, a realizacja wymaga wykorzystania *LTmin*=42 tranzystorów (w tab. 2.23 znajduje się porównanie ilościowe różnych realizacji sumatorów binarnych). Jak widać, złożoność sprzętowa otrzymanego za pomocą drugiego sposobu minimalizacji sumatora jest znacznie mniejsza.



Rys. 2.32. Zoptymalizowany układ jednobitowego sumatora prądowego

Rozwiązanie 3

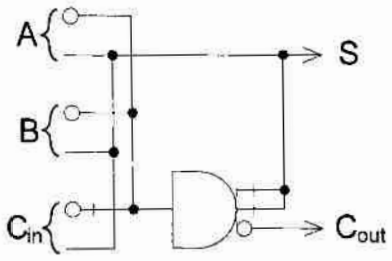
Analizując diagramy Veitcha funkcji S i C_{out} (rys. 2.28b i c) można zauważyć, że funkcja sumy S jest 3-argumentową funkcją wzorcową typu XOR (XBlok(3)), natomiast funkcja przeniesienia C_{out} jest 3-argumentową funkcją wzorcową typu T (TBlok(3)). Zatem funkcje te można przedstawić za pomocą wyprowadzonych w poprzednich podrozdziałach wyrażeń (2.62) i (2.63):

$$C_{out} = \overline{\overline{A} + \overline{B} + \hat{C}_{in}} = \overline{\overline{A} + \hat{B} + \overline{C}_{in}} = \overline{\hat{A} + \overline{B} + \overline{C}_{in}} \tag{2.62}$$

$$S = A + B + C_{in} + 2 \cdot \overline{(\overline{A} + \overline{B} + \hat{C}_{in})} = A + B + C_{in} + 2 \cdot \hat{C}_{out} \tag{2.63}$$

Wyrażenie (2.63) przedstawia zależność pomiędzy funkcją sumy i przeniesienia. Warto zauważyć, że układ zbudowany na podstawie powyższych wyrażeń najpierw określi wartość przeniesienia a na jej podstawie wartość sumy. Na rys. 2.33a przedstawiono realizacje sumatora jednobitowego przy wykorzystaniu wyrażeń (2.62) i (2.63). Układ ten składa się z 1 bramki o ogólnej liczbie wyjść 9, a jego realizacja wymaga użycia *LTmin*=22 tranzystorów, lub (jeśli zamienimy operacje podwójnej-inwersji na inwersje), użycia *LTmin*=19 tranzystorów (rys. 2.33b). W tab. 2.23 umieszczono porównanie realizacji różnych sumatorów jednobitowych w zrealizowanych algebrze bramek prądowych oraz w algebrze Boole'a. Można zauważyć, że stosując funkcje wzorcowe wprowadzone w podrozdziale 2.1. zaprojektowano układy prądowe mniej złożone (biorąc pod uwagę liczbę tranzystorów jak i liczbę połączeń) od układów napięciowych.

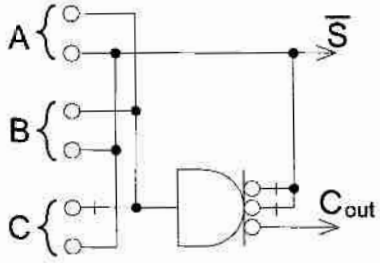
a)



$$C_{out} = \overline{\overline{A} + \overline{B} + \hat{C}_{in}}$$

$$S = A + B + C_{in} + 2 \cdot \hat{C}_{out}$$

b)



$$C_{out} = \overline{\overline{A} + \overline{B} + \hat{C}_{in}}$$

$$S = \overline{\overline{A} + \overline{B} + \hat{C}_{in} + 2 \cdot \hat{\overline{C}}_{out}}$$

Rys. 2.33. Przykłady różnych układów pełnych sumatorów jednobitowych i ich opis w algebrze bramek prądowych

Tab. 2.23. Podstawowe parametry sumatorów jednobitowych

	bramki	połączenia	tranzystory		opóźnienie
			LT	*LTmin*	[bramki]
sumator z rys. 2.29	8	26	85	76	2
sumator z rys. 2.32	3	13	51	42	2
sumator z rys. 2.33a	1	9	31	22	1
sumator z rys. 2.33b	1	9	28	19	1
sumator napięciowy [41]	9	22	36		5

2.3.2. Projekt sumatora czterobitowego z równoległym przeniesieniem

W rzeczywistych układach cyfrowych najczęściej nie występuje sumator jednobitowy lecz sumator wielobitowy. Istnieją różne rodzaje sumatorów wielobitowych, które się różnią układami realizującymi przeniesienie wyjściowe. Dzieje się tak dlatego, że najprostszy sprzętowo sumator z szeregowym przeniesieniem zwykle jest zbyt wolny i praktycznie nie jest wykorzystywany w systemach jednoukładowych czasu rzeczywistego. Stosowany tam jest sumator z równoległym przeniesieniem (ang. look-ahead adder) [41], który jest najszybszy ze znanych sumatorów i niestety, najbardziej złożony sprzętowo. Z tego powodu w systemach jednoukładowych zwykle sumator wielobitowy konstruowany jest (jak z klocków) z 4-bitowych sumatorów z równoległym przeniesieniem. W związku z tym, w ramach przeprowadzonych badań autor opracował taki 4-bitowy sumator „look-ahead" w oparciu o bramki prądowe i porównał jego złożoność sprzętową z odpowiednim sumatorem napięciowym pod względem liczby wykorzystanych bramek, połączeń, tranzystorów i powierzchni w układzie ASIC. Należy zaznaczyć, że topografie obu układów zostały opracowane w technologii CMOS 0,6 μm przez dr inż. Piotra Pawłowskiego.

Tab. 2.24 przedstawia podstawowe zależności pomiędzy argumentami w sumatorze jednobitowym, które można wykorzystać do realizacji równoległego przeniesienia

Tab. 2.24. Tabela prawdy pełnego sumatora jednobitowego oraz dodatkowych funkcji bazowych przy projektowaniu sumatora czterobitowego

C_i	A_i	B_i	C_{i+1}	S_{i+1}	$A_i \cdot B_i$	$A_i \vee B_i$	$\overline{A_i \cdot B_i}$	$(A_i \vee B) \cdot C_i$	$\overline{(A_i \vee B) \cdot C_i}$	$\overline{Q_i} \cdot \overline{P_i \cdot C_i}$
0	0	0	0	0	0	0	1	0	1	1
0	0	1	0	1	0	1	1	0	1	1
0	1	0	0	1	0	1	1	0	1	1
0	1	1	1	0	1	1	0	0	1	0
1	0	0	0	1	0	0	1	0	1	1
1	0	1	1	0	0	1	1	1	0	0
1	1	0	1	0	0	1	1	1	0	0
1	1	1	1	1	1	1	0	1	0	0
					Q_i	P_i	$\overline{Q_i}$	$P_i \cdot C_i$	$\overline{P_i \cdot C_i}$	

W tab. 2.24 wprowadzono dwie dodatkowe funkcje Q_i i P_i określone odpowiednio wyrażeniami (2.64) i (2.65):

$$Q_i = A_i \cdot B_i \tag{2.64}$$

$$P_i = A_i \vee B_i \tag{2.65}$$

Na podstawie tych wartości określić można wyrażenie odpowiadające funkcji XOR (2.66)(częściowa wartość sumy):

$$A_i \oplus B_i = \overline{Q_i} \cdot P_i = S_p \tag{2.66}$$

oraz końcowe wyrażenie na funkcję sumy (2.67):

$$S = S_p \oplus C_i \tag{2.67}$$

Dla projektowanego sumatora czterobitowego funkcje przeniesienia $C1$, $C2$, $C3$, $C4$ odpowiednio reprezentują wyrażenia (2.68) – (2.71).:

$$C_1 = \overline{\overline{Q_0} \cdot \overline{P_0 \cdot C_0}} \tag{2.68}$$

$$C_2 = \overline{\overline{Q_1} \cdot \overline{P_1 \cdot Q_0} \cdot \overline{P_1 \cdot P_0 \cdot C_0}} \tag{2.69}$$

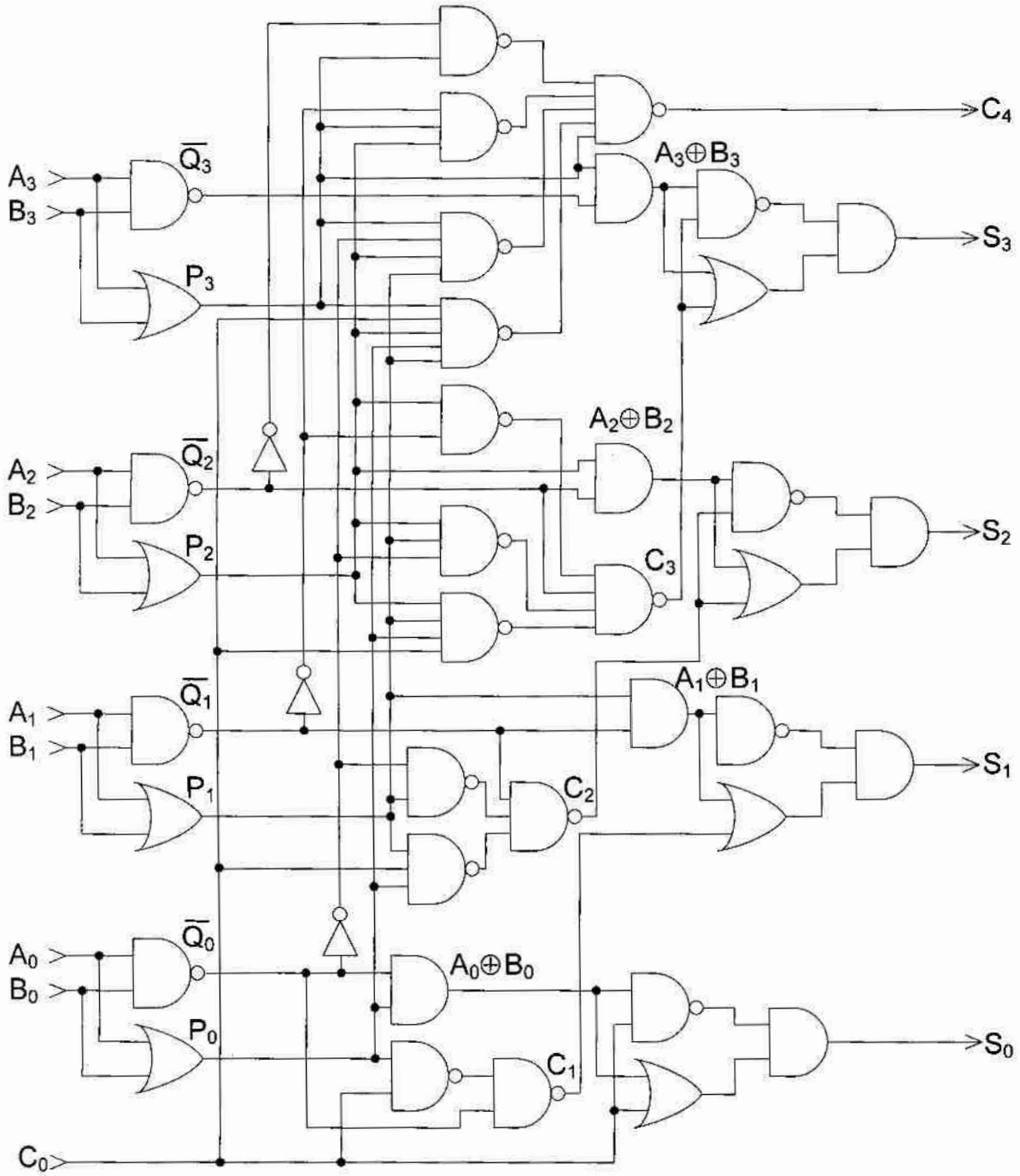
$$C_3 = \overline{\overline{Q_2} \cdot \overline{P_2 \cdot Q_1} \cdot \overline{P_2 \cdot P_1 \cdot Q_0} \cdot \overline{P_2 \cdot P_1 \cdot P_0 \cdot C_0}} \tag{2.70}$$

$$C_4 = \overline{\overline{Q_3} \cdot \overline{P_3 \cdot Q_2} \cdot \overline{P_3 \cdot P_2 \cdot Q_1} \cdot \overline{P_3 \cdot P_2 \cdot P_1 \cdot Q_0} \cdot \overline{P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0}} \tag{2.71}$$

Na podstawie powyższych wzorów zrealizowano układ czterobitowego sumatora z szybkim przeniesieniem w technologii bramek napięciowych (rys. 2.34).

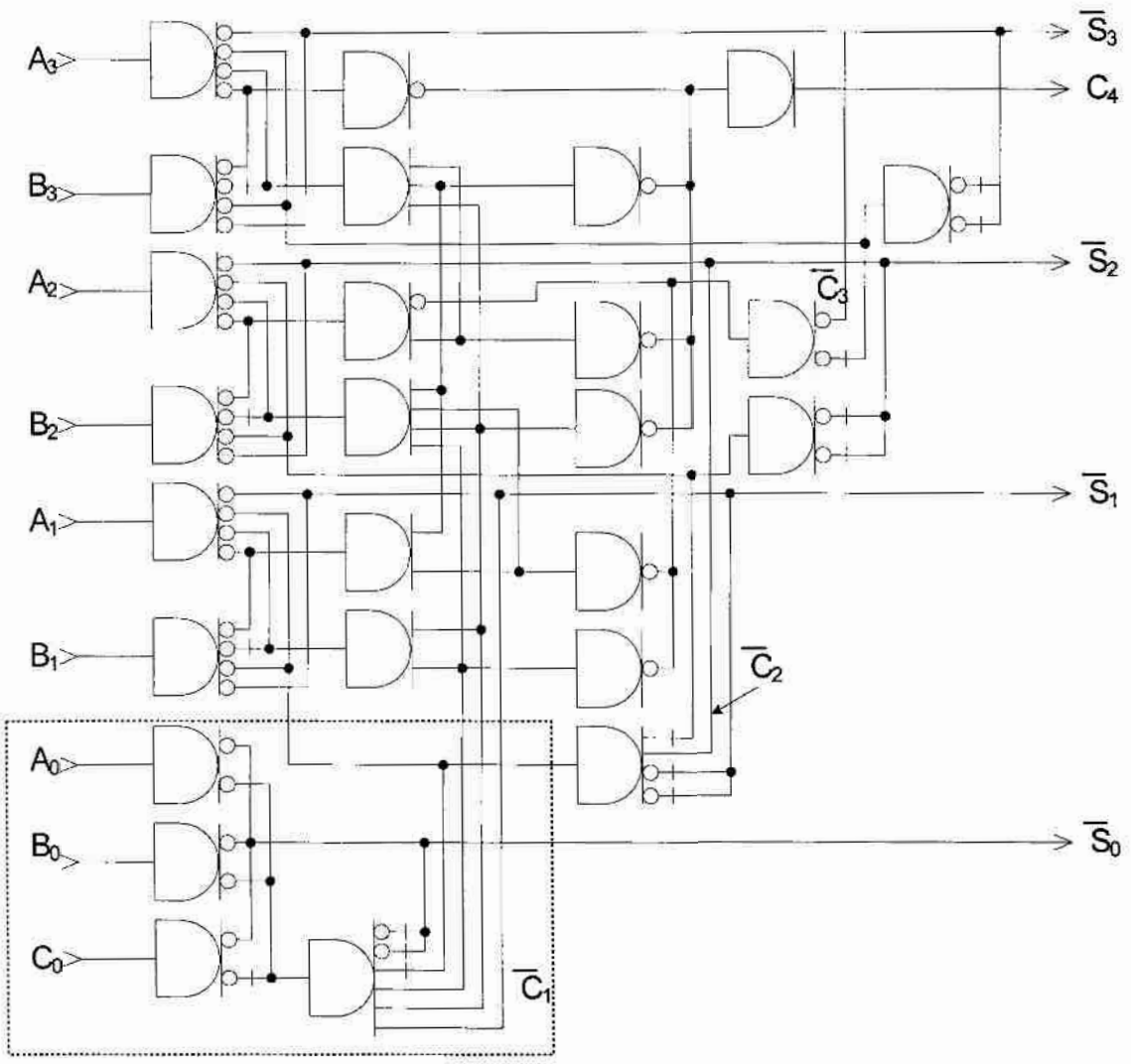
Rys. 2.35 przedstawia odpowiednik powyższego sumatora czterobitowego zrealizowanego w technologii bramek prądowych. Funkcje sumy i przeniesienia zrealizowano odpowiednio jako funkcję XBlok(3) i TBlok(3). Kropkowanym kwadratem wyróżniono realizację sumy i przeniesienia dla najmłodszych bitów liczb A i B.

Topografię obu układów zaprojektowano, kierując się regułami technologicznymi procesu technologicznego CMOS o minimalnej długości kanału tranzystora MOS 0,6 mikrometra, o bramce polikrzemowej i trzech warstwach metalizacji AMS CYE 0.6u firmy Austria Mikro Systeme. Następnie przekazano projekt topografii układu do firmy Tima-CMP w Grenoble, gdzie układ wyprodukowano. Tab. 2.25 przedstawia wybrane parametry obu wykonanych praktycznie sumatorów. Analiza danych z tab. 2.25 wskazuje, że sumator zbudowany z bramek prądowych zawiera mniej bramek oraz połączeń w porównaniu z układem napięciowym (liczba połączeń jest równa ogólnej liczbie wejść poszczególnych bramek oraz liczby wejść i wyjść układu dla układu napięciowego, natomiast dla układu prądowego jest równa ogólnej liczbie wyjść poszczególnych bramek oraz liczbie wejść i wyjść układu).



Rys. 2.34. Projekt szybkiego sumatora czterobitowego (ang. look-ahead) w technologii napięciowej

Liczba tranzystorów potrzebnych do zrealizowania obu układów wskazuje jednak, że układ prądowy potrzebuje ich blisko dwa razy więcej. Dzieje się tak dlatego, że w układzie sumatora znaleziono tylko jedną funkcję TBlok(3) oraz jedną funkcję XBlok(3). Ale i tak powierzchnie zajmowane przez układ napięciowy i prądowy są porównywalne (różnica wynosi ok. 2,5%).



Rys. 2.35. Projekt szybkiego sumatora czterobitowego (ang. look-ahead) w technologii prądowej

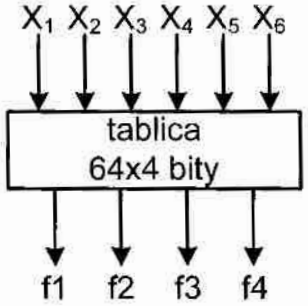
Tab. 2.25. Wybrane parametry zrealizowanych praktycznie szybkich sumatorów czterobitowych

Parametr	Układ prądowy	Układ napięciowy
Liczba bramek	26	38
Liczb połączeń	75	94
Liczba tranzystorów	436	222
Powierzchnia bramek	20317,44μm^2	11185,20μm^2
Powierzchnia wraz z połączeniami	43000μm^2	42000μm^2

2.3.3. Projektowanie układów realizujących funkcje S-Bloków algorytmu kryptograficznego DES

DES jest szyfrem blokowym operującym na 64-bitowym tekście jawnym, który poddawany jest permutacji początkowej a następnie, dzielony jest na dwie 32-bitowe części L i R. Realizacja sprzętowa algorytmu DES sprowadza się głównie do realizacji

operacji podstawiania, permutacji i operacji XOR. Permutacje początkowa oraz końcowa odwzorowują każdy bit wejściowy na odpowiadający mu bit wyjściowy. Operacja XOR i permutacje są proste w realizacji sprzętowej. Charakterystyczną operacją jest operacja podstawienia w S-blokach. Każdy S-blok (rys. 2.36) ma 6 wejść i cztery wyjścia. Do realizacji sprzętowej S-bloków w klasycznym rozwiązaniu stosuje się 64-komórkowe układy pamięci lub układy kombinacyjne (realizacja w FPGA wymaga użycia 4 bloków LUT na jeden bit wyniku). Każdy S-blok opisany jest przy pomocy tablicy składającej się z 4 wierszy i z 16 kolumn (tab. 2.26). Sygnały wejściowe X_2, X_3, X_4, X_5 łączy się tworząc 4-bitową liczbę określającą numer kolumny, a sygnały X_1 i X_6 tworzą 2-bitową liczbę określającą numer wiersza w tabeli. Tak wybrana liczba z tabeli przekazywana jest na wyjścia $f1, f2, f3, f4$. Wszystkie tablice są jawne. Dokładny ich opis znajduje się w [36]. Poniżej zaprezentowano projekt S-bloku $S1$, którego struktura opisana jest w tab. 2.26.



Rys. 2.36. Struktura pojedynczego S-bloku

Tab. 2.26. Zawartość bloku pamięci ROM dla S-bloku $S1$

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

W celu przetestowania możliwości opracowanych przez autora sposobów minimalizacji funkcji binarnych, podjęta została próba realizacji poszczególnych funkcji S-bloku $S1$ jako układów kombinacyjnych. W tym celu przeprowadzono minimalizację funkcji wszystkich wyjść ($f1, f2, f3, f4$) S-bloku za pomocą metody Quine-McCluskey'a. W niniejszej rozprawie jako przykład przedstawiono proces minimalizacji tylko jednego z wyjść – $f1$. Lista implikantów pierwotnych tej funkcji przedstawiona jest w tab. 2.4.

Autor przedstawia poniżej trzy różne opisy funkcji $f1$ w technologii bramek prądowych, z których pierwszy otrzymany został w oparciu o *sposób 1* opisany w podrozdziale 1.2.2,

a dwa pozostałe oparte są o funkcje wzorcowe opisane w rozdziale 2.1 i nowy sposób minimalizacji zaproponowany w rozdziale 2.2. W tab. 2.5 umieszczono listę implikantów prostych, na podstawie których można przedstawić zminimalizowany opis funkcji $f1$ w technologii bramek napięciowych. Zgodnie z otrzymanymi wynikami zaprojektowano układ zbudowany w oparciu o klasyczne bramki napięciowe (NOT, AND i OR). Główne parametry układów realizujących funkcję $f1$ S-bloku przedstawiono w tab. 2.30, i posłużą one do porównania z analogicznym układem zrealizowanym w technologii bramek prądowych.

Rozwiązanie 1

Korzystając ze *sposobu 1* opisanego w rozdziale 1.2.2 dokonano konwersji otrzymanej funkcji $f1$ w logice Boole'a do odpowiedniego opisu w algebrze bramek prądowych. Wyrażenie po konwersji przedstawia (2.72), natomiast główne parametry układu realizującego wyrażenie (2.72) przedstawiono w tab. 2.30. Bardziej szczegółowy opis procesu projektowania przedstawiony jest w [A14].

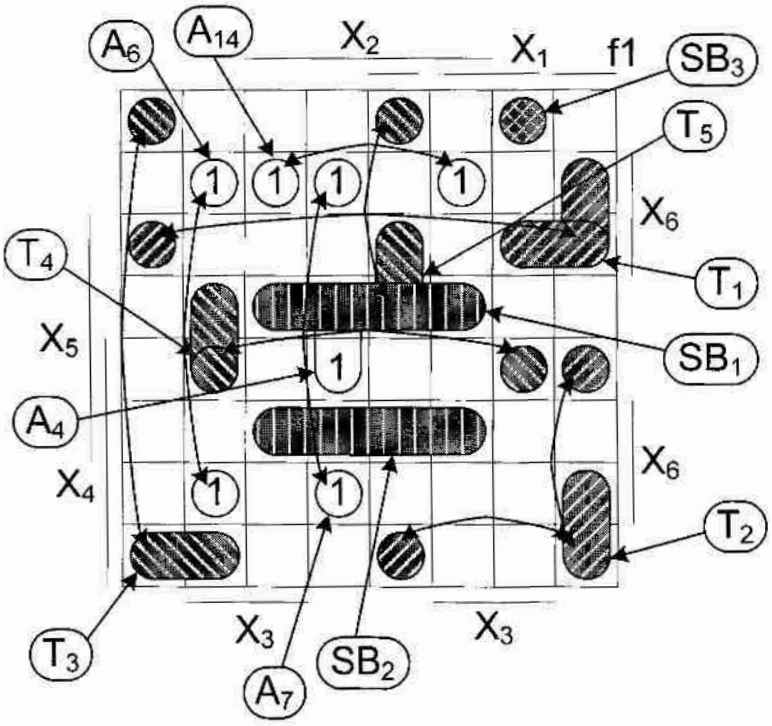
$$\begin{aligned} f1 = {} & X_1+X_2+X_3+X_4+X_5+X_6+X_1+X_2+X_3+X_5+X_6+X_1+X_2+X_4+X_5+X_6 \\ & +X_1+X_2+X_3+X_5+X_6+X_1+X_2+X_3+X_5+X_6+X_2+X_3+X_4+X_5+X_6+X_1+X_2+X_3+X_5+X_6 \\ & +X_1+X_2+X_3+X_5+X_6+X_1+X_2+X_3+X_4+X_6+X_1+X_3+X_4+X_5+X_6+X_1+X_2+X_3+X_4+X_5 \\ & +X_1+X_2+X_3+X_4+X_6+X_1+X_2+X_3+X_4+X_6+X_2+X_3+X_4+X_5+X_6+X_2+X_3+X_4+X_5+X_6 \\ & +X_1+X_2+X_3+X_4+X_5+X_1+X_2+X_4+X_5+X_6+X_2+X_4+X_5+X_6+X_2+X_4+X_5+X_6 \end{aligned} \qquad (2.72)$$

Rozwiązanie 2

Z grupy implikantów prostych (po otrzymaniu ich dowolną klasyczną metodą np. Quine'a-mcCluskey'a, Espresso – tab. 2.5) do wyrażenia końcowego funkcji dopisywane są opisy wyrażeń zawierających dwie lub więcej gwiazdek (na rys. 2.37 loki SB_1 i SB_2) i bez gwiazdek (na rys. 2.37 blok SB_3). Pozostałe implikanty proste posłużą do wyszukania funkcji wzorcowych typu T.

W pierwszym kroku należy rozwinąć implikanty podobne (wstawiając w miejsce „*" najpierw „0", a potem „1") tworząc pary implikantów pierwotnych, które posłużą do stworzenia tablic TX_i. Wynik został przedstawiony w tab. 2.27. Bloki T_1 i T_2 są funkcjami typu TBlok(3), pozostałe bloki – to są funkcje typu TBlok(2).

W tab. 2.28 pokazano sposób określania na podstawie tabel TX_i opisu funkcji w algebrze bramek prądowych.



Rys. 2.37. Ilustracja minimalizacji funkcji *f*1 w *rozwiązaniu 2* w oparciu o funkcje wzorcowe

Końcowe ogólne wyrażenie opisujące funkcję *f*1 w algebrze bramek prądowych jest przedstawione poniżej:

$$f1 = \overline{SB_1 + SB_2 + SB_3 + T_1 + T_2 + T_3 + T_4 + T_5 + A_4 + A_6 + A_7 + A_{14}} \qquad (2.73)$$

Do realizacji funkcji SB_1 – SB_3, A_4, A_6, A_7, A_{14} potrzeba po jednej bramce z wyjściem typu inwerter, a funkcji T_1 – T_5 po dwie bramki: jedna z wyjściem typu inwerter i jedna z wyjściem typu podwójny-inwerter. Cały układ będzie się składał z 18 bramek o ogólnej liczbie wyjść 64 i LT=321 tranzystorów.

Rozwiązanie 3

Z grupy implikantów prostych (po otrzymaniu ich dowolną klasyczną metodą np. Quine'a-McCluskey'a, Espresso) do wyrażenia końcowego funkcji dopisywane są opisy wyrażeń zawierających dwie lub więcej gwiazdek i bez gwiazdek, podobnie jak to miało miejsce w *rozwiązaniu 2*. Pozostałe implikanty proste (N-1 argumentowe) należy zamienić na odpowiadające im implikanty pierwotne, a następnie wykonać pierwszy etap algorytmu Quine'a-McCluskey'a aby otrzymać skróconą listę implikantów prostych. Mając implikanty proste postępujemy dalej tak jak w *rozwiązaniu 2*.

Tab. 2.27. Lista implikantów podobnych funkcji $f1$ bloku S w algorytmie DES – *rozwiązanie 2*

dec	X_1	X_2	X_3	X_4	X_5	X_6	implikanty podobne TX_i				funkcja wzorcowa
							N=1	N=2	N=3	N=4	
0	0	0	0	0	0	0	4	12			T_3
3	0	0	0	0	1	1	35				
4	0	0	0	1	0	0	0				
9	0	0	1	0	0	1	13				
10	0	0	1	0	1	0	14				
12	0	0	1	1	0	0	4				
13	0	0	1	1	0	1	9				
14	0	0	1	1	1	0	10	46			T_4
17	0	1	0	0	0	1	21				
18	0	1	0	0	1	0	22				
21	0	1	0	1	0	1	17				
22	0	1	0	1	1	0	18				
25	0	1	1	0	0	1	57				
33	1	0	0	0	0	1	35				
35	1	0	0	0	1	1	3	33	43		T_1
36	1	0	0	1	0	0	38	52	37		T_2
37	1	0	0	1	0	1	36				
38	1	0	0	1	1	0	36				
43	1	0	1	0	1	1	35				
46	1	0	1	1	1	0	14				
48	1	1	0	0	0	0	50				
50	1	1	0	0	1	0	48	51			T_5
51	1	1	0	0	1	1	50				
52	1	1	0	1	0	0	36				
57	1	1	1	0	0	1	25				

Końcowe ogólne wyrażenie opisujące funkcję $f1$ w algebrze bramek prądowych przedstawiono poniżej:

$$f1 = \overline{SB_1 + SB_2 + SB_3 + T_1 + T_2 + T_3 + T_4 + T_5 + A_1 + A_2 + A_3 + A_4} \tag{2.74}$$

Na rys. 2.38 zaznaczono poszczególne funkcje wchodzące w skład wyrażenia (2.74). Należy zaznaczyć, że niektóre funkcje TBlok (dla implikantów głównych o wartościach 4, 12, 38). Porównując te dwa rozwiązania autor zwrócił uwagę, że pomimo znalezienia funkcji wzorcowych typu T o większej liczbie argumentów (w drugim rozwiązaniu odnaleziono

funkcję TBlok(4), w pierwszym TBlok(3)). Ilość funkcji bazowych realizujących funkcję *f*1 nie zmieniła się. Oba rozwiązania będą zawierały podobną liczbę tranzystorów.

Tab. 2.28. Określenie wyrażeń opisujących odnalezione funkcje TBlok(3) (a-b) i TBlok(2) (c)

a)

K	1	2	3	4	5	6
T_1	X_1	X_2	X_3	X_4	X_5	X_6
	1	0	0	0	1	1
	0	0	0	0	1	1
	1	0	0	0	0	1
	1	0	1	0	1	1
S[K]	3	0	1	0	3	4

$$T_1 = \overline{X}_1 + X_3 + \hat{X}_5 + X_2 + X_4 + \overline{X}_6$$

b)

K	1	2	3	4	5	6
T_2	X_1	X_2	X_3	X_4	X_5	X_6
	1	0	0	1	0	0
	1	0	0	1	1	0
	1	1	0	1	0	0
	1	0	0	1	0	1
S[K]	4	1	0	4	1	1

$$T_2 = X_2 + X_3 + \hat{\overline{X}}_6 + \overline{X}_1 + X_3 + \overline{X}_4$$

c)

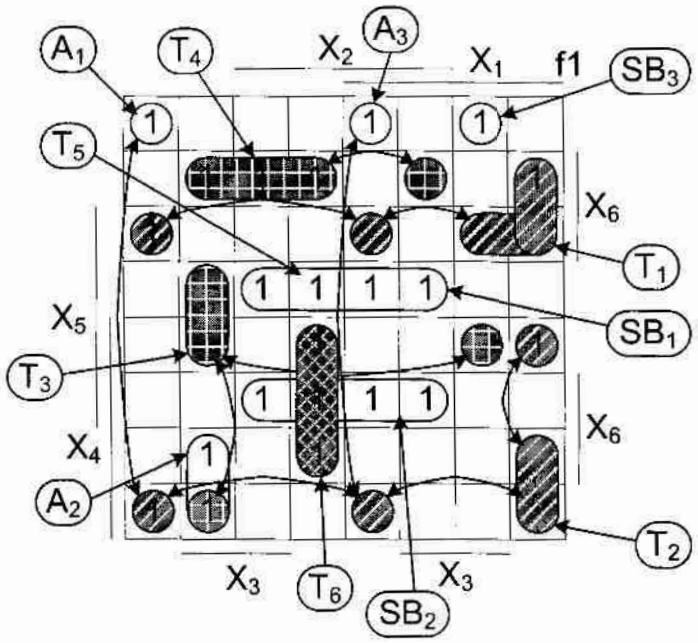
K	1	2	3	4	5	6
T_3	X_1	X_2	X_3	X_4	X_5	X_6
	0	0	0	0	0	0
	0	0	0	1	0	0
	0	0	1	1	0	0
S[K]	0	0	1	2	0	0

$$T_3 = X_3 + \hat{X}_4 + X_1 + X_2 + X_5 + X_6$$

c)

K	1	2	3	4	5	6
T_4	X_1	X_2	X_3	X_4	X_5	X_6
	0	0	1	1	1	0
	0	0	1	0	1	0
	1	0	1	1	1	0
S[K]	1	0	3	2	3	0

$$T_4 = X_1 + \hat{X}_4 + X_2 + \overline{X}_3 + \overline{X}_5 + X_6$$



Rys. 2.38. Ilustracja minimalizacji funkcji *f*1 w *rozwiązaniu 3* w oparciu o funkcje wzorcowe

Tab. 2.29. Skrócona lista implikantów pierwotnych funkcji $f1$ S-bloku $S1$ w algorytmie DES - *rozwiązanie 3*

dec	X_1	X_2	X_3	X_4	X_5	X_6	implikanty podobne TX_i				funkcja wzorcowa
							N=1	N=2	N=3	N=4	
0	0	0	0	0	0	0	4				
3	0	0	0	0	1	1	35				
4	0	0	0	1	0	0	0	12	36		
9	0	0	1	0	0	1	13	25			
10	0	0	1	0	1	0	14				
12	0	0	1	1	0	0	4	13	14		
13	0	0	1	1	0	1	9	12			
14	0	0	1	1	1	0	10	12	46		T_3
17	0	1	0	0	0	1	21	25			
18	0	1	0	0	1	0	22	50			T_5
21	0	1	0	1	0	1	17				
22	0	1	0	1	1	0	18				
25	0	1	1	0	0	1	9	17	57		T_4
33	1	0	0	0	0	1	35	37			
35	1	0	0	0	1	1	3	33	43	51	T_1
36	1	0	0	1	0	0	4	37	38	52	T_2
37	1	0	0	1	0	1	33	36			
38	1	0	0	1	1	0	36	46	51		
43	1	0	1	0	1	1	35				
46	1	0	1	1	1	0	14	38			
48	1	1	0	0	0	0	50	52			
50	1	1	0	0	1	0	18	48			
51	1	1	0	0	1	1	35	38			
52	1	1	0	1	0	0	36	48			
57	1	1	1	0	0	1	25				

Pierwsza wersja układu prądowego (*rozwiązanie 1*) zawiera podobną ilość połączeń i bramek w porównaniu do układu napięciowego, jednak liczba tranzystorów jest dużo większa. Po użyciu funkcji wzorcowych typu T układ zawiera blisko połowę mniej połączeń, ilość bramek się nie zmieniła, a liczba tranzystorów jest nieznacznie większa. Jednak analizując proporcje zachodzące pomiędzy w/w parametrami obu układów należy przypuszczać (na podstawie realizacji układu sumatora czterobitowego), że powierzchnia, którą zajęłyby układy przy realizacji praktycznej byłaby porównywalna, a nawet mniejsza

w układzie prądowym. Należy zaznaczyć, że porównanie złożoności sprzętowej układów realizujących pozostałe funkcje S-bloku $S1$ ($f2, f3, f4$) potwierdzają tę tezę.

Tab. 2.30. Podstawowe parametry układów realizujących funkcje $f1$ S-bloku $S1$

Parametry	układy prądowe			układ napięciowy
	rozwiązanie			
	1	2	3	
Liczba połączeń	119	64	74	113
Liczba bramek	20	18	18	20
Liczba tranzystorów	422	303	326	266

2.3.4. Projekt modułu szybkiego przeniesienia dla układu FPGA

Jednym z pierwszych dużych projektów, którego autor niniejszej rozprawy był współwykonawcą był projekt odpowiednika prądowego bloku SLICE układu FPGA SPARTAN II firmy Xilinx. Dokładny opis projektu przedstawiony jest w [A3, A5 – A7, A11].

Jednym z najważniejszych bloków układu SLICE jest moduł realizujący szybkie przeniesienie wyjściowe, wykorzystywane przy realizacji operacji dodawania algebraicznego liczb wielobitowych. Programy syntezy i implementacji układów cyfrowych w układach FPGA wykorzystują tą logikę do tworzenia sumatorów różnych typów i innych układów cyfrowych opartych o sumatory (np. komparatorów, bloków mnożenia, liczników i inne). Blok szybkiego przeniesienia zawiera dwuwejściowy multiplekser MX oraz bramkę XOR (rys. 2.39a). Bramka XOR pozwala na zbudowanie pełnego jednobitowego sumatora w oparciu o jeden blok LUT (ang. Look-up-Table), który realizuje funkcję logiczną (A_i xor B_i). Wykorzystanie bloku szybkiego przeniesienia pozwala znacząco zmniejszyć opóźnienie sumatorów wielobitowych realizowanych w układach FPGA.

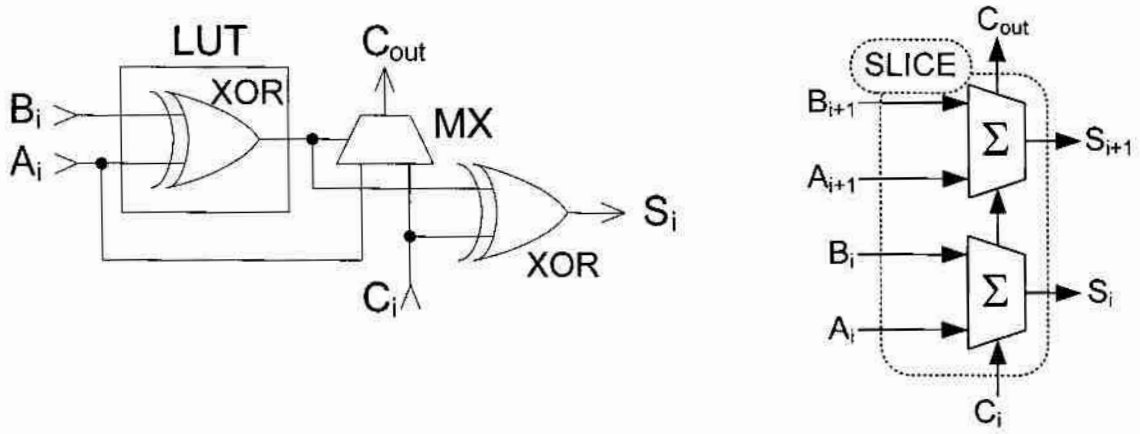
Zasada formowania sygnału przeniesienia wyjściowego oraz struktura wewnętrzna układu przeniesienia są pokazane na rys. 2.39a. Można zauważyć (tab. 2.31), że jeżeli bity A_i i B_i argumentów dodawania mają jednakową wartość, to stan przeniesienia wyjściowego C_{out} jest równy wartości A_i niezależnie od stanu wejścia C_i. Natomiast jeżeli wartości A_i i B_i są różne, to przeniesienie C_{out} jest równe wartości przeniesienia wejściowego C_i. W związku z tym, wynik (A_i xor B_i) otrzymywany z wyjścia LUT steruje multiplekserem MX w taki

sposób, żeby na wyjście C_{out} były podawane wartości A_i lub C_i. Poza tym, sygnał z wyjścia LUT uczestniczy w formowaniu sygnału sumy argumentów S_i.

Tab. 2.31. Tabela prawdy funkcji wykorzystanych w blokach szybkiego przeniesienia w układach FPGA Xilinx

A_i	B_i	C_i	A_i xor B_i	S	C_{out}	Funkcja
0	0	0	0	0	0	$A_i = B_i \rightarrow C_{out} = A_i$
0	0	1	0	1	0	$A_i = B_i \rightarrow C_{out} = A_i$
0	1	0	1	1	0	$A_i \neq B_i \rightarrow C_{out} = Ci$
0	1	1	1	0	1	$A_i \neq B_i \rightarrow C_{out} = Ci$
1	0	0	1	1	0	$A_i \neq B_i \rightarrow C_{out} = Ci$
1	0	1	1	0	1	$A_i \neq B_i \rightarrow C_{out} = Ci$
1	1	0	0	0	1	$A_i = B_i \rightarrow C_{out} = A_i$
1	1	1	0	1	1	$A_i = B_i \rightarrow C_{out} = A_i$

a) b)



Rys. 2.39. Układ szybkiego przeniesienia układu FPGA (a) i realizacja pełnego sumatora dwubitowego (b)

Układ SLICE zawiera dwa bloki LUT, można zatem zbudować w nim pełen sumator dwubitowy, którego schemat ogólny przedstawiony jest na rys. 2.39b. Sumatory n-bitowe tworzy się przez szeregowe łączenie takich dwubitowych sumatorów.

W obu układach LUT są zapisane wartości funkcji logicznej realizującej operacje *XOR* (A_i xor B_i). Wynik podawany jest na bramkę XOR, gdzie obliczany jest odpowiedni bit sumy S_i. Przeniesienie jest uzyskiwane w sposób opisany powyżej i podawane jest na wejście C_i kolejnego dwubitowego sumatora. Bity A_i i A_{i+1} argumentów dodawania podawane są odpowiednio na wejścia $F1$ i $G1$, bity B_i i B_{i+1} - na wejścia $F2$ i $G2$. Suma otrzymywana jest na wyjściach X (bit S_i) i Y (bit S_{i+1}), przeniesienie wyjściowe jest wydawane na wyjście układu Slice *COUT*. Na rys. 2.40 przedstawiono diagramy Veitcha dla funkcji S i C_{out} układu szybkiego przeniesienia.

a)

C_i \ A_i				
	1	0	0	1
	0	1	1	0

S

A_i xor B_i

b)

C_i \ A_i				
	1	1	1	0
	1	0	0	0

C_{out}

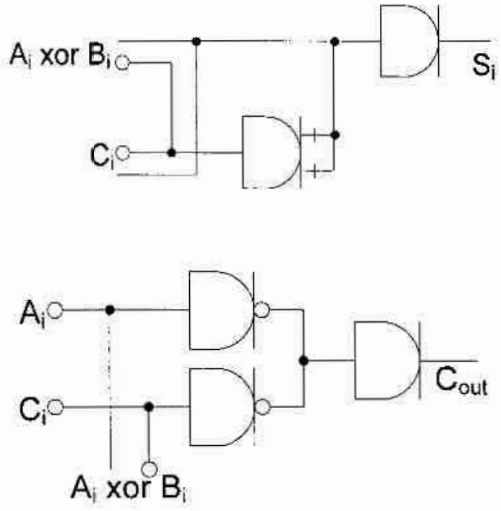
A_i xor B_i

Rys. 2.40. Diagramy Veitcha funkcji S (a) i C_{out} (b) stosowanych w układzie szybkiego przeniesienia FPGA Spartan II

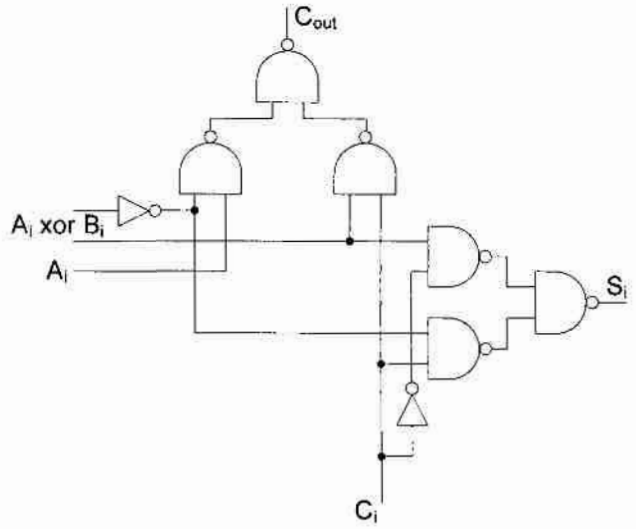
$$S = \overline{\overline{A_i xor B_i} + C_i} + \overline{A_i xor B_i + \overline{C_i}} \tag{2.75}$$

$$C_{out} = \overline{\overline{A_i xor B_i} + \overline{C_i}} + \overline{A_i xor B_i + \overline{A_i}} \tag{2.76}$$

a)



b)

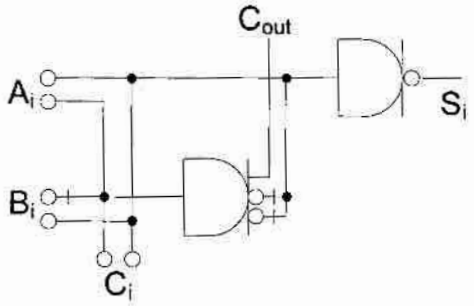


Rys. 2.41. Układ szybkiego przeniesienia: realizacja na bramkach prądowych (a) i napięciowych (b) – jedne z możliwych realizacji

Opisy (2.75) i (2.76) posłużyły do realizacji układu szybkiego przeniesienia w prototypie prądowym układu SLICE (rys. 2.41a). Układ realizujący powyższe funkcje na klasycznych bramkach CMOS przedstawiony jest na rys. 2.41b.

Ponadto, w celu odciążenia bloku LUT, autor zaproponował zastąpienie układu szybkiego przeniesienia zwykłym sumatorem zbudowanym z bramek prądowych (rys. 2.42). Dzięki temu 2 elementy w standardowym układzie SLICE (multiplekser i bramka XOR) zostały zamienione na 2 bramki prądowe, co pod względem liczby wykorzystanych tranzystorów jest prawie to samo. Jednak układ LUT w układzie prądowym może być równocześnie

wykorzystany do realizacji innych funkcji, a w układzie napięciowym – nie. Z tego powodu przedstawione rozwiązanie zostało wykorzystane w prototypie prądowym układu SLICE.



Rys. 2.42. Układ szybkiego przeniesienia zoptymalizowany w oparciu o *sposób 2*

Warto wspomnieć, że w ramach badań zespołu wykonano projekt układu scalonego ASIC zawierający m.in. uproszczony układ SLICE układu FPGA Spartan II (na bramkach prądowych). Projekt wykonano kierując się regułami procesu technologicznego CMOS o minimalnej długości kanału tranzystora MOS 0,6 mikrometra, o bramce polikrzemowej i trzech warstwach metalizacji AMS 0.6μ firmy Austria Mikro Systeme. Podczas testowania w sposób eksperymentalny został określony najlepszy poziom jedynki logicznej (dla tej technologii i wybranych rozmiarów tranzystorów), który wynosi 30 μA. Przy takim poziomie jedynki logicznej jednowyjściowa bramka pobiera prąd 30 μA od źródła zasilania, i prąd ten rośnie wprost proporcjonalnie ze zwiększeniem liczby wyjść bramki.

3. Projektowanie prądowych jednostek operacyjnych działających w arytmetykach *N*-wartościowej, *modulo N* i resztowej

W rozdziale 1.2.3 rozprawy zaznaczono, że najważniejszymi zaletami stosowania w systemach jednoukładowych, a szczególnie w ich jednostkach operacyjnych, arytmetyki wielowartościowej z podstawą N (N-wartościowej) są znaczne zmniejszenie liczby połączeń wewnętrznych w systemie oraz skrócenie czasu wykonywania operacji w szeregowych jednostkach operacyjnych, np. w szeregowych sumatorach, blokach mnożenia itd. Natomiast zaletami wykorzystania systemu RNS opartego o arytmetykę *modulo* N jest brak przeniesienia między poszczególnymi cyframi w liczbach oraz łatwość i szybkość wykonania operacji dodawania, odejmowania i mnożenia.

Z powodu tego, że układy prądowe działają (w przypadku ogólnym) w logice wielowartościowej MVL, a algebra bramek prądowych jest w przypadku ogólnym algebrą wielowartościową, autor niniejszej rozprawy podjął w tym podrozdziale próbę wykazania zalet stosowania układów prądowych w układach cyfrowych przeznaczonych do działania w arytmetyce resztowej RNS i N-wartościowej poprzez optymalizację starszych i opracowanie nowych, bardziej skomplikowanych projektów operacyjnych jednostek prądowych oraz ich porównanie pod względem złożoności sprzętowej z odpowiednimi układami zbudowanymi z klasycznych bramek CMOS. Ponieważ najbardziej rozpowszechnioną w praktycznych zastosowaniach arytmetyką (wśród wyżej wymienionych) jest arytmetyka RNS, autor niżej podaje kilka podstawowych wiadomości o tej arytmetyce.

Arytmetyka resztowa RNS jest wykorzystywana w systemach cyfrowego przetwarzania sygnałów czasu rzeczywistego ze względu na możliwość szybkiego wykonywania podstawowych operacji arytmetycznych (dodawanie, odejmowanie i mnożenie) bez uwzględniania przeniesienia między poszczególnymi cyframi w liczbach [1, 35, 42]. Ponadto, ze względu na separację poszczególnych cyfr systemu resztowego, jeżeli błąd zdarzy się w jednej cyfrze, to nie jest on propagowany na inne pozycje podczas kolejnych operacji. Arytmetyka RNS jest oparta o arytmetykę *modulo* N (bardziej szczegółowo oparta jest o kilka arytmetyk *modulo* N o różnych wartościach N). Jako przykład w tab. 3.1 przedstawiono zapis liczb dziesiętnych od 0 do 9 w różnych arytmetykach *modulo* N.

Tab. 3.1. Reprezentacja przykładowych liczb dziesiętnych w arytmetykach *modulo N*

dec	arytmetyka *modulo N*			
	N=3	*N*=5	*N*=7	*N*=8
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	0	3	3	3
4	1	4	4	4
5	2	0	5	5
6	0	1	6	6
7	1	2	0	7
8	2	3	1	0
9	0	4	2	1

W arytmetyce *modulo N* zapis liczby jest uzależniony od podstawy (modułu) *N* systemu wielowartościowego i pozwala na przedstawienie w sposób jednoznaczny *N* liczb. Zapis większych od podstawy systemu *N* liczb sprowadza się do obliczenia reszty z dzielenia tej liczby przez jego podstawę. W rzeczywistych zastosowaniach arytmetyka *modulo N* wykorzystywana jest jako składowa cześć systemu resztowego.

System resztowy (RNS) można zdefiniować jako zbiór liczb całkowitych $\{m_1, m_2, ..., m_q\}$, które są względnie pierwsze (najmniejsza wspólna wielokrotność jest ich iloczynem). Liczba naturalna X w zakresie (0, M-1) w systemach RNS jest kodowana przez q cyfr (reszt) $\{r_1, r_2, ..., r_q\}$, gdzie

$$r_i = x(\mathrm{mod}\, m_i) \text{ ..lub } r_i = |X|_{m_i}, \quad i = 1,2,...,q, \tag{3.1}$$

i zakresu M jest równy

$$M = \prod_{i=1}^{q} m_i \tag{3.2}$$

W tab. 3.2 przedstawione są liczby w różnych systemach resztowych (tj. w systemach RNS o różnych zbiorach modułów N) oraz odpowiadające im wartości zapisane w systemie dziesiętnym.

Przykładowo liczba 7_{10} zapisana w 3-cyfrowym (q=3) systemie resztowym RNS($N_1|N_2|N_3$)=RNS(7|5|3) będzie oznaczona jako $(0|2|1)_{RNS(7|5|3)}$. Jednoznacznie dla tego przykładu możemy zapisać liczby z zakresu od 0 do 104 (tj. M=3·5·7). Istotnym jest dobór poszczególnych modułów w systemie resztowym.

Powinien on spełniać kilka warunków:

- obejmować swoim zakresem cały zbiór liczb, na którym operuje projektowany system;
- podstawy powinny spełniać warunek: $NWW=N_1 \cdot N_2 \cdot \ldots \cdot N_q$ (najmniejsza wspólna wielokrotność każdej pary liczb reprezentujących podstawy systemów jest równa iloczynowi tych liczb);
- moduły powinny być o zbliżonej wartości - największy moduł decyduje o szybkości systemu, a ilość i wielkość modułów decyduje o złożoności systemu.

W tab. 3.2 przedstawione są zakresy liczb, na jakich można operować za pomocą wybranego systemu resztowego (zakres systemu resztowego).

Tab. 3.2. Reprezentacja przykładowych liczb dziesiętnych w różnych systemach RNS

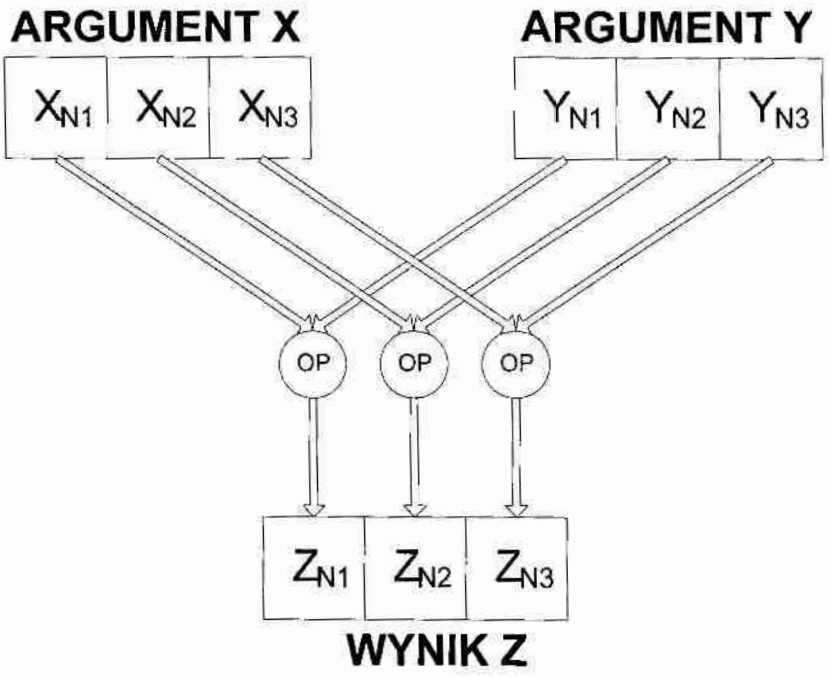
	Przykłady systemów resztowych trzycyfrowych			
dec	{ 3, 5, 7 }	{ 3, 7, 11 }	{ 5, 7, 13 }	{13, 14, 15}
-3	024	0 4 7	2 4 9	10 11 12
-2	135	1 5 9	3 5 10	11 12 13
-1	246	2 6 10	4 6 12	12 13 14
0	000	000	000	000
1	111	111	111	111
2	222	222	222	222
3	033	033	333	333
4	144	144	444	444
5	205	255	055	555
6	016	066	166	666
7	120	107	207	777
8	231	218	318	888
9	042	029	429	999
zakres systemu resztowego	3·5·7→ 0-104	3·7·11→ 0-230	5·7·13→ 0-454	13·14·15→ 0-2730
liczba bitów *M*	8	9	10	12
liczba binarna *M*-bit	256	512	1024	4096

Współczesne układy cyfrowe pracujące w systemach resztowych najczęściej zapisują poszczególne cyfry systemu resztowego w postaci liczb binarnych (przykładowo arytmetyka *modulo* 5 reprezentowana jest przez liczbę binarną 3-bitową, a ogólnie arytmetyka *modulo N* reprezentowana jest przez liczbę binarną $\lceil \log_2 N \rceil$ -bitową) [42]. W tab. 3.2 wskazano liczbę

bitów M jaka potrzebna jest do zapisania liczby wielowartościowej właśnie tym sposobem (tj. wykorzystując kodowanie binarne). Dla porównania obliczono również jaką maksymalną wartość można uzyskać tworząc liczbę binarną M-bitową. Jak widać większy zakres można otrzymać dla liczby binarne, co oznacza, że system RNS kodowany binarnie jest zbyteczny (nadmiarowy) w stosunku do klasycznego systemu binarnego. Należy zaznaczyć, że w tab. 3.2 umieszczone są również liczby ujemne, które obliczane są zgodnie z wyrażeniem (3.3):

$$(-X)_{Ni} = (N_i - X)_{Ni} \tag{3.3}$$

Ze względu na brak przeniesienia przy wykonywaniu podstawowych operacji arytmetycznych, wykonywane one mogą być równolegle na wszystkich cyfrach systemu, niezależnie od siebie, nie potrzeba tworzyć bloków formujących przeniesienie i operacje wykonywane są na małych cyfrach. Na rys. 3.1 pokazany jest ogólny schemat wykonywania podstawowych operacji (dodawanie, odejmowanie, mnożenie) w systemach resztowych, a przykłady takich operacji przedstawiono w tab. 3.3 i tab. 3.4.



Rys. 3.1. Ogólny schemat wykonywania podstawowych operacji w systemach resztowych

W związku z tym autor w następnych podrozdziałach rozprawy podejmuje próbę opracowania kilku różnego rodzaju jednostek operacyjnych (m.in. jednocyfrowych sumatorów N-wartościowych i *modulo N*, wielooperandowych sumatorów resztowych, bloków mnożących przez wartość stałą). Ponadto, ze względu na powszechność stosowania binarnego systemu liczbowego (BIN) w systemach komputerowych, ważnymi elementami

jednostek przetwarzających RNS są konwertery liczb pomiędzy systemami BIN i RNS i na odwrót.

Tab. 3.3. Przykład wykonania operacji dodawania w różnych systemach RNS

dec		$RNS_{(7,5,3)}$			$RNS_{(11,7,3)}$		
x	*y*	*x*	*y*	*suma (Z)*	*x*	*y*	*suma (Z)*
1	4	111	441	502	111	441	552
3	2	330	222	502	330	222	552
4	5	441	502	240	441	552	920
6	3	610	330	240	660	330	920

Tab. 3.4. Przykład wykonania operacji mnożenia w różnych systemach RNS

dec		$RNS_{(7,5,3)}$			$RNS_{(11,7,3)}$		
x	*y*	*x*	*y*	*suma (Z)*	*x*	*y*	*suma (Z)*
1	4	111	441	441	111	441	441
2	2	222	222	441	222	222	441
2	9	222	240	430	222	920	740
6	3	610	330	430	660	330	740

Oba typy konwerterów BIN-RNS oraz RNS-BIN oparte są najczęściej o bloki wieloargumentowych sumatorów resztowych MOMA [30], przy czym konwertery RNS-BIN budowane najczęściej w oparciu o Chińskie twierdzenie o resztach (ang. CRT – Chinese remainder theorem) lub algorytmu o zmiennej podstawie (ang. MRC – mixed-radix conversion). Projektowaniem takich właśnie konwerterów autor zajął się w niniejszym rozdziale rozprawy.

3.1. Projektowanie sumatorów jednocyfrowych

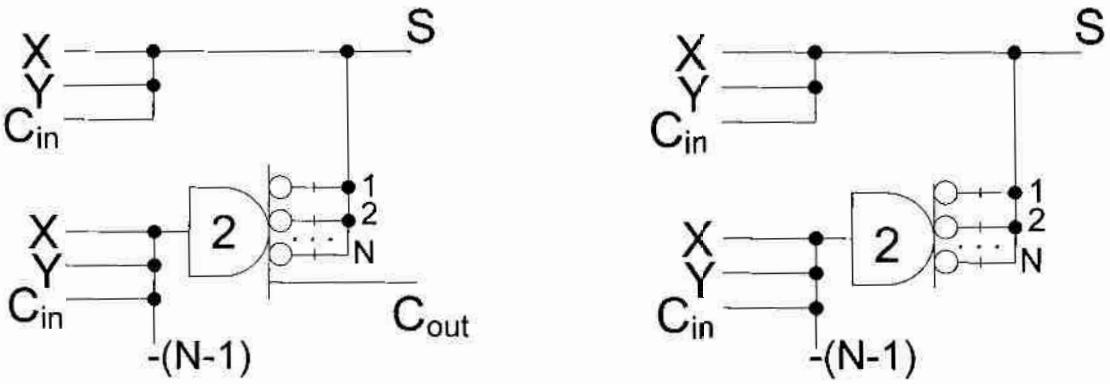
Jednobitowy sumator *N*-wartościowy posiada zawsze trzy wejścia i dwa wyjścia. Na dwa wejścia *X* i *Y* podaje się wartości od „0" do „*N*-1", a wejście C_{IN} odpowiada za podanie jednobitowej wartości przeniesienia z poprzedniego bloku (wartości logiczne „0" lub „1"). Na wyjściu otrzymujemy dwie wartości odpowiadające sumie *S* (wyjście *N*-wartościowe) i przeniesieniu wyjściowemu C_{OUT} (wyjście jednobitowe). Wyrażenia na sumę i przeniesienie dla sumatora *N*-wartościowego przedstawiają odpowiednio wyrażenia (3.4) i (3.5):

$$S = (X + Y + C_{IN}) \bmod N = \begin{cases} \Sigma, & if\ \Sigma < N \\ \Sigma - N, & if\ \Sigma \geq N \end{cases} \tag{3.4}$$

$$C_{OUT} = \begin{cases} 0, & if\ \Sigma < N \\ 1, & if\ \Sigma \geq N \end{cases} \tag{3.5}$$

W pracy [19] zaproponowano projekt ogólnej struktury sumatora dla arytmetyki N-wartościowej, który przedstawiony jest na rys. 3.2a. Opracowano również strukturę sumatora dla arytmetyki *modulo N* (rys. 3.2b).

a) b)



Rys. 3.2. Ogólne struktury sumatorów: N-wartościowego (a), *modulo N* (b)

Tab. 3.5 prezentuje tabelę prawdy sumatora pracującego w logice MVL o podstawie N=3, gdzie przez Σ zaznaczono sumą algebraiczną wartości argumentów wejściowych w systemie dziesiętnym.

W rozdziale 2.1.1 opisano funkcje wzorcowe typu T, której jedna z postaci wartości logiczne „1" na wyjściu uzyskiwała wówczas, gdy suma algebraiczna argumentów wejściowych osiągnęła pewną założoną wartość – równą podstawie systemu N (rys. 2.5a). Określono również podstawowy wzór jakim można opisać taką funkcję (2.10). Funkcję tą wykorzystano do budowy sumatora jednobitowego pracującego w logice binarnej. Autor proponuje wykorzystać powyższe założenie do opracowania funkcji wzorcowej typu T dla arytmetyki N-wartościowej, by następnie użyć jej do budowy sumatora jednobitowego dla arytmetyki N-wartościowej. Wyrażenie (3.6) przedstawia opis zaproponowanej funkcji TBlok(3) dla logiki MVL o podstawie N=3.

$$F_{TB} = X + \hat{\hat{Y}} + C_{IN} = C_{OUT} \tag{3.6}$$

W tab. 35 pokazano tabelę prawdy dla niepełnej funkcji TBlok(3) (niepełna, bo wejście przeniesienia C_{IN} jest binarne).

Tabela prawdy funkcji F_{TB} jest identyczna jak tabela prawdy dla funkcji C_{OUT} dla sumatora jednocyfrowego, zatem można użyć jej do realizacji przeniesienia. Należy zauważyć, że

funkcja przeniesienia używa operacji anty-podwójnej-inwersji, jednak musi ona być wykonana na jednym z sygnałów wielowartościowych (X lub Y), a nie na binarnym wejściu przeniesienia C_{IN}.

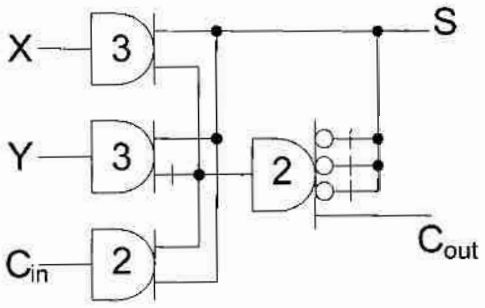
Tab. 3.5. Tabela prawdy sumatora MVL o podstawie N=3 oraz kilku funkcji bazowych

C_{IN}	X	Y	S	C_{OUT}	Σ	$X+\hat{\bar{Y}}+C_{IN}$	$\overline{\overline{X+\hat{\bar{Y}}+C_{IN}}}$	$F3$
0	0	0	0	0	0	-2	0	0
0	0	1	1	0	1	-1	0	0
0	0	2	2	0	2	0	0	0
0	1	0	1	0	1	-1	0	0
0	1	1	2	0	2	0	0	0
0	1	2	0	1	3	1	1	-3
0	2	0	2	0	2	0	0	0
0	2	1	0	1	3	1	1	-3
0	2	2	1	1	4	2	1	-3
1	0	0	1	0	1	-1	0	0
1	0	1	2	0	2	0	0	0
1	0	2	0	1	3	1	1	-3
1	1	0	2	0	2	0	0	0
1	1	1	0	1	3	1	1	-3
1	1	2	1	1	4	2	1	-3
1	2	0	0	1	3	1	1	-3
1	2	1	1	1	4	2	1	-3
1	2	2	2	1	5	3	1	-3

Drugie spostrzeżenie dotyczy realizacji funkcji sumy S dla powyższego sumatora. Jeżeli od wartości sumy argumentów wejściowych odejmiemy funkcję $F3$ (przedstawioną w tab. 3.5), to otrzymamy funkcję sumy S dla sumatora wielowartościowego o podstawie N=3. Funkcja $F3$ posiada wartości „-3" dla tych samych kombinacji wartości argumentów wejściowych dla których na wyjściu C_{OUT} pojawia się wartość „1". Można zatem wykorzystać funkcję C_{OUT} do realizacji funkcji sumy S, podobnie jak to miało miejsce dla sumatorów binarnych. Wyrażenie przedstawiające funkcję sumy S przedstawia (3.7).

$$S = X + Y + C_{IN} - 3 \cdot \hat{C}_{OUT} = X + Y + C_{IN} + 3 \cdot \hat{\overline{X + \hat{\bar{Y}} + C_{IN}}} \tag{3.7}$$

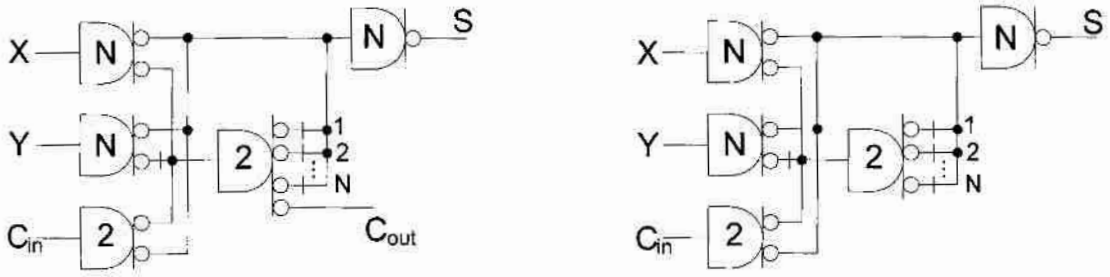
Na rys. 3.3 przedstawiony jest schemat sumatora dla logiki MVL o podstawie N=3 zrealizowany wg powyższych wytycznych.



Rys. 3.3. Schemat sumatora dla logiki MVL o podstawie N=3

Autor proponuje nowe ogólne schematy dla sumatorów jednocyfrowych dla logiki MVL o podstawie N (rys. 3.4a) i dla sumatorów *modulo N* (rys. 3.4b).

a) b)



Rys. 3.4. Zoptymalizowane przez autora ogólne struktury sumatorów: MVL o podstawie N (a), *modulo N* (b)

Sumator *modulo N* różni się od sumatora MVL jedynie brakiem wyjścia przeniesienia C_{OUT}. Przedstawione schematy różnią się od schematu dla sumatora MVL o podstawie N=3 z rys. 3.3. W schematach ogólnych użyto bramek z wyjściami typu inwerter i anty-inwerter, które wymagają mniej tranzystorów przy realizacji praktycznej w algebrze bramek prądowych. Wyrażenie (3.8) przedstawia ogólne wyrażenie na funkcję sumy S (dla obu sumatorów), a wyrażenie (3.9) ogólny wzór na funkcję przeniesienia C_{OUT} (dla sumatora MVL o podstawie N).

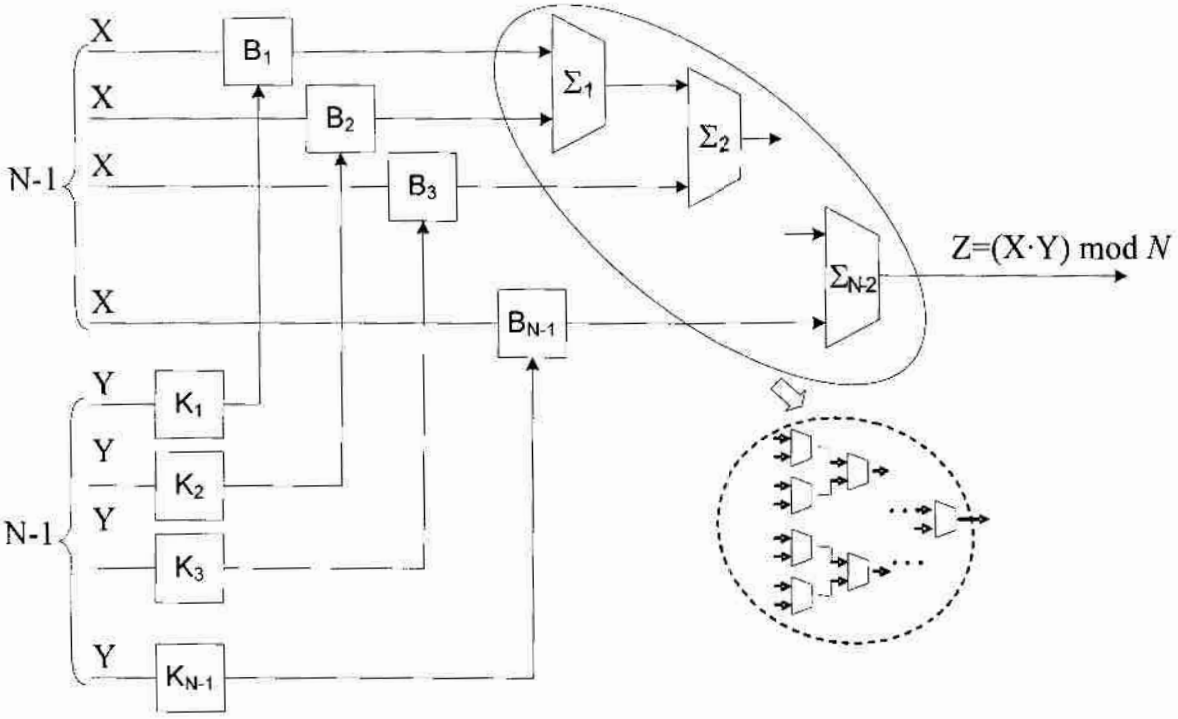
$$C_{OUT} = \overline{\overline{X} + \hat{Y} + \overline{C}_{IN}} \tag{3.8}$$

$$S = \overline{X} + \overline{Y} + \overline{C}_{IN} + N \cdot \overline{\overline{X} + \hat{Y} + \overline{C}_{IN}} \tag{3.9}$$

3.2. Opracowanie projektu układu mnożącego liczbę przez wartość stałą

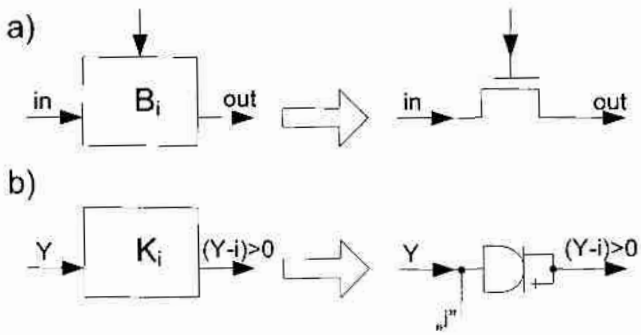
Klasyczne napięciowe układy mnożące w logice wielowartościowej projektowane są w oparciu o bloki ROM, głównie ze względu na zmniejszenie zajmowanej powierzchni w porównaniu z układami opartymi na sumatorach. Układ mnożący *modulo N* posiada dwa

wejścia X i Y, na które podaje się wartości od „0" do „N-1" i jedno wyjście Z (również N-wartościowe). Na rys. 3.5 przedstawiono ogólną strukturę układu mnożącego w arytmetyce *modulo N*. Dokładny opis znajduje się w [19].



Rys. 3.5. Ogólna struktura układu mnożącego *modulo N*

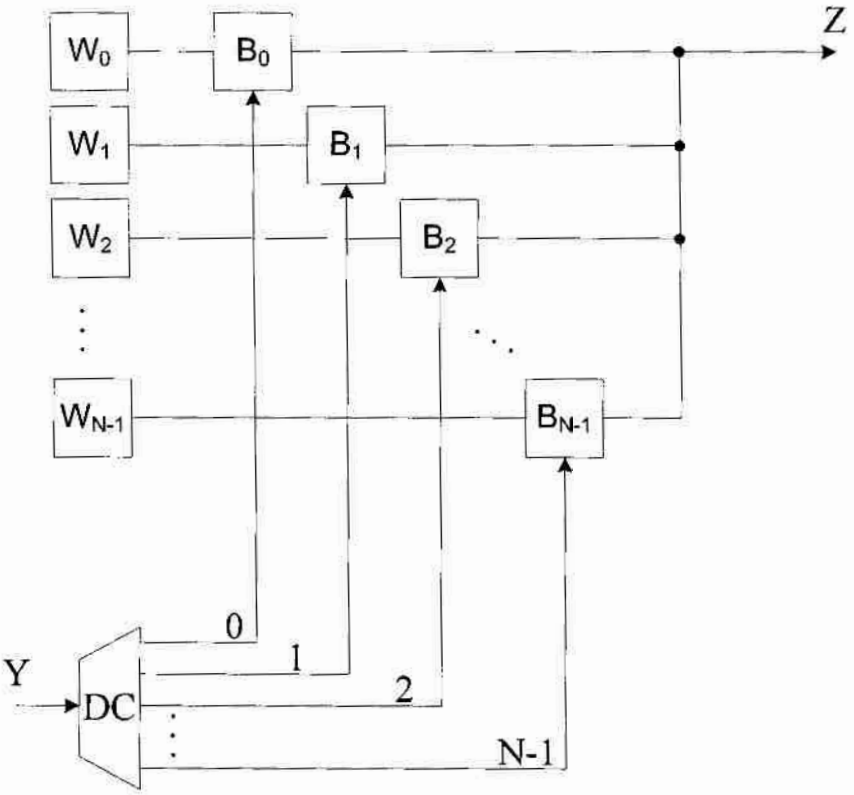
Struktura zawiera (N-2) sumatorów *modulo N* ($\sum_i$), których każde i-te wyjście podłączone jest do (i+1) wyjścia kolejnego sumatora. Opcjonalnie w celu zmniejszenia czasu opóźnienia układu można zastosować strukturę połączeń sumatorów z prawego dolnego rogu rys. 3.5. Dodatkowo struktura zawiera N-1 komparatorów K_i i buforów B_i. Każdy i-ty bufor przenosi na wejście sumatora wartość X wtedy gdy i-ty komparator jest aktywny. Komparator K_i staje się aktywny gdy wartość Y jest większa od i. Budowa bufora i komparatora w technologii bramek prądowych przedstawiona jest na rys. 3.6.



Rys. 3.6. Przykładowa realizacja bufora B_i (a) i komparatora K_i (b) w technologii prądowej

Autor na podstawie układu mnożącego opisanego powyżej proponuję uproszczoną strukturę układu mnożącego przez stałą dla arytmetyki *modulo N*.

Klasyczny układ mnożący przez stałą posiada jedno wejście N-wartościowe Y oraz jedno wyjście również N-wartościowe. Liczba wejściowa składa się z $L=\lceil \log_2 N \rceil$ bitów (zapis poszczególnych cyfr arytmetyki *modulo N* w postaci liczb binarnych). A blok ROM realizujący tą operacje musi zawierać $2^L \cdot L$ komórek. Przykładowo dla $N=10$ układ ROM powinien zawierać 16 czterobitowych komórek. Dla technologii bramek prądowych podobny układ ROM będzie wymagał użycia jedynie 10 komórek (N komórek w przypadku ogólnym). Również oczywistym jest, że cała struktura będzie zawierała mniej połączeń wynikających choćby z zastosowania jednego połączenia do realizacji Y zamiast wektora w tradycyjnych układach napięciowych. Ogólna struktura układu mnożącego przez stałą dla arytmetyki *modulo N* przedstawia rys. 3.7.



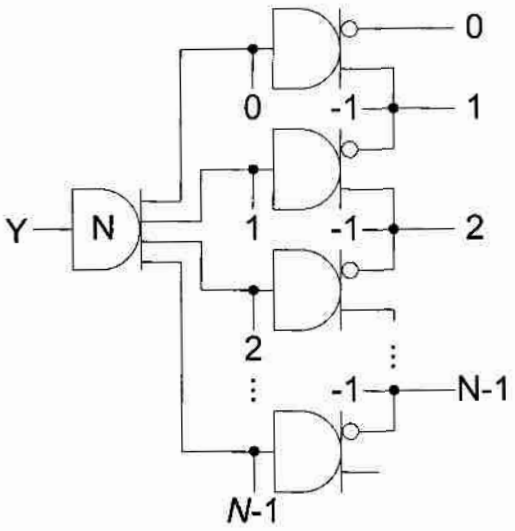
Rys. 3.7. Ogólna struktura układu mnożącego przez stałą dla arytmetyki *modulo N*

Struktura ta zawiera N komórek ROM i N buforów, których wyjścia połączone są w jeden punkt. Stałe W_i zawierają wartości opisane za pomocą wyrażenia (3.10):

$$W_i = \underbrace{(X + X + \ldots + X)}_{i} \bmod N \tag{3.10}$$

gdzie i może przyjmować wartości od „0" do „N-1".

Dekoder, którego struktura w technologii bramek prądowych pokazana jest na rys. 3.8 powoduje, że tylko jedna stała W_i jest przekazywana na wyjście Z układu. W zależności od wartości Y na wejściu dekodera aktywne jest tylko jedno z wyjść 0 –N-1.



Rys. 3.8. Ogólna struktura dekodera *DC*

Na podstawie projektu układu mnożącego przez stałą w arytmetyce resztowej można zaprojektować podobny układ działający w logice wielowartościowej o podstawie N. Dla układu tego dojdzie jedno dodatkowe wejście C_{MI} i jedno wyjście C_{MO}, a wyrażenie opisujące wyjście Z i C_{MO} przedstawia (3.11) i (3.12).

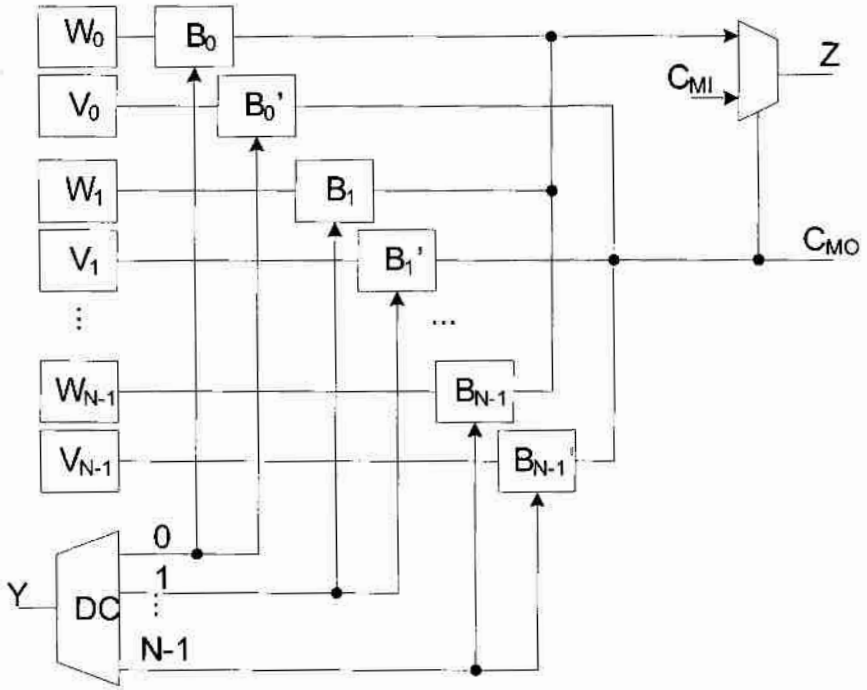
$$Z = ((X \cdot Y) + C_{MI}) \bmod N \tag{3.11}$$

$$C_{MO} = ((X \cdot Y) + C_{MI}) \operatorname{div} N \tag{3.12}$$

gdzie *div* oznacza operację dzielenia całkowitego.

Struktura w przypadku układu napięciowego wymaga 2 razy większego bloku ROM (w porównaniu do układu mnożącego przez stałą dla arytmetyki *modulo N*), w którym każda komórka zawiera $2 \cdot L$ bitów i dodatkowo dwa sumatory do korekcji wyniku Z i C_{MO}. Ta sama struktura w technologii bramek prądowych przedstawiona jest na rys. 3.9. Wymaga ona zastosowania bloku ROM powiększonego o $2 \cdot N$ komórek i dodatkowo jednego sumatora. Struktura ta zawiera podobny dekoder, który uaktywnia jednocześnie dwa bufory, jeden przekazujący wartość sumy, a drugi wartość przeniesienia. Zawartość poszczególnych komórek ROM oblicza się w oparciu o wyrażenia (3.10) dla sumy W_i i (3.13) dla przeniesienia V_i:

$$V_i = \underbrace{(X + X + \ldots + X)}_{i} \, div \, N \tag{3.13}$$



Rys. 3.9. Ogólna struktura układu mnożącego przez stałą dla logiki wielowartościowej o podstawie N

Przedstawiona struktura prądowa jest prostsza niż odpowiadająca jej struktura napięciowa pod względem liczby połączeń oraz ilości tranzystorów potrzebnych do realizacji układów.

3.3. Projektowanie konwertera liczb z systemu binarnego do systemu resztowego

Konwertery z systemu binarnego do systemu resztowego (BIN-RNS) przekształcają a-bitową liczbę $X=[x_{a-1},\ldots,x_1,x_0]$, gdzie $a=\lceil \log_2 M \rceil$, w odpowiednią q-cyfrową liczbę RNS i zazwyczaj składają się z q bloków o identycznej wewnętrznej strukturze. Każdy z tych bloków (zwanym generatorem resztowym albo generatorem *modulo* m_i) realizuje wyrażenie (3.1) dla jednej pozycji (cyfry) i = 1, 2,..., q systemu RNS. Klasyczny algorytm obliczający i-tą cyfrę resztową r_i oparty jest na wyrażeniu (3.1),

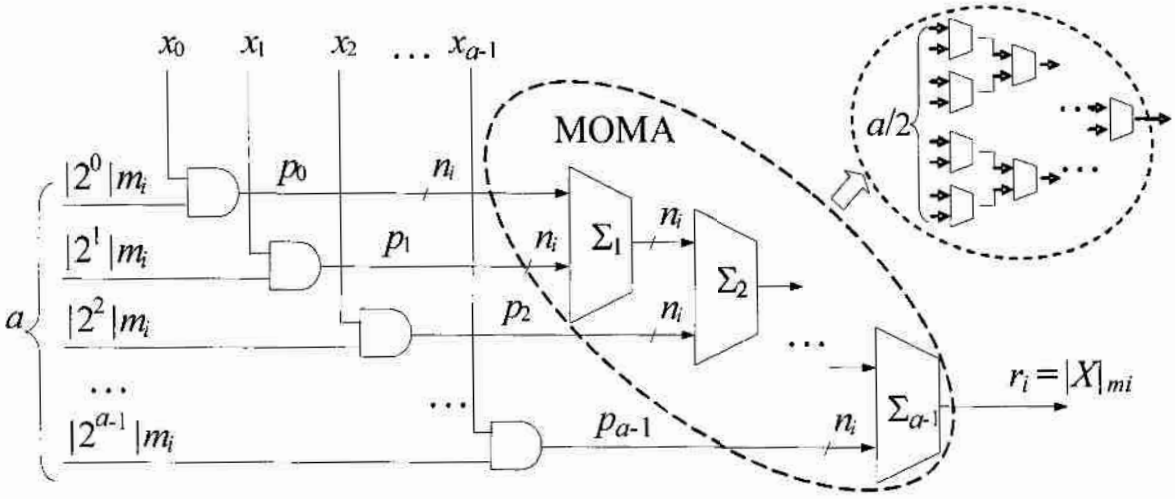
$$r_i = |X|_{m_i} = \left| \sum_{j=0}^{a-1} \left(2^j \cdot x_j\right) \right|_{m_i} \tag{3.14}$$

które może być uproszczone do następującej formuły:

$$r_i = |X|_{m_i} = \left| \sum_{j=0}^{a-1} \left(2^j \cdot x_j\right) \right|_{m_i} = \left| \sum_{j=0}^{a-1} \left(\left|2^j\right|_{m_i} \cdot x_j \right) \right|_{m_i} = \left| \sum_{j=0}^{a-1} \left(p_j\right) \right|_{m_i} \tag{3.15}$$

Ponieważ wartości $\left|2^j\right|_{m_i}$ są stałe, mogą być przechowane w pamięci ROM. $|X|_{m_i}$ mogą być

obliczone przez a-argumentowy n_i-bitowy sumator *modulo* m_i (gdzie $n_i = \lceil \log_2 m_i \rceil$). Taki wieloargumentowy sumator *resztowy* (MOMA) może być zrealizowany za pomocą (a-1) 2-argumentowych n_i-bitowych sumatorów *modulo* m_i, połączonych szeregowo lub za pomocą drzewiastej $\lceil \log_2(a\text{-}1) \rceil$- poziomowej struktury. Obie wspomniane realizacje MOMA są przedstawione na rys. 3.10, który pokazuje ogólną strukturę generatora *modulo* m_i.



Rys. 3.10. Ogólna struktura generatora *modulo* m_i opartego o ROM i MOMA

Logiczne właściwości technologii prądowej pozwalają na zredukowanie liczby połączeń w prądowej wersji generatora modulo m_i, ponieważ każdy dwu-argumentowy n_i-bitowy sumator modulo m_i w MOMA z rys. 3.10 (który zawiera $2 \cdot n_i$ wejść i n_i wyjść), może być zastąpiony przez odpowiedni sumator prądowy, który ma tylko dwa wejścia i jedno wyjście. Taka redukcja jest możliwa dla małych wartości $m_i \leq 12$. W innym przypadku (m_i>12), reszty $r_i = |X|_{m_i}$ powinny być przedstawione przez dwie lub więcej reszt odpowiadających nowym podstawom $\{b_1, b_2, \ldots, b_g\}$, gdzie $b_1 \cdot b_2 \cdot \ldots \cdot b_g \geq m_i$, tak aby

$$r_f^* = |r_i|_{b_f}, \quad f = 1,2,\ldots,g \tag{3.16}$$

Na przykład, dla m_i = 31, X = 102 i r_i = 9, wartości zmiennych g, b_f i r_f^* mogą być następujące: g = 2, b_1 = 5, b_2 = 7, $b_1 \cdot b_2$ = 35 ≥ 31, i stąd $r_1^* = 4$, $r_2^* = 2$. Wartość g = 2 jest typową dla praktycznych zastosowań generatora resztowego. Z tego powodu liczba połączeń w prądowej wersji generatora modulo m_i jest przynajmniej $n_i/2$ razy mniejsza w porównaniu z klasycznym prototypem przedstawionym na rys. 3.10.

Dalsze uproszczenie układu generatora modulo m_i ma na celu zmniejszenie liczby 2-argumentowych sumatorów modulo m_i w prądowej wersji MOMA. Uproszczenie to oparte

jest o realizację dodawania algebraicznego przez połączenie wszystkich argumentów w jeden węzeł (w przypadku technologii bramek prądowych).

Jest oczywistym, że dla danego modułu $m_i > 2^l$, następujące wyrażenie jest prawdziwe:

$$S^* = \left(2^0 + 2^1 + \ldots 2^{l-1}\right) < m_i \tag{3.17}$$

To znaczy, że

$$S^* \geq \sum_{j=0}^{l-1} \left(2^j \cdot x_j\right) \tag{3.18}$$

więc $\left|S^*\right|_{m_i} = S^*$, i

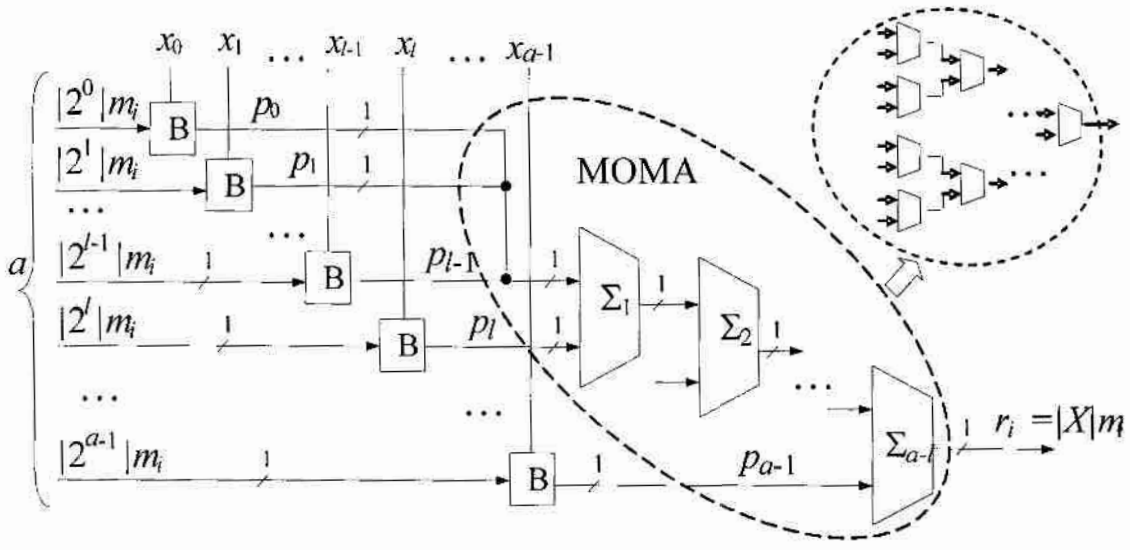
$$S^{**} = \left|\sum_{j=0}^{l-1} \left(p_j\right)\right|_{m_i} = \sum_{j=0}^{l-1} \left(p_j\right) < m_i \tag{3.19}$$

gdzie $p_j = \left|2^j\right|_{m_i} \cdot x_j$.

To pozwala zapisać wyrażenie (3.15) do następującego postaci

$$r_i = \left|X\right|_{m_i} = \left|S^{**} + \left|\sum_{j=l}^{a-1} \left(p_j\right)\right|_{m_i}\right|_{m_i} \tag{3.20}$$

i by obliczyć wartości S^{**} bez użycia sumatorów: pierwsze l argumentów w prądowej wersji MOMA łączy się w jeden węzeł.

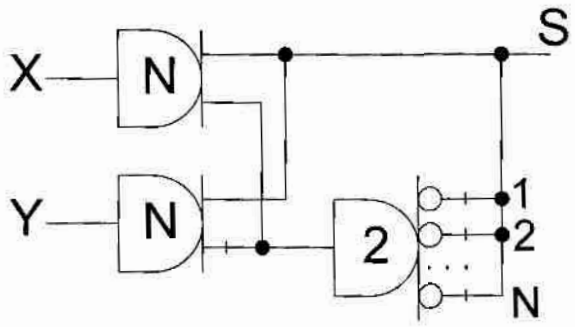


Rys. 3.11. Struktura prądowego generatora *modulo* m_i

W wyniku tej operacji, liczba dwu-argumentowych sumatorów modulo m_i jest zmniejszona od (a-1) do (a-l). Struktura prądowej wersji generatora modulo m_i z uproszczonym MOMA

jest przedstawiona na rys. 3.11.

Struktura dwu-argumentowego sumatora modulo m_i pokazana jest na rys. 3.12. Symbole B oznaczają bufory, które przekazują stałe $\left|2^j\right|_{m_i}$ do wejścia odpowiedniego sumatora, kiedy $x_j = 1$. Przykład realizacji bufora B przedstawiony jest na rys. 3.6b. Powyższa redukcja liczby dwu-argumentowych sumatorów jest możliwa dla wszystkich q generatorów resztowych, które tworzą konwerter BIN-RNS.



Rys. 3.12. Prądowa wersja sumatora *modulo N*

3.4. Projektowanie konwertera liczb z systemu resztowego do systemu binarnego

Konwersja z systemu RNS do systemu binarnego (RNS-BIN) jest bardziej skomplikowana niż w przeciwną stronę [29]. Jednak podstawowe bloki, które służą do realizacji konwerterów RNS-BIN mogą być używane do realizacji innych operacji w systemach RNS.

Ogólnie, konwersja RNS-BIN może być zrealizowana dwoma podstawowymi algorytmami: opartym na chińskim twierdzeniu o resztach (CRT) i o zmiennej podstawie (MRC – mixed-radix conversion) [29]. W pracy tej użyto pierwszego algorytmu do konwersji q-cyfrowej liczby z systemu RNS $[r_1, r_2, \ldots, r_q]$ o modułach $\{m_1, m_2, \ldots, m_q\}$ do odpowiadającej jej a-bitowej liczby binarnej $X=[x_{a-1},\ldots,x_1,x_0]$, gdzie $a=\lceil \log_2 M \rceil$ i M jest reprezentowane przez wyrażenie (3.2).

Zgodnie z chińskim twierdzeniem o resztach, konwerter RNS-BIN oblicza a-bitową binarną liczbę X zgodnie ze wzorem:

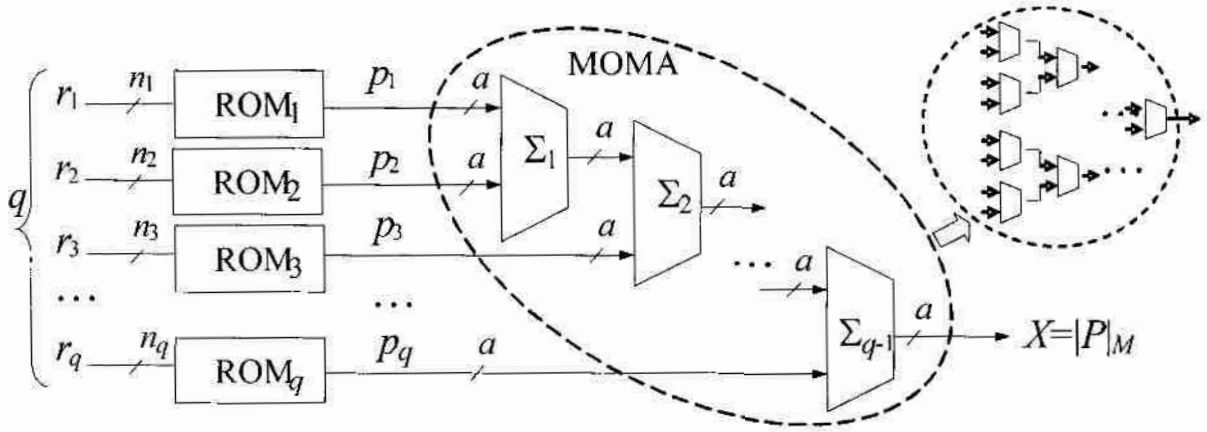
$$X = \left| \sum_{i=1}^{q} \left| r_i \cdot y_i \cdot M_i \right|_M \right|_M \tag{3.21}$$

gdzie $M_i = \frac{M}{m_i}$, i y_i jest odwrotnością multiplikatywną m_i (ang. multiplicative inverse), którą oblicza się ze wzoru $|y_i \cdot M_i|_{m_i} = 1$, ($i$ = 1, 2,..., q). Dla danego modułu $\{m_1, m_2, ..., m_q\}$, wartości $y_i \cdot M_i$ są stałe. Dlatego, konwertery RNS-BIN składać się mogą z q bloków pamięci ROM. I-ty blok ROM zawiera 2^{n_i} a-bitowych rejestrów (gdzie $n_i = \lceil \log_2 m_i \rceil$) i realizuje funkcję $p_i = |r_i \cdot y_i \cdot M_i|_M$. Wyrażenie (3.21) może być przedstawione w następujący sposób:

$$X = \left| \sum_{i=1}^{q} p_i \right|_M = |P|_M \tag{3.22}$$

gdzie $P = \sum_{i=1}^{q} p_i$, i może być obliczone przez q-argumentowy a-bitowy sumator *modulo M*.

Analogicznie do wieloargumentowego sumatora resztowego, który jest używany w konwerterze BIN-RNS, sumator ten może być zrealizowany przez (q-1) 2-argumentowych a-bitowych sumatorów *modulo M*, które są połączone szeregowo albo za pomocą drzewiastej $\lceil \log_2(a-1) \rceil$- poziomowej struktury. Obie wspomniane realizacje sumatorów MOMA są przedstawione na rys. 3.13, gdzie pokazana jest ogólna struktura konwerterów RNS-BIN.



Rys. 3.13. Ogólna struktura konwertera RNS-BIN

Główną wadą tego konwertera jest stosunkowo wysoka złożoność sprzętowa MOMA. W pracach [2, 3, 5, 16, 29-31] zaproponowano kilka ulepszonych struktur wieloargumentowych sumatorów resztowych MOMA. Jednak wspomniane struktury są przeznaczone dla systemów RNS z ograniczonymi wartościami modułów [5], albo są przeznaczone do szybkiej konwersji i oparte są na 3-argumentowych sumatorach CSA (ang. carry-safed adder) [2, 29-31], albo na nadmiarowej binarnej reprezentacji wyników [3]. W związku z tym w rozprawie zaproponowano nową i prostszą strukturę MOMA, która jest oparta na drzewie 2-argumentowych a-bitowych sumatorów CPA (ang. carry propagate

adder). Główna idea nowego MOMA jest oparta na następującym związku:

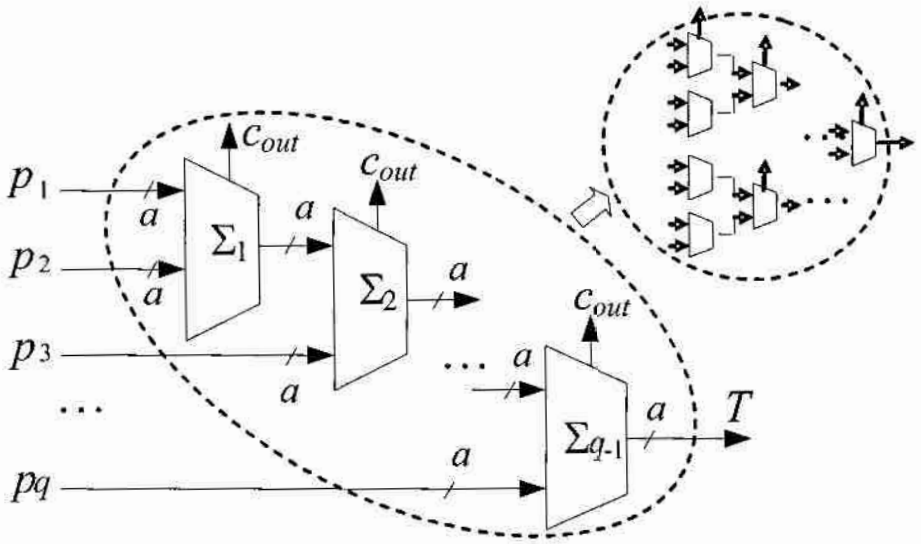
$$P = P - k \cdot 2^{\lceil \log_2 M \rceil} + k \cdot 2^{\lceil \log_2 M \rceil} = \left(\sum_{i=1}^{q} p_i - k \cdot 2^{\lceil \log_2 M \rceil} \right) + k \cdot 2^{\lceil \log_2 M \rceil} = T + S \tag{3.23}$$

gdzie $T = \left(\sum_{i=1}^{q} p_i - k \cdot 2^{\lceil \log_2 M \rceil} \right)$, $S = k \cdot 2^{\lceil \log_2 M \rceil}$, i k = 1, 2,..., (k < q) są takie, że $T < 2^{\lceil \log_2 M \rceil}$.

Dalej wyrażenie (3.22) może być przekształcone do wzoru (3.24):

$$X = |P|_M = |T + S|_M = |T + |S|_M|_M = |P^*|_M \tag{3.24}$$

gdzie $P^* = T + |S|_M$, i stąd $0 \le P^* < 2 \cdot M$. T może być obliczone za pomocą drzewa (q-1) 2-argumentowych a-bitowych sumatorów CPA, w których wszystkie (q-1) wyjścia przeniesienia nie wpływają na ostateczny wynik. Możliwe przykłady takich drzew sumatorów przedstawione są na rys. 3.14.



Rys. 3.14. Przykładowe struktury drzewa sumatorów CPA wykorzystanych w konwerterach RNS-BIN

Do obliczenia funkcji $|S|_M$ może być użyty blok ROM, który posiada (q-1) wejść połączonych z (q-1) wyjściami przeniesienia wszystkich sumatorów CPA. Blok ROM składa się więc z 2^{q-1} a-bitowych rejestrów i zawierają dane przedstawione w tab. 3.6.

Biorąc pod uwagę, że $0 \le P^* < 2 \cdot M$, oczywistym jest, że wyrażenie (3.24) można zapisać następująco:

$$X = \begin{cases} P^*, & P^* < M \\ (P^* - M), & P^* \ge M \end{cases} \tag{3.25}$$

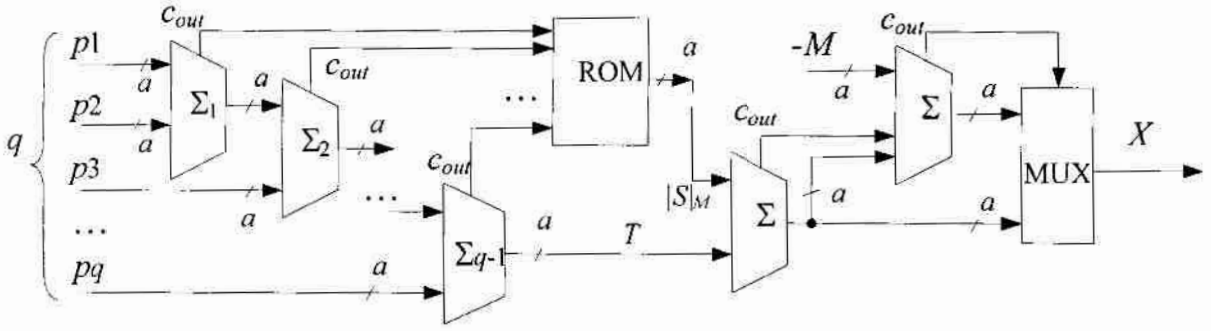
To wyrażenie może być zrealizowane przez jeden 2-argumentowy sumator CPA, który oblicza (P^*+(-M)) w kodzie U2 i multiplekser 2-do-1 sterowany wyjściem przeniesienia tego

sumatora. W wyniku, otrzymujemy ostateczną strukturę zaproponowanego MOMA składającą się z drzewa (q-1) 2-argumentowych a-bitowych sumatorów CPA, jednego bloku ROM, dwóch dodatkowych sumatorów CPA i jednego multipleksera.

Tab. 3.6. Zawartość bloku ROM wykorzystanego w konwerterze RNS-BIN

komórka ROM	kod adres $A_{q-1}...A_2A_1A_0$	$k=\sum_{i=0}^{q-1}A_i$	wartości ROM $\left\lvert k\cdot 2^{\lceil \log_2 M\rceil}\right\rvert_M$
0	0.....0 0 0	0	0
1	0.....0 0 1	1	$\left\lvert 2^{\lceil \log_2 M\rceil}\right\rvert_M$
2	0.....0 1 0	1	$\left\lvert 2^{\lceil \log_2 M\rceil}\right\rvert_M$
3	0.....0 1 1	2	$\left\lvert 2*2^{\lceil \log_2 M\rceil}\right\rvert_M$
4	0.....1 0 0	1	$\left\lvert 2^{\lceil \log_2 M\rceil}\right\rvert_M$
5	0.....1 0 1	2	$\left\lvert 2*2^{\lceil \log_2 M\rceil}\right\rvert_M$
6	0.....1 1 0	2	$\left\lvert 2*2^{\lceil \log_2 M\rceil}\right\rvert_M$
7	0.....1 1 1	3	$\left\lvert 3*2^{\lceil \log_2 M\rceil}\right\rvert_M$
...	...		...
2^{q-1} - 1	1.....1 1 1	q - 1	$\left\lvert (q-1)\cdot 2^{\lceil \log_2 M\rceil}\right\rvert_M$

Nowa struktura MOMA przedstawiona jest na rys. 3.15 i charakteryzuje się mniejszą złożonością sprzętową w porównaniu ze znanymi strukturami MOMA. Blok ROM w zaproponowanym MOMA zawiera 2^{q-1} a - bitowych rejestrów i jest kilka razy mniejszy w porównaniu z blokiem ROM z pracy [29] (dla q < 10). Tab. 3.7 przedstawia liczby komórek w zaproponowanym bloku ROM i w bloku ROM z [29], dla wartości q<10 (wartości q najczęściej stosowane w rzeczywistych systemach RNS).



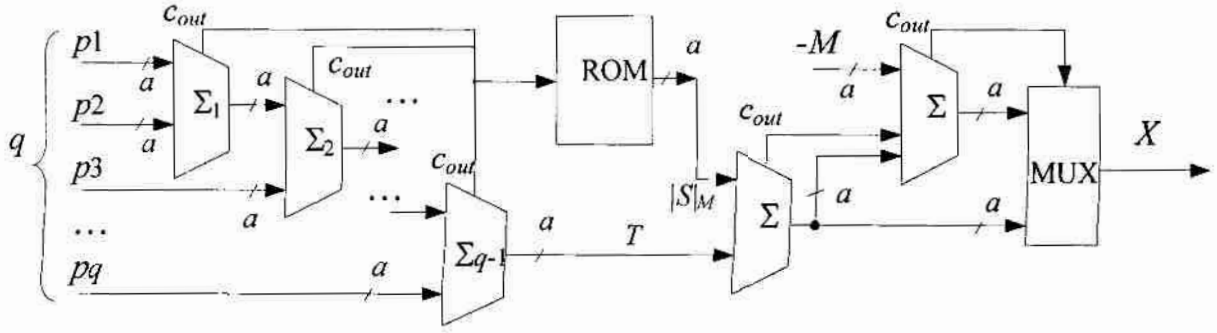
Rys. 3.15. Propozycja nowej struktury MOMA

Tab. 3.7. Objętość pamięci ROM dla różnych struktur MOMA

struktura MOMA	rozmiar bloku ROM (liczba a - bitowych rejestrów)					
	$q=4$	$q=5$	$q=7$	$q=8$	$q=9$	$q=10$
struktura znana [29]	64	128	256	256	512	512
proponowana struktura	8	16	64	127	256	512
realizacja prądowa proponowanej struktury	3	4	6	7	8	9

Każdy 2-argumentowy a-bitowy sumator CPA w strukturze MOMA z rys. 3.15, w prądowej wersji konwertera RNS-BIN, może być zbudowany z a jednocyfrowych sumatorów prądowych przedstawionych na rys. 3.12.

Ponadto, blok ROM, dla wartości $q<12$, w technologii prądowej, posiada tylko jedną wejściową linię adresową zamiast $(q\text{-}1)$ linii w strukturze z rys. 3.15, ponieważ wyjścia przeniesienia *Cout* wszystkich sumatorów CPA są połączone w jeden węzeł, w którym obliczana jest wartość k. To pozwala zmniejszyć rozmiar pamięci ROM z $(2^{q-1}-1)$ do $(q-1)$ komórek. I-ta komórka ROM zawiera teraz wartość $\left|i\cdot 2^{\lceil \log_2 M\rceil}\right|_M$. Prądowa wersja zaproponowanej struktury MOMA przedstawiona jest na rys. 3.16.

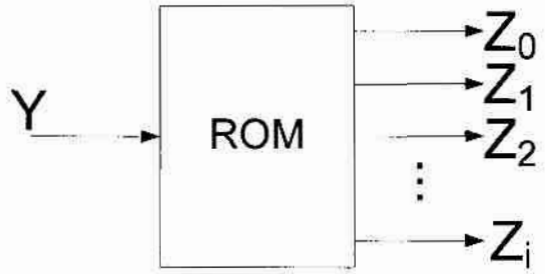


Rys. 3.16. Prądowa wersja zaproponowanej nowej struktury MOMA

Złożoność sprzętowa prądowej wersji konwertera RNS-BIN jest mniejsza w porównaniu z klasycznym prototypem przedstawionym na rys. 3.15.

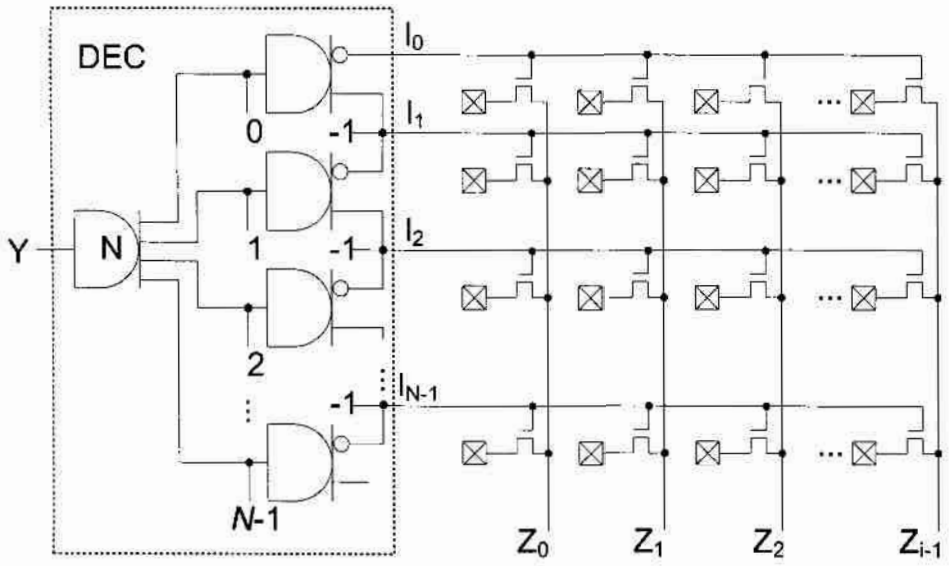
Ponieważ w proponowanym rozwiązaniu konwertera RNS-BIN uczestniczy blok pamięci stałej ROM o jednocyfrowym wejściu adresowym, autor dołączył opis możliwej realizacji takiego bloku w technologii prądowej.

Ogólny schemat bloku ROM zastosowanego w konwerterze RNS-BIN przedstawiono na rys. 3.17, a przykładowa realizacja przedstawiona jest na rys. 3.18.



Rys. 3.17. Ogólny schemat bloku ROM

Sygnał Y jest zmienną N-wartościową, dlatego na wejściu ROM może pojawić się dowolna wartość z zakresu 0 do N-1. Odpowiednia logiczna wartość wejściowa Y powoduje uaktywnienie jednej z linii $l_0 - l_{N-1}$ dekodera DEC w bloku ROM (rys. 3.18). Wartość logiczna 1 spowoduje uaktywnienie tranzystorów podłączonej do danej linii l oraz przekazanie odpowiednich wartości logicznych na wielowartościowe wyjścia Z_0 do Z_i.



Rys. 3.18. Przykładowa realizacja bloku ROM o jednocyfrowym wejściu adresowym w technologii prądowej

4. Sposoby i narzędzia programowe do opisu, konstruowania i weryfikacji cyfrowych układów prądowych na poziomie logicznym

Jednym z głównych zadań podczas projektowania prądowych układów cyfrowych jest testowanie ich modeli na poziomie logicznym, czyli sprawdzenie czy układ (model układu) działa poprawnie. Dla prostych układów (składających się z kilku bramek lub kilku wejść) układ można testować „ręcznie" sprawdzając dla każdej kombinacji sygnałów wejściowych jaka wartość logiczna powstaje na wyjściu. Dla większych układów operacja ta pochłania dużo czasu, dlatego należy skorzystać z narzędzi, które w sposób bardziej automatyczny i szybszy sprawdzą czy dany układ spełnia nasze oczekiwania.

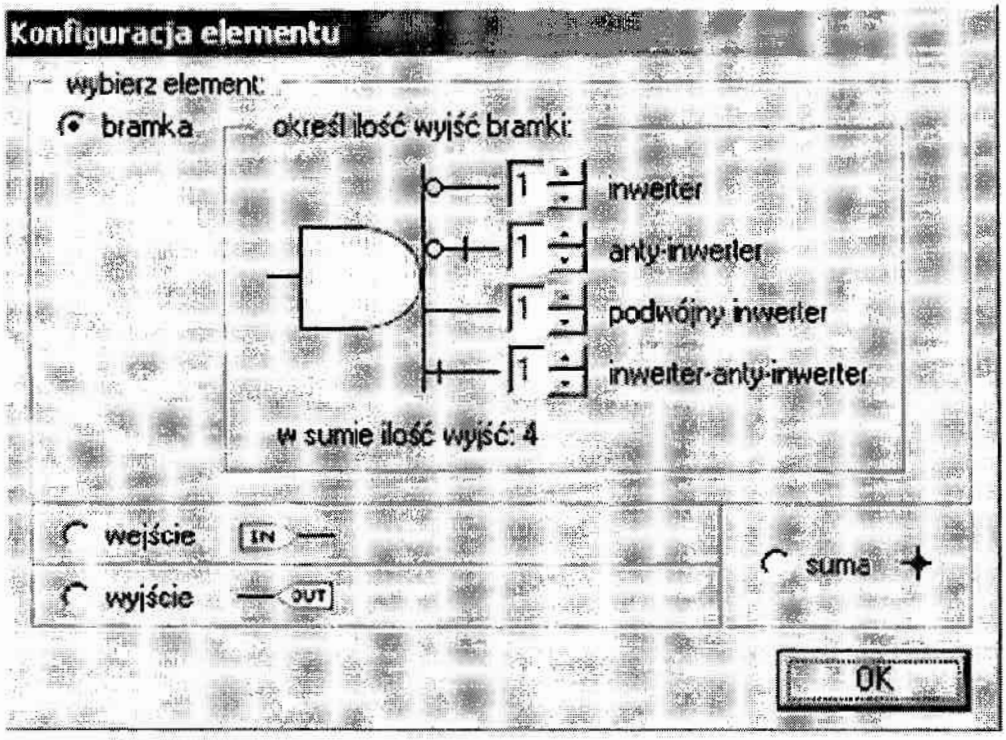
Logika bramek prądowych i same bramki prądowe całkowicie różnią się od logiki Boole'a i klasycznych bramek cyfrowych. Również narzędzia, które są dostępne do projektowania i testowania cyfrowych układów napięciowych nie nadają się do testowania układów prądowych. Zespół opracował nowe narzędzia przeznaczone do projektowania graficznego prądowych układów cyfrowych (patrz podrozdziały 4.1 i 4.2). Środowiska te nadawały się do testowania prostych i średnich modeli cyfrowych układów prądowych, przy większych systemach prądowych projektowanie układu jak i sprawdzanie poprawności połączeń trwało dość długo, dlatego wystąpiła potrzeba opracowania nowego sposobu testowania dużych układów prądowych. Wykorzystano do tego język VHDL i środowisko Active-HDL firmy Aldec, gdzie zaprojektowano logikę bramek prądowych (tablica rezolucji) jak i opracowano modele podstawowych bramek prądowych (podrozdział 4.3). Projektowanie w języku VHDL okazało się szybsze, łatwiejsze do weryfikacji i testowania szczególnie dużych układów. Samo projektowanie trwa nieco dłużej, ale sprawdzanie poprawności połączeń i testowanie za pomocą automatycznych testbenchów zdecydowanie przyspieszyło sprawdzanie poprawności działania układu. Wadą tej metody dla nowego projektanta jest konieczność znajomości języka VHDL, dlatego dodatkowo w środowisku Active-HDL umieszczono graficzne modele podstawowych bramek prądowych, dzięki czemu można tworzyć różnego rodzaju układy przy pomocy ich graficznych symboli.

4.1. Program MPUK do konstruowania i weryfikacji układów binarnych

Pierwszym narzędziem do konstruowania i symulacji modeli układów prądowych było środowisko graficzne „MPUK". Program powstał na początku badań nad bramkami

prądowymi, kiedy znane były tylko bramki binarne, a nie istniały żadne bramki prądowe dla systemów wielowartościowych o podstawie N>2.

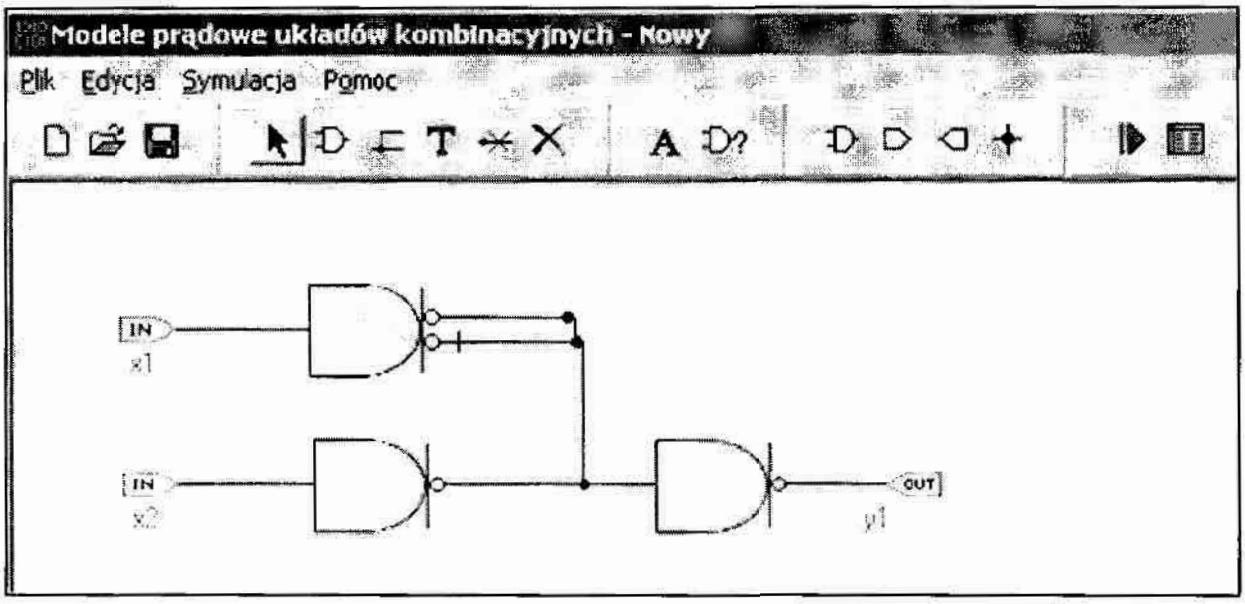
Środowisko „MPUK" zawiera wbudowany edytor graficzny, który określa szereg podstawowych operacji niezbędnych przy projektowaniu modeli układów. Na rys. 4.1 pokazane jest okno z elementami jakie można wybrać w programie „MPUK". Głównym elementem tego okna jest konfiguracja rodzaju bramki, którą można wstawić do projektowanego układu. Można ustalić jakie i ile wyjść różnego rodzaju ma znajdować się w budowanej bramce. Oprócz bramek w projektowanym układzie mogą się pojawić wejścia, wyjścia oraz węzły sumujące sygnały.



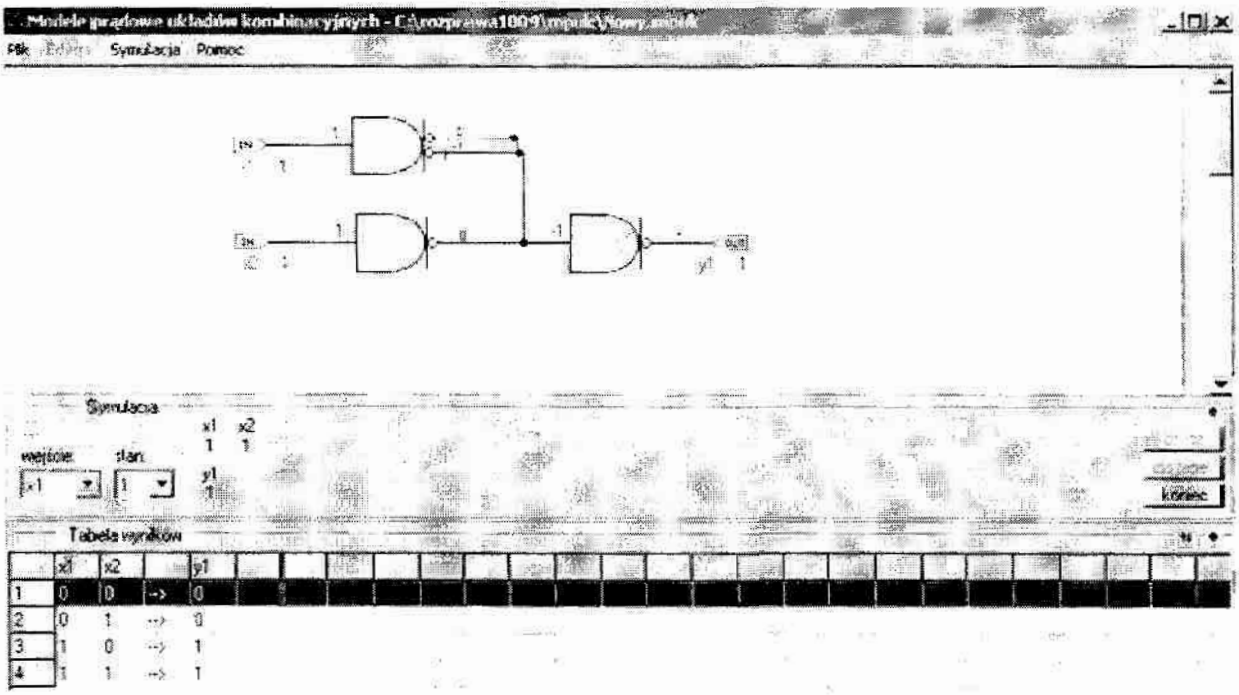
Rys. 4.1. Okno wyboru elementu schematu w programie „MPUK"

Po wybraniu elementu (bramki, wejścia wyjścia lub węzła) umieszcza się go w głównym oknie programu (rys. 4.2), a następnie używając połączeń tworzy się cały schemat.

Edytor umożliwia przesuwanie i usuwanie elementów. Łączenie elementów wykonuje się automatycznie, zadaniem projektanta jest wskazanie punktów początkowego i końcowego (wyprowadzeń), które chce ze sobą połączyć. Program umożliwia usuwanie błędnych połączeń. Można również wstawiać dowolne teksty. Wadą jest brak wpływu na ułożenie połączeń oraz brak możliwości zmian nazw wejść i wyjść. Schemat można zapisać do pliku binarnego, jednak nie można eksportować do postaci graficznej (np. jako plik bmp lub gif).



Rys. 4.2. Główne okno projektowe programu „MPUK" wraz z przykładowym schematem



Rys. 4.3. Okno symulacji programu „MPUK"

Po narysowaniu układu należy przejść do okna symulacji. Jeśli układ jest poprawnie zaprojektowany pojawi się nowe okno (rys. 4.3), gdzie można wybrać przy pomocy myszki wartości logiczne sygnałów wejściowych i odczytać wartości logiczne sygnałów wyjściowych (w tabeli i na schemacie) i wewnętrznych (tylko na schemacie).

Wartości logiczne sygnałów wejściowych łatwo zmieniać – kliknięcie prawym przyciskiem myszki na nazwie sygnału zwiększa wartość logiczną o 1 poziom, a lewym przyciskiem zmniejsza o 1 poziom. Po ustaleniu danych wejściowych przyciskiem „oblicz" ustalane są

wartości logiczne dla wyjść i sygnałów wewnętrznych. Również prostym przyciskiem „do tabeli" można „ręcznie" wygenerować tablicę prawdy dla testowanego układu. Wadami symulacji jest brak eksportu tabeli do pliku, brak możliwości usuwania danych dodanych do tabeli. Dobrym pomysłem byłoby pobieranie wektorów wejściowych z dodatkowego pliku, szczególnie jeśli projektant chce przetestować układ o wielu wejściach. Sygnały wejściowe mogą przybierać wartości logiczne z zakresu od „-8" do „+8".

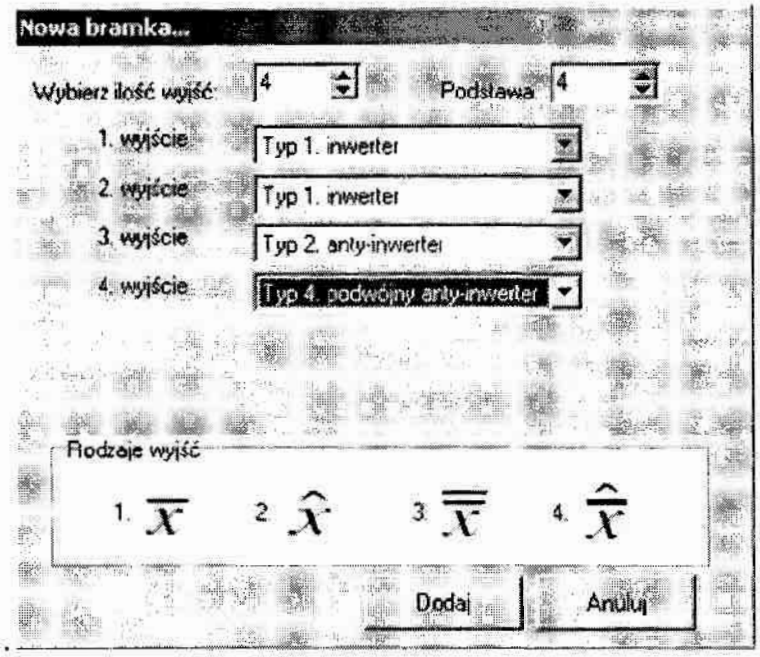
Program nadaje się do testowania prostych układów logicznych, w których występują tylko binarne bramki prądowe. Główną zaletą jest to, że wystarczy znać bramki prądowe i algebrę prądową, gdyż obsługa tego środowiska jest prosta i bardzo intuicyjna. Nie ma potrzeby znajomości dodatkowych zagadnień (jak choćby języka VHDL).

Program „MPUK" został stworzony w języku C++ i działa pod systemami Windows 9x, Windows 2000 oraz Windows XP.

4.2. Program „StreamSim" do konstruowania i weryfikacji układów *N*-wartościowych

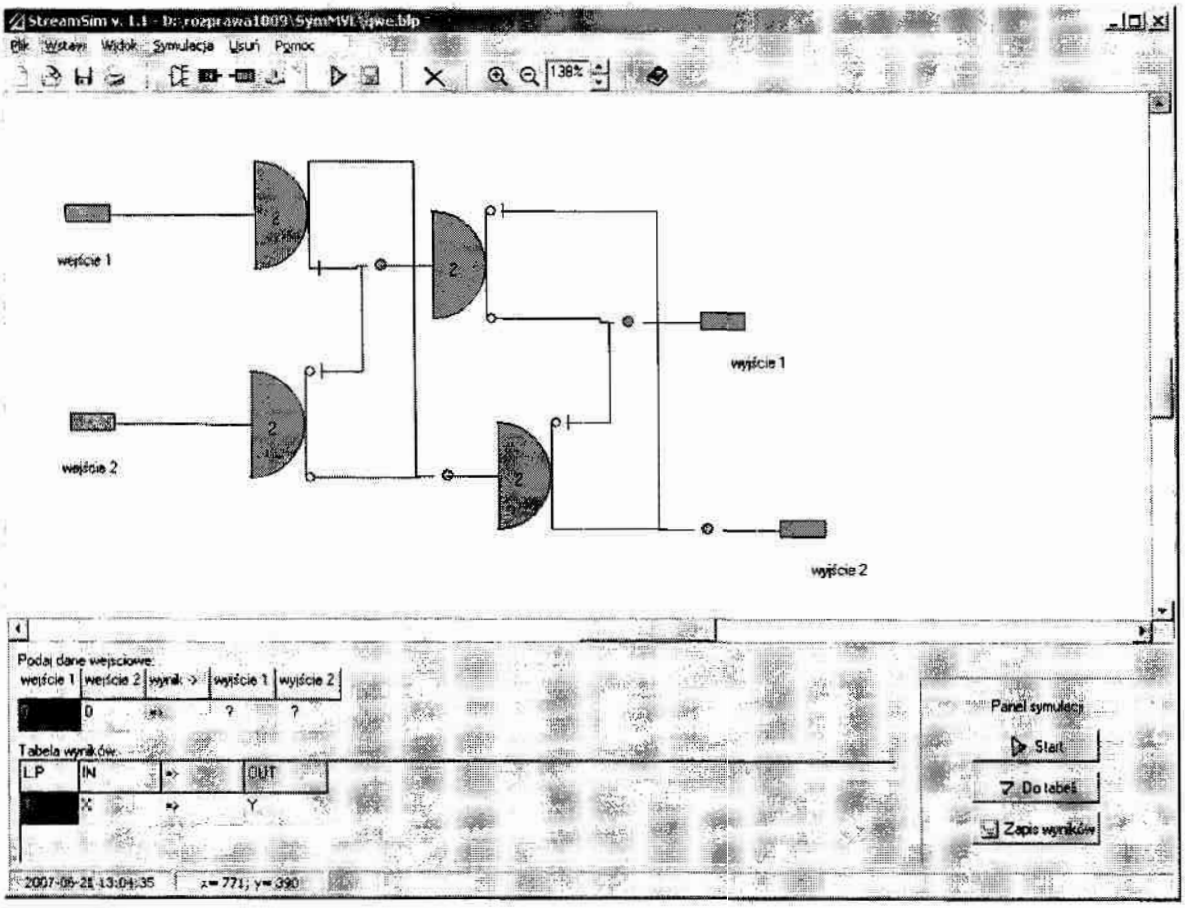
W ramach rozwoju prac nad algebrą prądową stworzono nowe bramki, których opisy nie zawierał program „MPUK". Opracowano nowe założenia algebry prądowej, szczególnie uwzględniając możliwość budowy układów działających w logice wielowartościowej o dowolnej podstawie *N*. Zaistniała potrzeba stworzenia nowego środowiska graficznego przeznaczonego do szybkiego i prostego testowania modeli układów pracujących w logice wielowartościowej z podstawą *N*. Nowe środowisko nazwano „StreamSim".

Program „StreamSim" podobnie jak „MPUK" pozwala na wprowadzenie 4 różnych elementów: bramek prądowych, wejść, wyjść oraz węzłów sumujących. Na rys. 4.4 pokazano okno do wyboru rodzaju bramki prądowej. Określa się w nim liczbę wyjść bramki (maksymalnie 8), a następnie dla każdego wyjścia określa się jego typ (jeden z czterech). Ważne jest również wybranie podstawy systemu wielowartościowego *N* w jakim modelowana bramka ma pracować (domyślnie N=2). Na rys. 4.4 przedstawiono przykładowy model bramki R1124. Bramka ta tworzona jest dla systemu wielowartościowego o podstawie 4, posiada dwa wyjścia typu inwerter i po jednym wyjściu typu anty-inwerter i anty-podwójny-inwerter.



Rys. 4.4. Okno wyboru bramki prądowej w środowisku „StreamSim"

Na rys. 4.5 pokazany jest przykładowy schemat układu zbudowanego w środowisku „StreamSim".



Rys. 4.5. Główne okno środowiska „StreamSim"

Środowisko umożliwia dodawanie, usuwanie i przesuwanie elementów (bramek, wejść, wyjść i węzłów). Istniej możliwość wprowadzania nazw wejść i wyjść, bardzo przydatna jest funkcja skalowania, która pozwala powiększyć lub pomniejszyć główne okno środowiska „StreamSim". Istnieje możliwość zapisania układu do pliku i późniejszego go odtworzenia. Zaletą też jest możliwość eksportu wyników symulacji. Wyniki te są eksportowane do pliku tekstowego w formie tabeli, która również jest widoczna w głównym oknie programu. Przykładowa zawartość wyników symulacji dla schematu z rys. 4.5 przedstawiono na rys. 4.6.

```
Wyniki symulacji cyfrowego ukladu pradowego
Start - dzien 2007-01-02 godz.00:34:06-
```

L.P.	in1	in2	wynik ->	out1
1.	0	0	=>	0
2.	1	0	=>	0
3.	2	0	=>	0
4.	3	0	=>	1
5.	4	0	=>	1
6.	4	1	=>	1
7.	3	1	=>	1
8.	2	1	=>	1
9.	1	1	=>	0
10.	0	1	=>	0

Rys. 4.6. Przykładowa tabela prawdy testowanego układu wygenerowana w środowisku „StreamSim"

Wartości logiczne na wejściu można zmieniać w podobny sposób jak to miało miejsce w programie „MPUK" – kliknięcie prawym przyciskiem myszki na nazwie sygnału zwiększa wartość logiczną o 1 poziom, a lewym przyciskiem zmniejsza o jeden. W tabeli wyników znajdują się wartości logiczne z wejść i wyjść układu umieszczone po kliknięciu na przycisk „do tabeli". Jeśli interesują nas sygnały wewnętrzne to po ustaleniu wartości logicznych na wejściu układu można je obejrzeć bezpośrednio na schemacie. Zaletą środowiska jest fakt, że można obejrzeć wartości sygnałów na wyjściu każdej bramki i dodatkowo wartość logiczną na węźle sumy (lub na wejściu kolejnej bramki), co nie było możliwe w środowisku Active HDL. Zaletą programu jest możliwość ustalania kolejności wyjść bramki tak jak jest to dla projektanta najwygodniej oraz możliwość usuwanie zbędnych wiersz z tabeli wyników. Przydatna jest również opcja zmiany podstawy systemu wielowartościowego, którą zadeklarował projektant przy tworzeniu bramki. Prosta i intuicyjna obsługa środowiska powoduje, że bardzo łatwo i szybko można testować dowolny układ działający w logice o dowolnej podstawie N. Wadą środowiska jest brak możliwości łączenia węzłów, brak wpływu na położenie połączeń wewnątrz układu, oraz brak możliwości wczytania z pliku

pobudzeń układu, tak by automatycznie otrzymać wyniki testów dla różnych wektorów testowych. Program „StreamSim" został stworzony w języku C++ i działa pod systemami Windows 9x, Windows 2000 oraz Windows XP.

4.3. Wykorzystanie języka VHDL i środowiska Active-HDL do opisu, weryfikacji i wizualizacji układów prądowych

Dla dużych systemów projektowanie przy użyciu środowisk graficznych jest pracochłonne i skomplikowane, a sprawdzenie poprawności połączeń zabiera dużo czasu. Szukając sposobów rozwiązania tego problemu spróbowano wykorzystać narzędzia jakie są dostępne przy projektowaniu układów napięciowych. Skorzystano w tym celu z języka opisu sprzętu VHDL i środowiska Active-HDL firmy Aldec. Standardowa biblioteka IEEE, w której zadeklarowane są podstawowe bramki i typy sygnałów dla logiki napięciowej nie nadaje się do projektowania układów prądowych. Głównym typem danych wykorzystywanym przy realizacji cyfrowych układów napięciowych w języku VHDL jest typ „std_logic". Istnieje również tzw. tablica rezolucji, w której opisano jak określić rezultat, gdy połączy się ze sobą dwa sygnały typu „std_logic". Bazując na tych danych zespół badawczy stworzył nowy typ danych „nstd_logic" opisujący logikę prądową oraz tablicę rezolucji opisującą jak określić rezultat w przypadku połączenia dwóch sygnałów typu „nstd_logic". Elementy te zostały umieszczone w bibliotece „nstd_logic_2000".

Zadeklarowany typ „nstd_logic" składa się z 12 różnych symboli opisanych w tab. 4.1. Dziesięć z nich opisuje wartości logiczne jakie mogą pojawić się w układach prądowych. Wartość 'U' opisuje stan początkowy układu, kiedy nie wiadomo jakie wartości znajdują się w poszczególnych węzłach.

Tab. 4.1. Opis typów danych „nstd_logic" z biblioteki „nstd_logic_2000"

symbol	znaczenie	symbol	znaczenie
E	sygnalizacja błędu	1	wartość logiczna 1
U	stan niezainicjowany	0	wartość logiczna 0
5	wartość logiczna 5	*A*	wartość logiczna -1
4	wartość logiczna 4	*B*	wartość logiczna -2
3	wartość logiczna 3	*C*	wartość logiczna -3
2	wartość logiczna 2	*D*	wartość logiczna -4

Na podstawie symboli umieszczonych w tab. 4.1 stworzono tablicę rezolucji przedstawioną na rys. 4.7. Dwa sygnały wejściowe (zmienne logiczne) określają wyjściową wartość

logiczną. Poszczególne pola wypełniane były zgodnie z regułami logiki prądowej, czyli wynikiem jest sumę algebraiczną tych wartości wejściowych. W przypadku przekroczenia wartości granicznych ('5' lub '-4') na wyjściu pojawi się symbol określający błąd ('*E*').

	U	***E***	***D***	***C***	***B***	***A***	**0**	**1**	**2**	**3**	**4**	**5**
U	*U*	*E*	*D*	*C*	*B*	*A*	0	1	2	3	4	5
E	*E*	*E*	*E*	*E*	*E*	*E*	*E*	*E*	*E*	*E*	*E*	*E*
D	*D*	*E*	*E*	*E*	*E*	*E*	*D*	*C*	*B*	*A*	0	1
C	*C*	*E*	*E*	*E*	*E*	*D*	*C*	*B*	*A*	0	1	2
B	*B*	*E*	*E*	*E*	*D*	*C*	*B*	*A*	0	1	2	3
A	*A*	*E*	*E*	*D*	*C*	*B*	*A*	0	1	2	3	4
0	0	*E*	*D*	*C*	*B*	*A*	0	1	2	3	4	5
1	1	*E*	*C*	*B*	*A*	0	1	2	3	4	5	*E*
2	2	*E*	*B*	*A*	0	1	2	3	4	5	*E*	*E*
3	3	*E*	*A*	0	1	2	3	4	5	*E*	*E*	*E*
4	4	*E*	0	1	2	3	4	5	*E*	*E*	*E*	*E*
5	5	*E*	1	2	3	4	5	*E*	*E*	*E*	*E*	*E*

Rys. 4.7. Tablica rezolucji dla typu „nstd_logic" z biblioteki „nstd_logic_2000"

W bibliotece również zadeklarowano 4 podstawowe bramki logiczne występujące w algebrze bramek prądowych. Nazwy tych bramek i ich symbole przedstawiono w tab. 4.2.

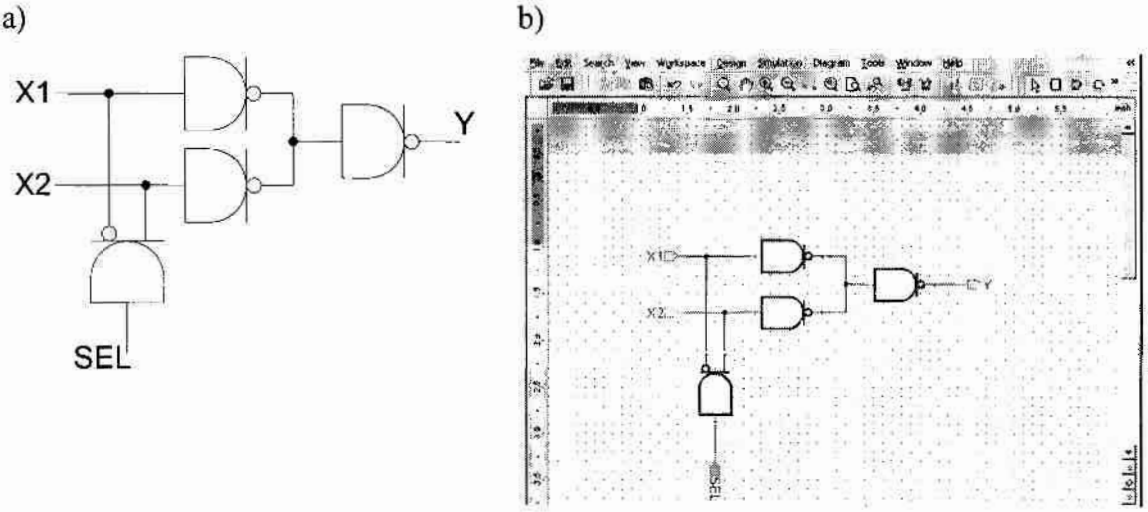
Tab. 4.2. Typy bramek prądowych umieszczone w bibliotece „nstd_logic_1999"

symbol	typ bramki
R1	inwerter
R2	anty-inwerter
R3	podwójny-inwerter
R4	anty-podwójny-inwerter

Biblioteka „nstd_logic_2000" pozwala na projektowanie układów prądowych o dowolnej liczbie bramek, a jedynym ograniczeniem była wartość zmiennej logicznej w pojedynczym węźle. Program (4.1) przedstawia opis w języku VHDL układu prądowego, którego schemat znajduje się na rys. 4.8a (układ multipleksera MUX).

W zależności od stopnia złożoności układu, który się projektuje testowanie modelu układu zrealizowanego za pomocą języka VHDL można przeprowadzić na kilka sposobów. Najprostszym sposobem jest ręczne generowanie wektorów testowych (ustalanie wartości logicznych na wejściach układu) i odczytywanie wartości na wyjściach. W środowisku

Active-HDL wygodne jest korzystanie z przebiegów w formie „Waveformów" lub „List", gdzie przedstawione są wartości logiczne na wejściach i wyjściach oraz dodatkowo można odczytać wartości sygnałów wewnątrz układu.



Rys. 4.8. Układ multipleksera MUX zbudowany z bramek prądowych- schemat (a), okno programu wykorzystującego graficzne symbole biblioteki „nstd_logic_2000" (b)

```
LIBRARY nstd_logic_2000;
use nstd_logic_2000.nstd_logic_2000.all;
entity MUX is
port(   X1,X2,SEL : in nstd_logic;
        Y : out nstd_logic      );
end MUX;
architecture MUX of MUX is
signal X1A,X2A,YA : nstd_logic;
component R1
port(   in1: in nstd_logic;
     out1: out nstd_logic );
end component R1;
component R13
port(   in1: in nstd_logic;
                out1: out nstd_logic;
                out2: out nstd_logic   );
end component R13;
begin
        X1A <= X1;
        X2A <= X2;
        B1 : R1  port map(X1A,YA);
        B2 : R1  port map(X2A,YA);
        B3 : R13 port map(SEL,X1A,X2A);
        B4 : R1  port map(YA,Y);
end MUX;
```

(4.1)

W przypadku projektowania rozbudowanych układów prądowych wygodnie jest skorzystać z *Testbenchy*, które pozwalają na wielokrotne szybkie testowanie układu określonymi, stałymi wektorami testowymi. *Testbenche* zapisane w języku VHDL, pozwalają wygenerować przebiegi (w formie Waveformów lub List) wejściowe i wyjściowe. Umożliwiają również porównanie otrzymanych wyników z rzeczywistą tablicą prawdy układu.

Uzyskane doświadczenie w weryfikacji modeli układów prądowych wykazało, że w przypadku układów prostych lub o regularnej strukturze prościej jest tworzyć schemat graficzny układu, niż tworzyć jego odpowiednik w języku VHDL. Środowisko Active-HDL umożliwia utworzenie graficznych symboli dla elementów znajdujących się w zaprezentowanej bibliotece „nstd_logic_2000". Korzystając z symboli graficznych łatwo wizualnie zaprojektować dowolny układ prądowy, a następnie automatycznie wygenerować z niego opis w języku VHDL. Na rys. 4.8b pokazany jest multiplekser MUX „narysowany" w środowisku Active-HDL zbudowany przy wykorzystaniu elementów graficznych dodanych do biblioteki „nstd_logic_2000".

Biblioteka „nstd_logic_2000" w swojej ostatecznej wersji zawiera opis typu „nstd_logic", tablicę rezolucji, o której wspomniano powyżej, cztery podstawowe bramki prądowe (R1, R2, R3, R4), dużą ilość bramek „pochodnych" (posiadających różne ilość wyjść różnych typów bramek np. R12, R22, R34) najczęściej używanych przy projektowaniu prądowych układów cyfrowych, stałe wartości logiczne ('*D*', '*C*', ...,'0', ..., '4', '5') oraz graficzne symbole wszystkich wymienionych elementów.

W trakcie badań nad układami pracującymi w logice wielowartościowej okazało się, że biblioteka „nstd_logic_2000" nie pozwala na zaprojektowanie i przetestowanie niektórych układów. Główną wadą okazało się ograniczona liczba wartości logicznych, które można przedstawić w tejże bibliotece – jedynie 10 różnych wartości logicznych ('*D*', '*C*', ...,'0', ..., '4', '5'). Proste rozszerzenie o dodatkowe wartości logiczne ('6', '7', ... '*E*', '*F*') pozwala na rozszerzenie do 20 różnych (sensownych, bo po wartości logicznej '9' nie bardzo wiadomo jakiej kolejnej cyfry użyć), a maksymalnie do 36 (wszystkie cyfry arabskie i litery alfabetu łacińskiego). Projektowanie nawet niezbyt skomplikowanych układów pracujących w logice wielowartościowej wykazało, że należałoby użyć przynajmniej typu danych o 60 różnych wartościach logicznych.

W pracy proponuje utworzenie nowej biblioteki „nstd_logic_mvl" przeznaczonej do testowania modeli układów prądowych pracujących w logice wielowartościowej. Biblioteka została zaprojektowana tak by móc ją (w razie potrzeby) w prosty sposób rozszerzać o kolejne

wartości logiczne. Podstawowym typem zadeklarowanym w nowej bibliotece jest typ „nstd_logic", który składa się z 63 stanów logicznych. 62 z tych stanów to odpowiednie poziomy logiczne (od wartości -30 do +30) oraz dwa stany: *U* – niezainicjowany oraz *X* – określający błąd. Wszystkie stany logiczne dla typu „nstd_logic" przedstawione są w tab. 4.3.

Tab. 4.3. Opis typu danych „nstd_logic" z biblioteki „nstd_logic_mvl"

symbol	**wartość logiczna**	**symbol**	**wartość logiczna**	**symbol**	**wartość logiczna**	**symbol**	**wartość logiczna**
E	błąd	*m*16	-16	*p*1	+1	*p*17	+17
U	niezainicjowany	*m*15	-15	*p*2	+2	*p*18	+18
*m*30	-30	*m*14	-14	*p*3	+3	*p*19	+19
*m*29	-29	*m*13	-13	*p*4	+4	*p*19	+19
*m*28	-28	*m*12	-12	*p*5	+5	*p*20	+20
*m*27	-27	*m*11	-11	*p*6	+6	*p*21	+21
*m*26	-26	*m*10	-10	*p*7	+7	*p*22	+22
*m*25	-25	*m*9	-9	*p*8	+8	*p*23	+23
*m*24	-24	*m*8	-8	*p*9	+9	*p*24	+24
*m*23	-23	*m*7	-7	*p*10	+10	*p*25	+25
*m*22	-22	*m*6	-6	*p*11	+11	*p*26	+26
*m*21	-21	*m*5	-5	*p*12	+12	*p*27	+27
*m*20	-20	*m*4	-4	*p*13	+13	*p*28	+28
*m*19	-19	*m*3	-3	*p*14	+14	*p*29	+29
*m*18	-18	*m*2	-2	*p*15	+15	*p*30	+30
*m*17	-17	*m*1	-1	*p*16	+16		

Zadeklarowano tablice rezolucji, określającą wartość logiczną jaka powstanie w przypadku połączenia ze sobą dwóch sygnałów typu nstd_logic. Dodatkowo zadeklarowano typ „nstd_logic_vector", który umożliwia tworzenie wektora sygnałów, operacji często wykorzystywanej przy projektowaniu modeli układów cyfrowych. W bibliotece przewidziano wprowadzenie czasów opóźnień generowanych przez poszczególne bramki, odpowiada za nie stała „delay_time". W trakcie pisania rozprawy czasy opóźnień nie były dokładnie określone i w bibliotece wstawiono w te miejsca same zera. Resztę biblioteki tworzą bramki prądowe (podstawowe i pochodne) dla logik wielowartościowych o różnych podstawach. Biblioteka „nstd_logic_mvl" zawiera również ich symbole graficzne. Mimo, że istnieją tylko cztery typy wyjść w technologii bramek prądowych to różna liczba bramek jakie można z nich utworzyć

jest bardzo duża (bramka R1, R2, R21, R3333, R113, itd.). Poza tym te same bramki pracujące w logikach o różnych podstawach będą działały inaczej, czyli potrzebne są nowe modele umieszczone w bibliotece. Obecnie biblioteka „nstd_logic_mvl" nie zawiera modeli wszystkich bramek prądowych lecz jedynie te, które były potrzebne w dotychczas tworzonych projektach. Każdy nowy model bramki prądowej tworzony jest w oparciu o odpowiednią funkcję „*gateN*" znajdującą się w bibliotece „nstd_logic_mvl", gdzie „*N*" w nazwie oznacza podstawę systemu wielowartościowego w jakiej ten model działa. Przykładowo program (4.2) przedstawia opis funkcji „*gate*2" w języku VHDL, na której podstawie tworzone są modele dla systemu binarnego.

```
function gate (typ : integer) return
                nstd_logic_vector is
begin
     case typ is
          when 1 => return (p0,p1);
          when 2 => return (m1,p0);
          when 3 => return (p1,p0);
          when 4 => return (p0,m1);
          when others => return (X,X);
     end case;
end function gate;
```
(4.2)

Program (4.3) przedstawia opis funkcji „*gate5*" przeznaczonej do tworzenia modeli dla systemu wielowartościowego o podstawie 5. Przyjęto również ogólną zasadę pomijania we wszelakich nazwach wartości podstawy systemu, jeśli model pracuje w systemie binarnym (system wielowartościowy o podstawie 2). Jednak dla przejrzystości w bibliotece umieszczono obie wersje modeli.

```
function gate5 (typ : integer) return
     nstd_logic_vector is
begin
     case typ is
          when 1 => return (p0,p1,p2,p3,p4);
          when 2 => return (m4,m3,m2,m1,p0);
          when 3 => return (p4,p3,p2,p1,p0);
          when 4 => return (p0,m1,m2,m3,m4);
          when others => return (X,X,X,X,X);
     end case;
end function gate5;
```
(4.3)

Każda funkcja „*gateN*" zawiera instrukcję „case", która wybiera typ wyjścia bramki prądowej (za każdym razem jednej z czterech), której model chce się stworzyć i zwraca wektor, który

określa wszystkie możliwe wartości jakie to wyjście może zwrócić (patrz tab. 4.2). Przykładowo dla bramek binarnych jest to wektor dwuelementowy, a dla logiki wielowartościowej o podstawie 5 wektor pięcioelementowy. W zwracanym wektorze pierwsza wartość określa wynik jaki otrzymano jeżeli na wejściu modelu bramki pojawi się wartość logiczna (N-1) lub większa. Kolejne elementy wektora określają odpowiednio wynik dla wartości logicznej na wejściu (N-2), (N-3), ..., 1, 0. Program (4.4) przedstawia model w języku VHDL bramki prądowej R1, jest to bramka w wyjściem typu inwerter działająca w logice binarnej. Liczby z lewej strony oznaczają numer wiersza i należy je pominąć w projekcie VHDL.

```
1  LIBRARY nstd_logic_mvl;
2  use nstd_logic_mvl.nstd_logic_mvl.all;
3     entity R1 is generic (tprop:time:=delay_time(1));
4     port (    in1: in nstd_logic;
5               out1: out nstd_logic );
6     end R1;
7
8     architecture R1 of R1 is
9          signal in1temp : nstd_logic;
10    begin
11                                                          (4.4)
12    with in1 select
13         in1temp <= p0 when m30 to p0,
14                    p1 when p1 to p30,
15                    in1 when others;
16
17    with in1temp select
18         out1 <= gate(1)(1-nstd2int(in1temp))
                              after tprop when p0 to p1,
19                 U after tprop when U,
20                 X after tprop when others;
21    end architecture R1;
```

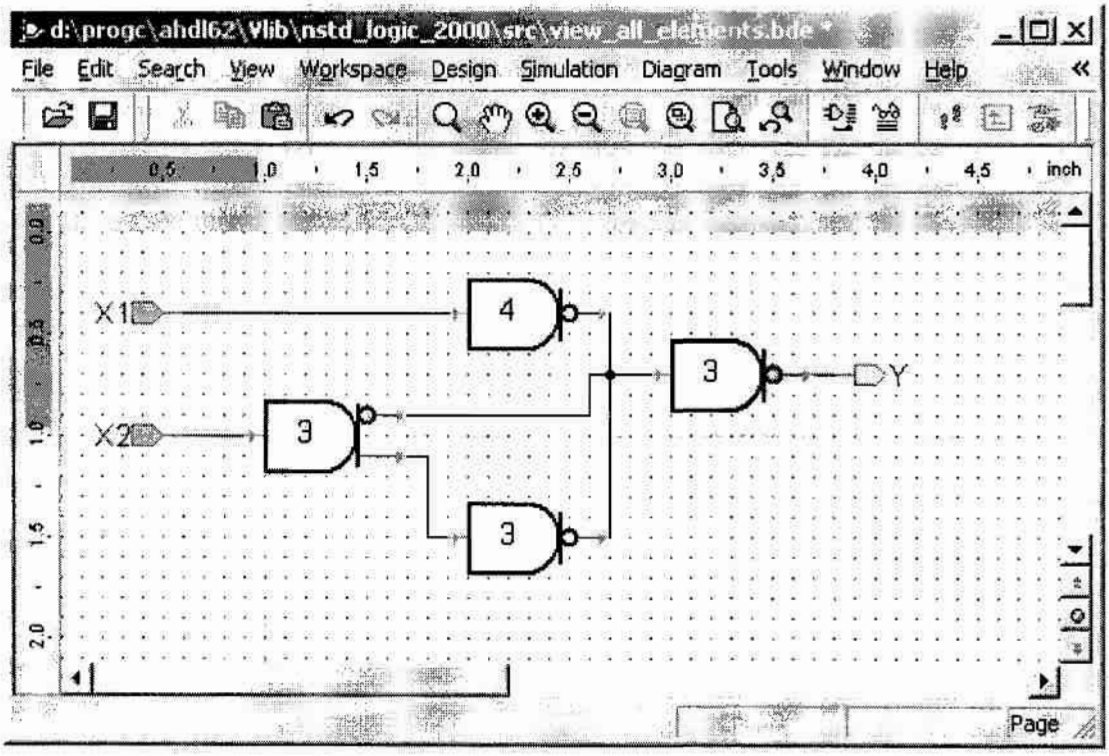
Program (4.5) przedstawia model w języku VHDL bramki prądowej R12, z wyjściem typu inwerter i anty-inwerter działającej w logice wielowartościowej o podstawie 5. Każdy model bramki w języku VHDL składa się z kilku charakterystycznych bloków. Linie 1 i 2 w programach (4.4) i (4.5) wskazują na bibliotekę układów prądowych i muszą zawsze występować. Linie od 3 do 6 opisują zewnętrzne parametry bramki (ilości wejść i wyjść) oraz wskazują jej nazwę. Kolejne linie opisują odpowiedni model bramki. Istnieje tyle różnych poziomów logicznych na wyjściu bramki jaka jest liczba podstawy systemu wielowartościowego, w której ta bramka pracuje. Linie 12 – 15 powodują ograniczenie wartości poziomów logicznych do tych jakie są dozwolone w wybranym

systemie wielowartościowym. Ograniczenie to jest potrzebne ze względu na fakt, że w całym układzie mogą się pojawić dowolne poziomy logiczne, jednak sama bramka działa w systemie wielowartościowym o ustalonej podstawie. Linie 17 – 20 generują odpowiedni poziom logiczny na pierwsze wyjście bramki, wykorzystując opisaną wcześniej funkcję „gateN". Dla bramki R1 będzie to ostatni element modelu VHDL. Dla innych bramek w zależności od liczby wyjść model VHDL będzie zawierał tyle podobnych bloków (linii 17 – 20) ile wyjść posiada dana bramka. Bramka R12, której model przedstawiony jest w programie (4.5) posiada dwa wyjścia zatem powinien znajdować się w niej jeden blok więcej niż dla bramki R1. Blok ten tworzą linie 22 – 25, generując odpowiedni poziom logiczny na wyjście drugie modelu.

```
1 LIBRARY nstd_logic_mvl;
2 use nstd_logic_mvl.nstd_logic_mvl.all;
3    entity R12p5 is generic (tprop:time:=delay_time(2));
4    port (in1: in nstd_logic;
5          out1,out2: out nstd_logic );
6 end R12p5;
7
8    architecture R12p5 of R12p5 is
9         signal in1temp : nstd_logic;
10   begin
11
12   with in1 select
13        in1temp <= p0 when m30 to p0,
14                   p4 when p4 to p30,
15                   in1 when others;
16
17   with in1temp select
18        out1 <=   gate5(1)(4-nstd2int(in1temp))
                         after tprop when p0 to p4,
19                  U after tprop when U,
20                  X after tprop when others;
21
22   with in1temp select
23        out2 <=   gate5(2)(4-nstd2int(in1temp))
                        after tprop when p0 to p4,
24                  U after tprop when U,
25                  X after tprop when others;
26   end architecture R12p5;
```

(4.5)

Na rys. 4.9 przedstawiony jest przykładowy projekt wykonany za pomocą elementów graficznych umieszczonych w bibliotece *„nstd_logic_mvl"*. Opis tego układu w języku VHDL przedstawia program (4.6).



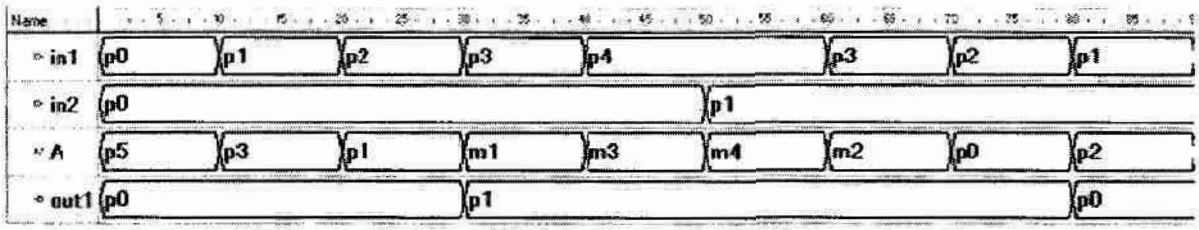
Rys. 4.9. Okno programu z przykładowym schematem układu wykonanego z elementów znajdujących się w bibliotece „nstd_logic_mvl"

```
LIBRARY nstd_logic_mvl;
use nstd_logic_mvl.nstd_logic_mvl.all;
     entity test is
     port (in1,in2: in nstd_logic;
             out1: out nstd_logic );
     end test;

     architecture test of test is
     component R1
     generic(tprop : TIME := delay_time(1));
     port(in1 : in nstd_logic;
          out1 : out nstd_logic);
     end component;
     component R12p5
     generic(tprop : TIME := delay_time(2));
     port(in1 : in nstd_logic;
          out1 : out nstd_logic;
          out2 : out nstd_logic);
     end component;
     signal A : nstd_logic;
     begin

     U1 : R12p5     port map (in1,A,A);
     U2 : R1   port map (in2,A);
     U23: R1   port map (A,out1);
     end architecture test;
```

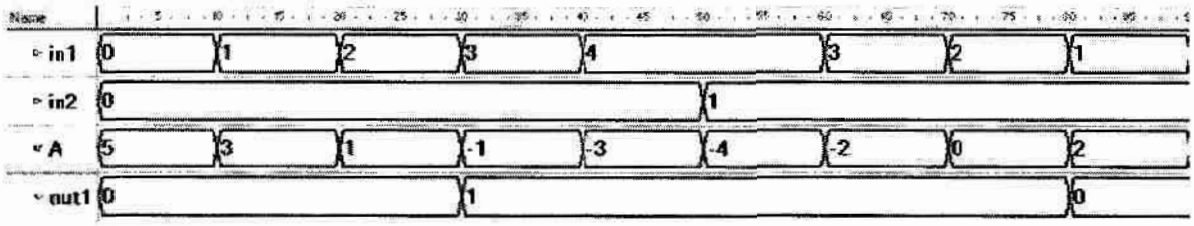
(4.6)

Po wykonaniu modelu układu (schemat graficzny z rys. 4.9. lub opis w języku VHDL z programu *(4.6)*) należy go przetestować korzystając z narzędzi dostępnym w środowisku Active-HDL. Można wygenerować waveform (rys. 4.10a), w którym ujęte są sygnały wejściowe i wyjściowe oraz można dodać do niego dowolne sygnały wewnętrzne. Dodatkowo można użyć aliasów, które zmienią nazwy poziomów logicznych z biblioteki „nstd_logic_mvl" (*m*1, *p*0, *p*1..) na zwykły zapis (-1, 0, 1) – rys. 4.10b.

a)



b)



Rys. 4.10. Przykładowy „waveform" wygenerowany dla układu z programu (4.6) w środowisku Active-HDL (a) oraz po użyciu aliasów (b)

Drugim sposobem przedstawienia wyników symulacji w środowisku Active-HDL jest lista (tabela), pokazana na rys. 4.11. Niestety nic ma tutaj możliwości stosowania aliasów jak to było w przypadku waveformów.

Time	A	in1	in2	out1
10.000 ns	p3	p1	p0	p0
20.000 ns	p1	p2	p0	p0
30.000 ns	m1	p3	p0	p1
40.000 ns	m3	p4	p0	p1
50.000 ns	m4	p4	p1	p1
60.000 ns	m2	p3	p1	p1
70.000 ns	p0	p2	p1	p1
80.000 ns	p2	p1	p1	p0
90.000 ns	p4	p0	p1	p0

Rys. 4.11. Przykładowa lista wygenerowana dla układu z programu (4.6) w środowisku Active-HDL

4.4. Porównanie opracowanych narzędzi programowych i ewentualne kierunki ich rozwoju

Testowanie modeli układów to jedna z najważniejszych prac wykonywanych przez projektanta. Proste i intuicyjne narzędzia pozwalają na szybkie i dokładne wprowadzenie modelu do określonego środowiska i sprawdzenie poprawności działania. Próby przeprowadzane na przestrzeni kilku lat wykazały przewagę środowiska Active HDL nad środowiskami graficznymi głównie dzięki możliwości szybkiej zmiany biblioteki bramek, z których budowane są większe modele. Środowisko Active HDL wymaga jednak dobrej znajomości języka VHDL, co dla początkującego projektanta może wydać się skomplikowane. Badania prowadzą do powstania konkretnych bramek (najpierw binarnych, następnie wielowartościowych). W momencie zakończenia prac nad nowymi bramkami można stworzyć środowisko graficzne proste i intuicyjne w obsłudze, które nie będzie wymagało znajomości żadnych dodatkowych zagadnień oprócz tych związanych bezpośrednio z algebrą prądową oraz z bramkami prądowymi. Rozwiązania zaprezentowane w programach „MPUK" i „StreamSim" pozwolą na stworzenie środowiska graficznego łatwiejszego w projektowaniu układów prądowych niż Active-HDL. Bardzo dobrze został rozwiązany sposób modelowania bramki wielowartościowej w środowisku „StreamSim". Zdecydowanie przewyższa pod tym względem bibliotekę graficzną Active-HDL, gdzie kolejność wyjść jest stała. W programie „StreamSim" jest możliwość wskazania ilości wyjść oraz ich kolejności. Jest to szczególnie przydatne podczas tworzenia dokumentacji, nie potrzebny jest dodatkowy program do tworzenia schematów. Dużym minusem obu aplikacji graficznych jest brak możliwości ręcznego ustawiania połączeń pomiędzy elementami. Tworzone przez projektanta modele układów można testować, ale wizualnie są często nieczytelne. Ważna jest możliwość edycji nazw wejść i wyjść oraz możliwość nazywania poszczególnych elementów. Środowisko powinno umożliwiać również wprowadzanie dowolnych tekstów i eksport narysowanego modelu zarówno do plików graficznych jak i zapisu do pliku aby móc go później odtworzyć. Drugim odrębnym zagadnieniem jest symulacja wprowadzonego modelu układu. Bardzo dobrze została rozwiązana zmiana sygnałów wejściowych, klikając prawym (w celu zwiększenia wartości) lub lewym (w celu zmniejszenia) przyciskiem myszy można symulować działanie układu dla pojedynczych, dowolnych pobudzeń. Jednak najczęściej testując układ projektant chce otrzymać wynik dla wszystkich (albo dla dużej liczby) wektorów testowych. Najprościej takie wektory trzymać w postaci pliku tekstowego. Środowisko powinno mieć możliwość wczytywania wektorów

pobudzeń i automatycznego generowania wyników albo w postaci tabeli wyników, którą można ręcznie (jak to ma miejsce w środowisku „StreamSim") albo automatycznie zapisać do pliku. Bardzo dobrym pomysłem jest wizualne przedstawienie symulacji dla określonego wektora pobudzeń bezpośrednio na schemacie, który widoczny jest cały czas na ekranie. Ważnym jest w algebrze prądowej, aby widoczne były nie tylko wartości na wejściach i wyjściach układu oraz w węzłach, ale również na wyjściach bramek. Wizualizacja taka bardzo ułatwia analizowanie układu i wyszukiwanie ewentualnych błędów projektowych.

Podsumowanie

W materiałach niniejszej rozprawie wykazano, że znane metody minimalizacji funkcji logicznych (np. Quine'a-McCluskey'a, Veitcha-Karnaugha i inne) mogą być bezpośrednio wykorzystane w algebrze bramek prądowych do uzyskania listy implikantów prostych zadanej N-argumentowej funkcji binarnej lecz dalsza minimalizacja funkcji już nic jest możliwa. Doświadczenie autora świadczy natomiast o tym, że grupa (N-1)-argumentowych implikantów z tej listy ma znaczący wpływ na złożoność wyrażenia wynikowego opisującego zadaną funkcję binarną, jak i na złożoność sprzętową układu prądowego realizującego tę funkcję, ponieważ liczba tych implikantów zwykle jest duża, przynajmniej większa od liczby wszystkich pozostałych implikantów prostych. Z tego powodu, w celu podwyższenia efektywności znanych metod minimalizacji w przypadku ich stosowania w algebrze bramek prądowych i (jako skutek) zmniejszenia złożoności sprzętowej układów prądowych, autor w rozprawie podjął próbę odnalezienia n-argumentowych funkcji binarnych ($n \leq N$), które składają się właśnie z implikantów (N-1)-argumentowych oraz których realizacja sprzętowa w technologii prądowej wymaga stosowania mniejszej liczby bramek (a nawet tranzystorów), niż w technologii napięciowej. W rozprawie funkcje takie zostały odnalezione i nazwano je funkcjami wzorcowymi typu T lub TBlok(n). Ponadto, autor wyprowadził wyrażenie ogólne opisujące n-argumentową funkcję TBlok(n), i udowodnił, że realizacja funkcji TBlok(n) wymaga wykorzystania O($3n$) tranzystorów w technologii prądowej (zamiast O($2n^2$) tranzystorów w technologii napięciowej).

Zachęcające wyniki porównania złożoności funkcji wzorcowych typu T i badań nad właściwościami logicznymi tych funkcji pozwoliły autorowi opracować m.in. prosty i nadający się do realizacji komputerowej algorytm do odnalezienia funkcji TBlok(n) o różnej liczbie argumentów n w N-argumentowej funkcji logicznej, zadanej za pomocą listy implikantów prostych ($n \leq N$). Udowodniono, że opracowany algorytm gwarantuje odnalezienie wszystkich nie powtarzających się funkcji TBlok(n) o różnej liczbie argumentów n, jeśli tylko takie funkcje istnieją wśród (N-1)-argumentowych implikantów funkcji wejściowej. Ponadto złożoność obliczeniowa algorytmu wynosi O($2 \cdot L^2$) operacji porównania (gdzie L – liczba (N-1)-argumentowych implikantów prostych funkcji wejściowej), co pozwala na jego stosowanie w kombinacji z dowolną znaną komputerową metodą minimalizacji funkcji binarnych bez widocznego wydłużenia czasu trwania całego procesu minimalizacji.

W oparciu o ten algorytm autor opracował sposób minimalizacji funkcji binarnych w algebrze bramek prądowych, który operuje na liście implikantów prostych lub pierwotnych, na tablicy prawdy lub na diagramach Veitcha-Karnaugha zadanej funkcji binarnej, przy czym w ostatnim przypadku (przy tzw. projektowaniu „ręcznym") określono kolejność działań projektanta w celu odnalezienia najprostszego wyrażenia opisującego funkcję wejściową. Ponadto, autor uprościł wyrażenia opisujące w algebrze bramek prądowych funkcje logiczne XOR dla różnej liczby argumentów, co pozwoliło znacznie uprościć złożoność sprzętową projektowanych układów prądowych (realizujących te funkcje) w porównaniu do wcześniej opracowanych projektów. Dlatego autor uważa, że główny cel pracy tj. dostosowanie znanych metod minimalizacji funkcji logicznych (np. Quine'a-McCluskey'a, Veitcha-Karnaugha lub innych) do minimalizacji funkcji binarnych w algebrze bramek prądowych, które było skierowane na zmniejszenie złożoności sprzętowej projektowanych układów prądowych pod względem zajmowanej powierzchni (w układzie ASIC), został osiągnięty.

Opracowany sposób minimalizacji został wielokrotnie wykorzystany przez autora przy projektowaniu i optymalizacji różnego stopnia złożoności układów prądowych, m.in. sumatorów jednobitowych oraz czterobitowych z równoległym przeniesieniem (ang. look-ahead adder), funkcji S-bloku z algorytmu kryptograficznego DES, a nawet podukładów bloku SLICE układów reprogramowalnych z rodziny Spartan II i Virtex. Niektóre z wyżej wymienionych projektów zostały zweryfikowane na poziomie logicznym w oparciu o opracowane z udziałem autora narzędzia programowe MPUK, StreamSim i bibliotekę prądową dla Active-HDL oraz zrealizowane fizycznie w postaci podukładów full-custom ASIC w technologii 0,6um przez dr inż. P. Pawłowskiego. Wyniki badań eksperymentalnych potwierdziły prawidłowość działania większości opracowanych układów w podstawowych trybach pracy. Ponadto stwierdzono, że powierzchnia opracowanych w oparciu o proponowany sposób minimalizacji układów prądowych jest porównywalna z powierzchnią odpowiednich układów zrealizowanych w technologii napięciowej (jest większa nie więcej niż o 2,5%). Z tego powodu autor uważa, że pierwsza główna teza pracy została udowodniona.

Technologia bramek prądowych umożliwia pojawienie się w złożonych z nich układach wartości logicznych różnych od „0" i „1" (np. mniejszych od „0" i/lub większych od „1") oraz pozwala, w bardzo prosty i naturalny sposób, na wykonanie operacji arytmetycznych na tych wartościach. Dzięki temu pojawia się możliwość wykorzystania bramek prądowych w układach arytmetycznych działających w N-wartościowych systemach

liczbowych, np. w arytmetyce *modulo N* i RNS, jeśli opracowane zostaną sposoby projektowania takich układów. Dlatego autor w rozprawie dokonał modyfikacji opracowanego w zespole badawczym „ręcznego" sposobu minimalizacji funkcji *N*-wartościowych o argumentach binarnych opartego o wykorzystanie funkcji bazowych i diagramy Veitcha-Karnaugha. Modyfikacja ta jest oparta o zaproponowany przez autora warunek pozwalający określać „pokrywające się" funkcje bazowe, tj. łączyć w bloki, na diagramach Veitcha-Karnaugha, nawet te kratki, które odpowiadają różnym wartościom minimalizowanej funkcji. Mimo heurystycznego charakteru zmodyfikowanego sposobu, jego główną zaletą jest to, że pozwala on (szczególnie przy minimalizacji funkcji arytmetyki *N*-wartościowej oraz funkcji logicznych z małą liczbą argumentów) na otrzymanie rozwiązań znacznie lepszych, niż za pomocą innych metod minimalizacji. Zmodyfikowany sposób autor wykorzystał do optymalizacji starszych i opracowania nowych, bardziej skomplikowanych projektów układów prądowych przeznaczonych do działania w arytmetyce *N*-wartościowej, *modulo N* i RNS. Między innymi, opracowano jednocyfrowe sumatory *N*-wartościowe i *modulo N*, jednocyfrowe bloki mnożące *N*-wartościowe i *modulo N*, wielooperandowe sumatory *modulo N*, a nawet konwertery liczb z systemu binarnego do RNS i na odwrót. Porównanie złożoności sprzętowej opracowanych układów z odpowiednimi układami zbudowanymi z klasycznych bramek CMOS wykazały znaczącą przewagę układów prądowych zarówno pod względem liczby wykorzystanych bramek, jak i tranzystorów oraz połączeń a nawet komórek pamięci ROM i sumatorów. Na przykład objętość bloku ROM w opracowanym układzie konwertera liczb z q-cyfrowego systemu RNS do a-bitowego systemu binarnego została zmnicjszona z $O(2^q)$ w znanym projekcie napięciowym do $O(q)$ w projekcie prądowym, a liczba sumatorów w konwerterze liczb a-bitowych do q-cyfrowych RNS została zmniejszona odpowiednio z $O(q\cdot(a\text{-}1))$ do $O\left(q\cdot a-\sum_{i=1}^{q}(\lfloor\log_2 m_i\rfloor)\right)$, gdzie m_i – to poszczególne moduły systemu RNS (m_i<12). Obiecujące wyniki porównania złożoności sprzętowej opracowanych przez autora układów świadczą o osiągnięciu drugiego celu pracy i potwierdzają udowodnienie drugiej tezy pracy.

Bibliografia

[A1] O. Maslennikow, A. Guziński, J. Kaniewski, R. Berezowski, „Rules of Current-mode Digital Circuit Design and Analysis", Proc. of the XXII Nat.Conf. on Circuit Theory and Electronic Networks, Warszawa-Stare Jablonki, Poland, 1999, s.149-154

[A2] O. Maslennikow, R. Berezowski, D. Gretkowski, A. Wąsik, „Projektowanie bloków sterowania wyspecjalizowanych równoległych urządzeń zbudowanych w oparciu o układy FPGA", Prace IV Konferencji Krajowej Reprogramowalne układy cyfrowe, RUC'2001, Szczecin, 2001, s.147-156

[A3] O. Maslennikow, R. Berezowski, P. Soltan, M. Rajewska, „Designing Prototype of the Spartan II FPGA Slice with the Current-Mode Gates", Proc. of the IEEE Int.Conf. on Circuits and Systems for Communication, ICCSC'2002, ST.-Petersburg, s.182-185

[A4] O. Maslennikow, P. Pawłowski, P. Sołtan, R. Berezowski, „Current-Mode Digital Gates and Circuits: Conception, Design and Verification", Proc. of the IEEE Int.Conf. on Electronic Circuits and Systems, ICECS'2002, Horwacja, Vol.2, s.623-626

[A5] O. Maslennikow, R. Berezowski, P. Sołtan, „Model komórki układu FPGA zbudowanego w oparciu o bramki prądowe", Prace V Konferencji Krajowej Reprogramowalne układy cyfrowe, RUC'2002, Szczecin, 2002, s.189-196

[A6] M. Białko, R. Berezowski, N. Maslennikowa, P. Sołtan, „Realizacja logiki szybkiego przeniesienia w prototypie prądowym układu FPGA SpartanII", Prace I Krajowej Konferencji Elektroniki, KKE'2002, Kołobrzeg, 2002, s.849-855

[A7] P. Sołtan, O. Maslennikow, R. Berezowski, M. Rajewska, „Automatyzacja procesu implementacji układów cyfrowych w technologii prądowych układów FPGA" Prace I Konferencji Krajowej Elektroniki, KKE'2002, Kołobrzeg, 2002, s.843-848.

[A8] M. Rajewska, R. Berezowski, „Prądowe sumatory i układy mnożące dla logiki wielowartościowej i arytmetyki resztowej", Proc. of the 5-th Int. Electronic and Telecommunication Conference of Students and Young Scientific Workers, SECON'2003, Warszawa, 2003

[A9] M. Białko, M. Rajewska, R. Berezowski, „Wykorzystanie języka VHDL do modelowania układów pracujących w logice wielowartościowej i w arytmetyce resztowej", Prace X Konferencji Krajowej Komputerowe wspomaganie badań naukowych, KOWBAN'2003

[A10] O. Maslennikow, M. Białko, P. Pawłowski, R. Berezowski, „Przerzutniki prądowe dla logiki wielowartościowej i arytmetyki resztowej", Prace II Konferencji Krajowej Elektroniki, KKE'2003, Kołobrzeg, 2003, s.725-730

[A11] R. Berezowski, M. Rajewska, „Projekt i weryfikacja praktyczna podstawowych bloków układów FPGA zbudowanych w oparciu o bramki prądowe", Prace III Konferencji Krajowej Elektroniki, KKE'2004, Kołobrzeg, 2004, s.339-344

[A12] O. Maslennikow, P. Pawłowski, N. Maslennikowa, R. Berezowski, „Current-Mode Multipliers for Multiple-Valued Logic and Residue Number System Arithmetic", Proc. of 2-th IEEE Int. Conf. on Circuits and Systems for Communication, ICCSC'2004, Moscow

[A13] M. Rajewska, O. Maslennikow, R. Berezowski, „Układy prądowe dokonujące konwersji liczb z systemu binarnego do systemu RNS i odwrotnie", Prace IV Konferencji Krajowej Elektroniki, KKE'2005, Kołobrzeg, 2005, s.591-596

[A14] M. Rajewska, R. Berezowski, O. Maslennikow, „Realizacja S-bloków systemu kryptograficznego DES na bramkach prądowych", Prace XII Konferencji Krajowej Komputerowe wspomaganie badań naukowych, KOWBAN'2005, Polanica-Zdrój, 2005, s.121-126

[A15] N. Maslennikowa, O. Maslennikow, R. Berezowski, J. P. Lienou, „Design of FPGA-based Multi-Operand Modular Adders For Residue Number System Converters", Proc. 13-th Int. Conf. On Mixed Design of Integrated Circuits and Systems, MIXDES'2006, s.264-268

[A16] O. Maslennikow, R. Berezowski, M. Rajewska, N. Maslennikowa, „Design of binary-to-residue and residue-to-binary number system converters with the current-mode gates", CADSM'2007, Lviv-Polyana, Ukraine,

[A17] O. Maslennikow, M. Rajewska, R. Berezowski, „Hardware Realization of the AES Algorithm S-Block Functions in the Current-Mode Gate Technology", CADSM'2007, Lviv-Polyana, Ukraine

[A18] R. Berezowski, O. Maslennikow, M. Rajewska, „Minimalizacja funkcji binarnych w algebrze bramek prądowych w oparciu o funkcje wzorcowe", Prace VI Konferencji Krajowej Elektroniki, KKE'200&, Darłówko, 2007

[1] I. J. Akuszskij, D. Judickij, „Maszynnaja arifmetika w ostatocznych klassach", Moskwa, Sowetskoje Radio, 1968

[2] G. Alia, E. Martinelli, „Designing multi-operand modular adders", El.Letters, Vol.32, No1., 1996

[3] K. Ariyama, H. Toyoshima, „Hardware Implementation of Chinese Remainder Theorem Using Redundant Binary Representation", Proc. VLSI Signal Processing, VIII, 1995, s.552-561

[4] R. Brayton, G. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli, „Logic minimization algorithms for VLSI synthesis", Kluver Academic Publishers, Boston, 1984

[5] R. Conway, J. Nelson, „New CRT-Based RNS Converter Using Restricted Moduli Set", IEEE Trans. Comp., Vol.52, No5, 2003, s.572-578

[6] M. R. Garey, D. S. Johnson, „Computers and Intractability: A guide to the Theory of NP-Completeness", W. H. Freeman and Co., San Francisco, 1979

[7] grant KBN 8T11 B 04214, „Cyfrowe układy pracujące w trybie prądowym dla mieszanych systemów analogowo-cyfrowych", 1998-1999

[8] grant KBN 7T11 B 00420, „Programowalne mieszane układy analogowo-cyfrowe pracujące w trybie prądowym", 2001-2003

[9] D. Gretkowski, A. Guziński, J. Kaniewski, O. Maslennikow, „VHDL models of digital combinatorical circuits on the current-mode gates", Proc. 6-th Int. Conf. Mixed design of integrated circuits systems, MIXDES'99 , Kraków, Poland, 1999, s.253-258

[10] D. Gretkowski, O. Maslennikow, „Projekt i realizacja kombinacyjnych i sekwencyjnych układów cyfrowych przełączanych prądem", Prace IV Konferencji Krajowej Reprogramowalne układy cyfrowe, RUC'2001, Szczecin, 2001, s.249-256

[11] Guziński, A. Kiełbasiński, „Current-Mode Digital Circuits Operating in Mixed Analog-Digital Systems", Bulletin of the Polish Academy of Sciences, Technical Sciences, Vol. 44, No. 2, 1996, s.193-198

[12] Guziński, P. Pawłowski, „Current-mode digital circuits for low-voltage mixed analog-digital systems", Proc. 6-th Int.Conf. Mixed design of integrated circuits systems, MIXDES'99, Kraków, Poland, 1999, s.369-372

[13] Guziński, P. Pawłowski, D. Czwyrow, J. Kaniewski, O. Maslennikow, N. Maslennikowa, D. Rataj, „Design of Digital Circuits with Current-Mode Gates",

Bulletin of the Polish Academy of Sciences, Technical Sciences, Vol.48, No1, 2000, s.73-91

[14] F. Hill, G. Peterson, Switching theory and logical design", Wiley, New York, 1981

[15] T. Kalganova, J.F. Miller, N. Lipnitskaya, „Multiple-Valued Combinational Circuits Synthesized Using Evolve Hardware Approach", Proc. Of the 7th Workshop on Post-binary Ultra Large Scale Integration Systems, Fukuoka, Japan, 1998

[16] R. Krishnan, J. Ehrenberg, G. Ray, „A core function based residue to binary decoder for RNS filterarchitecture", Circuits and Systems, 1989, s.837 – 840

[17] O. Maslennikow, „Approaches to Designing and Examples of Digital Circuits Based on the Current-Mode Gates", Data Recording, Storage & Processing, Vol.3, No2, 2001, s.84-98

[18] O. Maslennikow, „Designing Prototype of the Xilinx FPGAs Function Generator with the Current-Mode Gates", Proc. Int. Conf. ICSES'2001, Łódź, Poland, 2001, s.333-338

[19] O. Maslennikow, „Podstawy teorii zautomatyzowanego projektowania reprogramowalnych równoległych jednostek przetwarzających dla jednoukładowych systemów czasu rzeczywistego", Wydawnictwo Uczelniane Politechniki Koszalińskiej, Koszalin, 2004

[20] O. Maslennikow, D. Czwyrow, A. Guziński, J. Kaniewski, P. Pawłowski, „Digital circuits on the current-mode gates", Proc. of the 5-th Int.Conf. MIXDES'98, Łódź, Poland, 1998, s.103-108

[21] O. Maslennikow, N. Maslennikowa, A. Guziński, J. Kaniewski, P. Pawłowski, „Design of adders with current-mode gates", Proc. of the XXI Nat.Conf. on Circuit Theory and Elektronic Networks, Poznań, Poland, 1998, s.119-124

[22] O. Maslennikow, N. Maslennikowa, D. Gretkowski, A. Guziński, P. Pawłowski, „Designing basic blocks of FPGA cells with the current-mode gates", Proc. of the 8-th Int.Conf. on Mixed Design, MIXDES'2001, Zakopane, Poland, 2001, s.153-158

[23] O. Maslennikow, P. Sołtan P. „Automated Implementation of Digital Circuits in Current-Mode FPGA Chips", Proc. 7 Int.Conf. Expepience of Designing and Application of CAD Systems in Microelecronics, CADSM'2003, Lwow, 2003, s.223-225

[24] E. McCluskey, „Logic design principles", Prentice-Hall, Englewood Cliffs, 1986

[25] E. McCluskey, „Minimization of Boolean functions", The Bell System Technical Journal, Vol.35, s.1417-1444

[26] P. Pawłowski, „Ocena przydatności bramek cyfrowych pracujących w trybie prądowym w mieszanych systemach analogowo-cyfrowych", Rozprawa doktorska, Koszalin, 2004

[27] P. Pawłowski, „Projektowanie i symulacja wielowartościowych bramek cyfrowych", Prace XII Konferencji Krajowej Komputerowe wspomaganie badań naukowych, KOWBAN'2005, Polanica-Zdrój, 2005, s.179-184

[28] P. Pawłowski, A. Guziński, J. Kaniewski, O Maslennikow, D. Czwyrow, „Low-voltage current-mode digital circuits", Proc. of the XXI Nat.Conf. on Circuit Theory and Electronic Networks, Poznań, Poland, 1998, s.119-124

[29] S. J. Piestrak, „Design of High-Speed Residue-to-Binary Number System Converter Based on Chinese Remainder Theorem", Proceedings of the1994 IEEE International Conference on Computer Design: VLSI in Computer & Processors, 1994, s.508-511

[30] S. J. Piestrak, „Design of residue generators and multi-operand modular adders using carry-saved adders", IEEE Trans. Comp., Vol.43, 1991, s.68-77

[31] B. Premkumar, „Comments 'An RNS to Binary Converters in a Three Moduli Set with Common Factors", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 51, 2004

[32] W. Quine, „The problem of simplifying truth functions", American Mathematical Monthly, Vol.59, 1952, s.521-531

[33] M. Rajewska, „Model urządzenia arytmetyczno – logicznego (ALU) zbudowanego w oparciu o bramki prądowe", Koszalin, 2000

[34] R. Rudell, A. Sangiovanni-Vincentelli, „Multiple-valued minimization for PLA optimization", IEE Transactions on CAD/ICAS, Vol.CAD-6, No5, 1987, s.727-750

[35] N. S. Szabo, R. I. Tanaka, „Residue arithmetic and its applications to computer technology", New York, Mc Graw-Hill, 1967

[36] B. Schneier, „Kryptograafia dla praktyków", WNT, Warszawa, 2002

[37] M. Soderstrand, „A New Hardware Implementation of Modulo Adders for Residue Number Systems", Proc. IEEE 26th Midwest Symp. Circuits and Systems, 1983, s.432-415

[38] T. Temel, A. Morgul, „Implementation of Multi-Valued Logic Gates Using Full Current-Mode CMOS Circuits", Proc. Int. Conf. Eleco'2001, 2001

[39] Thoidis, D. Soudris, I. Karafyllidis, A. Thanailakis, T. Stouraitis, „Design Metodology of Multiple-valued logic voltage-mode storage circuits", Proc. IEEE Int. Symp. Circuits and Systems (ISCAS), Vol.2, 1998, s.125-128

[40] B. Turton, „Extending Quine-McCluskey for Exclusive-Or logic synthesis", IEEE Transaction on Education, Vol.39, No1, 1996, s.81-85

[41] W. Traczyk, „Układy cyfrowe. Podstawy teoretyczne i metody syntezy", WNT, Warszawa, 1986

[42] Z. Ulman, „Resztowe systemy liczbowe z quasi-podstawą", Zeszyty naukowe Politechniki Gdańskiej, Nr 560, 1998

[43] Site of IEEE CS Technical Committee on Multiple-Valued Logic: http://www.stfx.ca

[44] Site of IEEE International Symposium on Multiple-Valued-Logic: http://www.ee.pdx.edu

[45] Site of International Symposium on Multiple-Valued Logic: http://www.csr.uvic.ca

[46] Site of Multiple-Valued Logic Research Group: http://www.gscit.monash.edu.au

[47] Site of Universal Algebra and Multiple-Valued Logic: http://www.cms.math.ca

[48] Site of Vienna Group for Multiple-valued Logics (VGML): http://www.logic.at

Dodatek A: Spis rysunków

Dodatek B: Spis tabel