Wydział Elektroniki

PRACA DOKTORSKA

Akceleracja sprzętowa działań arytmetycznych w algorytmach oświetlenia globalnego

Tadeusz Tomczak

Promotor: dr hab. inż. Janusz Biernat, prof. P.Wr.

słowa kluczowe:
układy cyfrowe, resztowe systemy liczbowe,
algorytmy oświetlenia globalnego

krótkie streszczenie:
Praca obejmuje szereg zagadnień związanych ze sprzętową akceleracją algorytmów oświetlenia globalnego w układach FPGA z użyciem arytmetyki resztowej.

Wrocław 2007

Spis treści

1 Definicja tematyki i cel pracy **12**

1.1 Algorytmy oświetlenia globalnego i ich realizacje 12

1.2 Układy reprogramowalne 17

1.3 Resztowe systemy liczbowe 18

1.4 Teza i cel pracy 20

1.5 Struktura pracy 22

2 Podstawy teoretyczne **24**

2.1 Algorytmy oświetlenia globalnego 24

2.1.1 Rekursywne śledzenie promieni 25

2.1.2 Metody energetyczne 35

2.2 Resztowe systemy liczbowe 42

2.2.1 Wybrane zagadnienia teorii liczb 42

2.2.2 Konwersja pomiędzy RNS a systemem pozycyjnym 47

2.2.3 Resztowe układy arytmetyczne 50

2.2.4 Detekcja znaku w RNS 54

2.3 Hierarchiczne resztowe systemy liczbowe 63

2.3.1 Wprowadzenie 65

2.3.2 Konwersja pomiędzy HRNS a systemem pozycyjnym 67

2.4 Podsumowanie 82

3 Układy arytmetyki resztowej w FPGA **85**

3.1 Jednostki arytmetyczne w strukturach FPGA 86

3.1.1 Układ wytwarzania iloczynów częściowych 88

3.1.2 Sumator wstępny . . . 91
3.1.3 Generator wyniku . . . 98
3.1.4 Obszar i opóźnienie jednostek arytmetycznych . . . 108
3.2 Algorytm automatycznej generacji jednostek arytmetycznych . . . 121
3.2.1 Układ wytwarzania iloczynów częściowych i sumator wstępny . . . 122
3.2.2 Generator wyniku . . . 125
3.2.3 Kompletna jednostka arytmetyczna . . . 129
3.2.4 Złożoność obliczeniowa . . . 130
3.3 Podsumowanie . . . 132

4 Implementacja **135**
4.1 Algorytm automatycznej generacji jednostek arytmetycznych . . . 135
4.1.1 Implementacja algorytmu . . . 136
4.1.2 Parametry generowanych jednostek . . . 139
4.2 Jednostki arytmetyczne w strukturach FPGA . . . 148
4.2.1 Charakterystyki AT dotychczasowych rozwiązań . . . 149
4.2.2 Charakterystyki AT nowych jednostek . . . 154
4.2.3 Wykorzystanie okresowości potęg 2 modulo M_* . . . 168
4.3 Hierarchiczne RNS w strukturach FPGA . . . 170
4.3.1 Opis struktury jednostek . . . 171
4.3.2 Wyniki implementacji . . . 175
4.4 Procesor wspomagający obliczenia w AOG . . . 181
4.4.1 Założenia . . . 182
4.4.2 Struktura procesora . . . 184
4.4.3 Zastosowanie arytmetyki resztowej . . . 190
4.4.4 Implementacje algorytmów oświetlenia globalnego . . . 192
4.4.5 Wyniki implementacji . . . 196
4.5 Podsumowanie . . . 198

Wnioski i dalsze kierunki badań **202**

Wykaz oznaczeń

i, j, k	—	liczba całkowita nieujemna
$\mathbf{K}$	—	wektor kierunkowy prostej
$\mathbf{O}$	—	punkt początkowy półprostej
$\mathbf{A}, \mathbf{A}', \mathbf{A}'', \mathbf{J}, \mathbf{O}'$	—	punkty w przestrzeni $\mathbb{R}^3$
$\mathbf{E_1}, \mathbf{E_2}, \mathbf{T}, \mathbf{Q}, \mathbf{S}$	—	wektory w przestrzeni $\mathbb{R}^3$
(A_x, A_y, A_z)	—	współrzędne punktu w przestrzeni $\mathbb{R}^3$
u, v	—	współrzędne barycentryczne
t	—	liczba rzeczywista
$\mathbf{N}$	—	wektor normalny
$I(\mathbf{A}, \mathbf{A}')$	—	natężenie światła przekazywanego z punktu $\mathbf{A}'$ do punktu $\mathbf{A}$
$B(\mathbf{A}), \mathbf{B}$	—	promienistość w punkcie $\mathbf{A}$, wektor promienistości
$E(\mathbf{A}), \mathbf{E}$	—	promienistość własna w punkcie $\mathbf{A}$, wektor promienistości własnych
$F_{ij}, \mathbf{F}$	—	współczynnik konfiguracji pomiędzy płatami i, j, macierz współczynników konfiguracji
ρ	—	współczynnik odbicia
$\mathbf{P}$	—	macierz współczynników odbicia
$\xi_i(\mathbf{A})$	—	i–ta funkcja bazowa w punkcie $\mathbf{A}$
$\mathbf{\Xi}$	—	macierz splotu funkcji bazowych
n	—	liczba modułów w RNS
M_i	—	moduł w RNS
m_i	—	liczba bitów reprezentacji binarnej M_i
M	—	zakres dynamiczny RNS
m	—	liczba bitów reprezentacji binarnej M
R, T, U, V, W, X, Y, Z	—	liczby całkowite
H	—	suma iloczynów częściowych
$\mathbf{H}, \mathbf{R}, \mathbf{T}, \mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}$	—	wektory binarne reprezentujące $H, R, T, U, V, W, X, Y, Z$
$h_i, r_i, t_i, u_i, v_i, w_i, x_i, y_i, z_i$	—	i–ty bit wektora $\mathbf{H}, \mathbf{R}, \mathbf{T}, \mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}$
X_i, Y_i, Z_i, W_i	—	reszty X, Y, Z, W modulo M_i
$\mathbf{X}_i, \mathbf{Y}_i, \mathbf{Z}_i, \mathbf{W}_i$	—	wektory binarne reprezentujące X_i, Y_i, Z_i, W_i

$x_j^i, y_j^i, z_j^i, w_j^i$ — j–ty bit wektora $\mathbf{X}_i, \mathbf{Y}_i, \mathbf{Z}_i, \mathbf{W}_i$

$x_{k:\,j}^i$ — pole bitowe wektora $\mathbf{X}_i$ zawierające bity o indeksach od j do k, gdzie $j \leq k$

$\mathbf{a}, \mathbf{b}$ — wektory opisujące podział na pola wektorów wejściowych $\mathbf{X}_*, \mathbf{Y}_*$ układu wytwarzania iloczynów częściowych

$\mathbf{d}$ — wektor opisujący podział na pola wektora wejściowego $\mathbf{U}$ reduktora modulo

a_i, b_i, d_i — współrzędne wektorów $\mathbf{a}, \mathbf{b}, \mathbf{d}$

$a(i), b(i), d(i)$ — indeksy najwyższych bitów i–tych pól zdefiniowanych przez wektory $\mathbf{a}, \mathbf{b}, \mathbf{d}$

$X_*(i), Y_*(i), U(i)$ — wartości zapisane na i–tych polach $\mathbf{X}_*, \mathbf{Y}_*, \mathbf{U}$ zdefiniowanych przez a_i, b_i, d_i

$x_{j:i}^*$ — pole bitowe wektora $\mathbf{X}_*$ zawierające bity o indeksach od i do j, gdzie $i \leq j$

l — liczba dodatkowych składników sumowanych w jednostce arytmetycznej

σ_{jk} — iloczyn częściowy (wynik mnożenia pól o wartościach $X_*(j), Y_*(k)$)

MSB(σ_{jk}) — indeks najwyższego bitu wektora binarnego reprezentującego σ_{jk}

LSB(σ_{jk}) — indeks najniższego bitu wektora binarnego reprezentującego σ_{jk}

$\mathbf{c}$ — macierz definiująca sposób generowania iloczynów częściowych

c_{jk} — element macierzy $\mathbf{c}$ definiujący sposób wytwarzania iloczynu σ_{jk}

$\chi(a_j, b_k)$ — funkcja określająca wartości elementów macierzy $\mathbf{c}$

$\hat{H}$ — zbiór szerokości wektorów reprezentujących sumy iloczynów częściowych

$\mathbf{\Psi}$ — macierz opisująca kaskadę sumatorów w reduktorze modulo

$\mathbf{\Psi}_i$ — wektor opisujący i–ty poziom sumatorów w reduktorze modulo

ψ_j^i — rekord opisujący wektor j–tej sumy na i–tym poziomie reduktora modulo

$\mathbf{\Theta}$ — macierz opisująca strukturę sumatora wstępnego

$\mathbf{\Theta}_i$ — wektor opisujący i–ty poziom sumatora wstępnego

θ_j^i — rekord zawierający pełny opis j–tego sumatora na i–tym poziomie kaskady sumatorów

$\Upsilon\colon \{\upsilon = (\mathbf{a}, \mathbf{b}, \mathbf{c})\}$ — zbiór konfiguracji układu wytwarzania iloczynów częściowych i sumatora wstępnego

$\mathbf{D}$ — macierz opisująca strukturę kaskady reduktorów modulo

$\Lambda\colon \{\lambda = (\mathbf{D})\}$	—	zbiór konfiguracji kaskad reduktorów modulo
$\Omega\colon \{\omega = (\upsilon, \lambda)\}$	—	zbiór konfiguracji kompletnej jednostki arytmetycznej
$\Gamma(k)$	—	zbiór kompozycji liczby k
$\mathbf{\Gamma}^i$	—	wektor opisujący i–tą kompozycję k
F_i	—	i–ty element ciągu Fibonacciego
$\alpha, \beta, \mu, \delta$	—	liczby rzeczywiste
S	—	pole powierzchni
$\varphi(k)$	—	funkcja Eulera
g	—	generator grupy multiplikatywnej
$P(M_i), HP(M_i)$	—	okres i półokres potęg 2 modulo M_i
A_L, T_L	—	obszar i opóźnienie komórki zawierającej jedną tablicę LUT wraz z układami towarzyszącymi

Spis rysunków

2.1 Idea algorytmu śledzenia promieni. 26

2.2 Wyznaczanie punktu przecięcia według algorytmu z [MT97]. 28

2.3 Wyznaczanie punktu przecięcia według algorytmu z [SF01]. 29

2.4 Drzewo kd. 32

2.5 Wybrane położenia promienia przy przeglądaniu drzew kd. 34

2.6 Idea algorytmu rekursywnego śledzenia promieni. 34

2.7 Wyznaczanie współczynników kształtu metodą Nusselta. 39

2.8 Wyznaczanie współczynników kształtu metodą próbkowania półsześcianu. 40

2.9 Porównanie błędu obliczonych wartości promienistości w funkcji liczby iteracji dla różnych metod rozwiązywania układu równań oświetlenia [CW93]. 42

2.10 Implementacja sumatorów modulo w układach FPGA. 52

2.11 Implementacja układów mnożących modulo modulo w układach FPGA. 53

2.12 Jednostka mnożąca modulo wykorzystująca małe twierdzenie Fermata. 54

2.13 Zakresy dynamiczne RNS dla a) $Z = -2^{3k-1} + 2^{k-1}$, b) $Z = -2^{3k-1} + 2^{k}$. 57

2.14 Schemat układu detekcji znaku dla RNS $(2^k - 1, 2^k, 2^k + 1)$. 62

2.15 Przykład układu wyznaczania reszty słowa 18–bitowego modulo 109. 69

2.16 Układ wyznaczania reszty modulo 5,7 i 13 dla słowa 18–bitowego w kodzie U2. . . . 74

3.1 Struktura resztowych jednostek arytmetycznych. 87

3.2 Znaczenie elementów wektora $\mathbf{a}$. 89

3.3 Struktury sumatora wstępnego. 95

3.4 Struktury sumatora wstępnego z wykorzystaniem okresowości. 98

3.5 Sumator z wbudowaną pamięcią. 99

3.6 Liczba możliwych konfiguracji jednostki modulo w funkcji szerokości modułu . . . 131

3.7 Przykład jednostki arytmetycznej mnożenia akumulacyjnego modulo 109. 133

4.1 Czas wykonania prototypowej implementacji algorytmu w funkcji szerokości modułu 138

4.2 Zależność obszaru od opóźnienia dla wybranych modułów. 140

4.3 Zależność obszaru całej jednostki od obszaru UWIC i SW. 141

4.4 Minimalny obszar jednostki w funkcji modułu M_*. 142

4.5 Minimalna długość ścieżki krytycznej jednostki w funkcji modułu M_*. 143

4.6 Zależność obszaru i opóźnienia od szerokości modułu dla układów o minimalnym obszarze. 143

4.7 Zależność opóźnienia oraz obszaru od szerokości modułu dla układów o minimalnym opóźnieniu. 144

4.8 Pozycja jednostki o najmniejszym obszarze po implementacji względem pozycji układu wybranego przez algorytm w funkcji modułu M_*. 145

4.9 Zależność iloczynu AT od obszaru dla wybranych modułów. 147

4.10 Porównanie obszaru i opóźnienia dla różnych struktur układów mnożenia modulo. . 150

4.11 Porównanie obszaru i opóźnienia dla różnych struktur układów mnożenia modulo. . 151

4.12 Porównanie iloczynu AT dla różnych struktur układów mnożenia modulo. 152

4.13 Porównanie iloczynu AT^2 dla różnych struktur układów mnożenia modulo. 153

4.14 Porównanie zajmowanego obszaru z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną. 155

4.15 Porównanie długości ścieżki krytycznej z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną. 156

4.16 Porównanie iloczynu AT z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną. 157

4.17 Porównanie iloczynu AT^2 z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną. 158

4.18 Porównanie zajmowanego obszaru z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo. 159

4.19 Porównanie długości ścieżki krytycznej z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo. 160

4.20 Porównanie iloczynu AT z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo. 161
4.21 Porównanie iloczynu AT^2 z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo. 162
4.22 Porównanie zajmowanego obszaru z układem mnożącym wykorzystującym prawo różnicy kwadratów. 164
4.23 Porównanie długości ścieżki krytycznej z układem mnożącym wykorzystującym prawo różnicy kwadratów. 165
4.24 Porównanie iloczynu AT z układem mnożącym wykorzystującym prawo różnicy kwadratów. 166
4.25 Porównanie iloczynu AT^2 z układem mnożącym wykorzystującym prawo różnicy kwadratów. 167
4.26 Porównanie obszaru oraz opóźnienia w funkcji modułu dla układów o minimalnym obszarze wykorzystujących okresowość. 169
4.27 Porównanie obszaru oraz opóźnienia w funkcji modułu dla układów o minimalnym opóźnieniu wykorzystujących okresowość. 169
4.28 Porównanie charakterystyk AT różnych implementacji jednostki mnożenia akumulacyjnego. 179
4.29 Struktura procesora. 185
4.30 Struktura układu arytmetycznego jednostki obliczeniowej. 188
4.31 Struktura procesora wykorzystującego HRNS. 191
4.32 Porównanie częstotliwości taktowania różnych implementacji procesora wspomagającego AOG. 197

Wstęp

Modelowanie zjawisk fizycznych zachodzących w rzeczywistym świecie za pomocą maszyn cyfrowych ma niemal tak długą historię, jak same maszyny. Począwszy od pierwszych „mózgów elektronowych" używanych do obliczania trajektorii pocisków balistycznych, a kończąc na najnowszych badaniach nad kreowaniem rzeczywistości wirtualnej, podejmowane są nieustannie prace nad cyfrowym odwzorowaniem zjawisk zachodzących w istniejącym świecie. Ponieważ zdecydowaną większość informacji o otoczeniu uzyskujemy za pomocą wzroku, doskonałym przykładem jest tutaj grafika komputerowa. Próby generowania metodami cyfrowymi obrazów jak najbardziej zbliżonych do rzeczywistych widoków, nazywanych *obrazami fotorealistycznymi*, są przedmiotem intensywnych prac od lat 80 dwudziestego wieku. Niestety, złożoność modeli obliczeniowych wymaga maszyn o ogromnej wydajności, niedostępnych w owym czasie. Dopiero od kilku lat postęp technologiczny w dziedzinie wytwarzania układów scalonych wielkiej skali integracji oraz powstanie wydajnych algorytmów obliczeniowych umożliwiły generowanie obrazów fotorealistycznych na szeroką skalę.

Wytworzenie obrazu fotorealistycznego wymaga zbudowania modelu świata, zwanego *sceną*, a następnie symulowania rozkładu energii świetlnej. Niestety, niezwykle skomplikowane opisy zjawisk zachodzących podczas propagacji fal elektromagnetycznych wymuszają stosowanie uproszczonych algorytmów obliczeniowych modelujących zjawiska zachodzące podczas propagacji i odbić światła. Algorytmy te nazywane są *algorytmami oświetlenia*. Pierwszymi algorytmami stosowanymi w grafice komputerowej były algorytmy oświetlenia lokalnego (AOL), w których brano pod uwagę jedynie wzajemne położenie oświetlanego obiektu, źródła światła oraz obserwatora. Ich zaletą był niski koszt implementacji, natomiast uzyskiwany efekt pozostawiał wiele do życzenia. Urządzenia wykorzystujące AOL są do dziś używane na szeroką skalę w komercyjnych produktach.

Podstawową wadą AOL jest pominięcie wpływu oświetlenia światłem rozproszonym i odbitym od pozostałych elementów występujących na scenie. Powstały więc metody pozwalające na doskonalsze modelowanie rozkładu energii świetlnej. Algorytmy te, nazywane *algorytmami oświetlenia*

globalnego (AOG), wykorzystują aparat matematyczny pozwalający na opis zjawisk związanych z wzajemnym oświetlaniem się obiektów światłem odbitym. Istnieją dwie podstawowe techniki tworzenia AOG: rekursywne śledzenie promieni (*ang. recursive ray tracing*) i metody energetyczne (*ang. radiosity*). Pierwsza z nich wykorzystuje umowne promienie światła, które podlegają wszystkim zjawiskom związanym z odbiciem, załamaniem i propagacją w różnych ośrodkach. Ideą metod energetycznych jest potraktowanie światła jako energii wymienianej pomiędzy powierzchniami obiektów na scenie. Stosowany jest także szereg metod będących hybrydą obu pomysłów.

Główną zaletą AOG jest możliwość modelowania wielu zjawisk występujących w świecie rzeczywistym, takich jak cienie, odbicia zwierciadlane, rozlewanie barwy czy załamanie światła. Choć implementacje odpowiednich algorytmów charakteryzują się bardzo dobrą skalowalnością na maszynach równoległych, to wymagają one ogromnych mocy obliczeniowych. Dotychczas opracowano wiele implementacji AOG na różnorodnych platformach, zaczynając od rozwiązań wyłącznie programowych, także komercyjnych, poprzez klastry wieloprocesorowe, logikę rekonfigurowalną, procesory graficzne (*ang.* ***G****raphic* ***P****rocessing* ***U****nit, GPU*) aż do powstałych w ostatnim czasie procesorów specjalizowanych. Największą wydajność oferują rozwiązania wykorzystujące wspomaganie sprzętowe, niestety są one bardzo kosztowne. Inne metody zapewniają zbyt niską wydajność dla zastosowań interakcyjnych. Konieczne jest zatem poszukiwanie nowych technik akceleracji sprzętowej pozwalających na zachowanie wysokiej wydajności przy niewielkich kosztach.

W ciągu ostatnich lat daje się zaobserwować znaczący postęp w dziedzinie wytwarzania układów scalonych pozwalający na umieszczanie coraz bardziej złożonych systemów cyfrowych w jednym układzie. Wraz ze wzrostem złożoności systemów rośnie także czas potrzebny na powstanie projektu oraz wdrożenie produkcji, co stanowi poważną barierę hamującą dalszy rozwój. Dodatkowym utrudnieniem jest konieczność posiadania rozbudowanego zaplecza technologicznego, dostępnego jedynie dla największych producentów. W odpowiedzi na te problemy powstały *reprogramowalne układy cyfrowe*, których konfiguracja może być łatwo zmieniana.

Układy reprogramowalne zrewolucjonizowały metodologię projektowania i wytwarzania systemów cyfrowych, umożliwiając szybkie projektowanie systemów złożonych z milionów bramek przy minimalnym ryzyku i kosztach. Szczególnie ważne jest skrócenie czasu „od pomysłu do produktu” (*ang. time–to–market*) pozwalające na dotrzymanie kroku potrzebom rynku. Możliwość zmian funkcji układu w pracującym systemie umożliwia usuwanie błędów oraz wprowadzanie dodatkowej funkcjonalności bez potrzeby wykonywania kolejnej wersji układu. Cechy te powodują, że układy

reprogramowalne są coraz częściej stosowane w projektach.

Podstawową wadą układów reprogramowalnych jest mniejsza liczba użytecznych zasobów w stosunku do układów specjalizowanych (*ang.* ***A****pplication* ***S****pecific* ***I****ntegrated* ***C****ircuit*, *ASIC*). Z tego względu konieczne jest poszukiwanie technik pozwalających na konstrukcję złożonych systemów cyfrowych o dużej wydajności przy zachowaniu niewielkich wymagań odnośnie zajmowanego obszaru. Jedną z nich jest arytmetyka resztowa, pozwalająca na dekompozycję obliczeń na zbiór niezależnych kanałów operujących na niewielkich liczbach. Uzyskuje się dzięki temu zwiększenie częstotliwości taktowania układu przy jednoczesnym ograniczeniu zajmowanego obszaru i poboru mocy.

Rozdział 1

Definicja tematyki i cel pracy

Tematyka pracy obejmuje szereg zagadnień związanych z metodami sprzętowej akceleracji operacji obliczeniowych wykorzystywanych w implementacjach AOG. Rozprawa zawiera zarówno analizę samych AOG wraz z przykładami ich implementacji, jak i wyniki badań nad metodami zwiększania wydajności cyfrowych układów arytmetycznych ze szczególnym uwzględnieniem arytmetyki resztowej.

1.1 Algorytmy oświetlenia globalnego i ich realizacje

Generowanie obrazów fotorealistycznych z wykorzystaniem algorytmów oświetlenia globalnego było przedmiotem badań od 1980 r. Niestety, ogromne jak na możliwości ówczesnej technologii zapotrzebowanie na moce obliczeniowe spowodowało, że dopiero kilkanaście lat później pojawiły się pierwsze próby implementacji tego typu systemów na powszechnie dostępnych maszynach. Dający się obecnie zaobserwować gwałtowny wzrost zainteresowania tą tematyką spowodowany jest szerokim obszarem zastosowań (symulacja i badania naukowe, wizualizacja danych medycznych, modelowanie architektoniczne, projektowanie wspomagane komputerowo, zarządzanie i sterowanie, sztuka, rozrywka, reklama, itd.), atrakcyjnością ekonomiczną oraz bardzo wysoką jakością uzyskiwanych efektów.

Zadaniem AOG jest określenie rozkładu energii świetlnej w danym otoczeniu. Najczęściej stosowanymi modelami obliczeniowymi są metody energetyczne [DDM02], [CW93] i śledzenie promieni [WS01]. Pierwsza z nich doskonale symuluje efekty związane z wzajemnym oświetlaniem się przez obiekty na scenie (np.: rozlewanie barwy), domeną drugiej są natomiast wszelkie zjawiska związa-

ne z odbiciami i załamaniami promieni światła. Algorytm śledzenia promieni doskonale modeluje wszelkiego rodzaju odbicia i załamania światła, nie uwzględnia natomiast oświetlenia energią odbitą od powierzchni matowych, co zmusza do stosowania stałego członu związanego z oświetleniem otoczenia i powoduje powstawanie nienaturalnych, ostrych krawędzi cieni. Metoda wyznaczania promienistości pozbawiona jest tego mankamentu, jednak w podstawowej wersji ogranicza się do matowych powierzchni lambertowskich i pomija wszelkie zjawiska związane z odbiciem kierunkowym bądź przezroczystymi elementami sceny, w tym także z zakłóceniami ośrodka przenoszącego energię (np. mgły). W systemach oferujących najwyższą jakość stosowane są rozwiązania hybrydowe.

Modelowanie oświetlenia w metodzie rekursywnego śledzenia promieni sprowadza się do symulacji biegu umownych promieni światła i wyznaczeniu obiektów, na które te promienie padają. Podstawową operacją dla tego algorytmu jest wyznaczanie punktu przecięcia półprostej modelującej promień z najbliższym obiektem. Wymaga to obliczenia współrzędnych punktu przecięcia dla pewnego podzbioru obiektów występujących na scenie, a następnie wyboru punktu najbliższego obserwatora.

Ograniczenie liczności podzbioru obiektów, dla których obliczane są współrzędne punktu przecięcia, pozwala na zmniejszenie złożoności obliczeniowej. Uzyskuje się to m.in. za pomocą algorytmów hierarchicznego podziału przestrzeni [Bit99] pozwalających sprowadzić metodę śledzenia promieni do klasy złożoności $O(\log(n))$. Dla porównania, algorytmy stosowane w rozpowszechnionych systemach grafiki rastrowej mają złożoność $O(n)$. Ponieważ jednak koszt pojedynczej operacji w algorytmie śledzenia promieni jest dość wysoki, wzrost wydajności można zaobserwować dopiero dla skomplikowanych scen zawierających dużą liczbę elementów. Istotnym ograniczeniem śledzenia promieni jest także ogromne obciążenie magistrali komunikacji z pamięcią.

Rekursywne śledzenie promieni posiada wiele zalet w porównaniu z klasycznymi systemami grafiki rastrowej stosowanymi w sprzęcie powszechnego użytku. Należą do nich wspomniane wyżej znacznie wyższa jakość generowanych obrazów oraz łatwość renderowania skomplikowanych scen ze względu na logarytmiczną złożoność obliczeniową. Większość umownych promieni jest niezależna, umożliwia to zatem proste uaktualnianie dowolnego wycinka obrazu wyjściowego. Możliwa jest niezwykle efektywna skalowalność na maszynach równoległych, przy czym stosowane są dwa podejścia. Dla scen z niewielką liczbą elementów pojedynczy procesor odpowiedzialny jest za obliczenia dla pojedynczego promienia, natomiast dla skomplikowanych środowisk pojedynczy promień „wędruje” pomiędzy procesorami odpowiedzialnymi za poszczególne fragmenty sceny. Wszystkie

wspomniane cechy, w połączeniu z mniejszą złożonością obliczeniową w stosunku do metod energetycznych, powodują, że przez światowych liderów w dziedzinie produkcji sprzętu graficznego prowadzone są obecnie intensywne prace nad nową generacją procesorów graficznych wykorzystujących rekursywne śledzenie promieni [Gib06], [Hur05], [Sto06].

Idea metody energetycznej polega na wyznaczeniu promienistości (*ang. radiosity*), czyli strumienia energii emitowanego z danej powierzchni, dla każdego punktu na scenie. W rzeczywistych rozwiązaniach sprowadza się ona do aproksymacji powierzchni obiektów na scenie za pomocą płatów elementarnych (najczęściej płaskich, np.: trójkątów) i znalezienia wartości promienistości dla każdego z nich. Wymaga to wyznaczenia współczynników sprzężenia (*ang. form factors*) określających, jaka część energii opuszczającej płat i dociera do płatu j. Wartości promienistości dla każdego płatu elementarnego są rozwiązaniem układu równań liniowych określonego przez wektor promienistości własnych płatów (źródła światła), macierz współczynników odbicia płatów oraz macierz współczynników sprzężenia [CW93].

Można tu wyróżnić dwa podstawowe problemy wymagające dużych mocy obliczeniowych. Pierwszym z nich jest liczba zmiennych w układzie równań nierzadko przekraczająca 10^6, co czyni klasyczne metody rozwiązywania o złożoności $O(n^3)$ bezużytecznymi. Możliwe jest jednak zastosowanie relaksacyjnych metod rozwiązywania układów równań, które dla powstających układów równań zwracają zadowalające przybliżenie po niewielkiej liczbie iteracji. Drugim poważnym utrudnieniem jest fakt, iż wyznaczenie współczynników sprzężenia wymaga znalezienia funkcji widoczności, tzn. zbadania, czy pomiędzy każdą parą płatów znajdują się jakieś przeszkody ograniczające ich wzajemną widoczność. Funkcja ta może być wyznaczona przy użyciu sprzętowych układów bufora głębokości lub za pomocą rzucania promieni (*ang. ray casting*) [CW93], [Kel97].

Obie zaprezentowane metody generowania obrazów fotorealistycznych mają wiele cech wspólnych. Pomijając dużą złożoność obliczeniową, istotne z punktu widzenia implementacji są: dobra skalowalność na maszynach równoległych i intensywne wykorzystanie mnożenia akumulacyjnego (*ang. multiply–accumulate, MAC*). Mnożenie akumulacyjne jest podstawową operacją wykonywaną w procesie iteracyjnego rozwiązywania układu równań oświetlenia opisanego w pracy [CW93]. Także wśród wielu algorytmów obliczania punktu przecięcia prostej z trójkątem [Bad90], [BA88], [Eri97], [MT97], [SF01] w co najmniej dwóch [MT97], [SF01] mnożenie akumulacyjne jest podstawowym działaniem. Pozwala to na adaptację efektywnych sprzętowych struktur MAC opracowanych na potrzeby cyfrowego przetwarzania sygnałów (*ang. Digital Signal Processing, DSP*).

Implementacje algorytmów oświetlenia globalnego

Implementacje AOG są ściśle powiązane ze sprzętem. Specyfikowane są cztery klasy realizacji:

1. wieloprocesorowe superkomputery [HLG99],[Kre97],[PPL+99],[PMS+99], specjalizowane karty DSP [HA96] oraz klastry PC [WS01],[WSBW01],[WSB01],

2. procesory graficzne (*ang. Graphics Processing Unit, GPU*) - [CHH02],[PBMH02],[Pur04],

3. specjalizowane procesory - [KiSK+01],[KiSSO02],[Saa07],[SWS02], jedno komercyjne [Hal01], [CRR04],

4. logika rekonfigurowalna - [SL02],[TL01].

Wykorzystanie maszyn wieloprocesorowych dla potrzeb generowania obrazów realistycznych było do niedawna jedyną techniką oferującą wystarczającą moc obliczeniową. W pracy [Kre97] podano przegląd takich rozwiązań. Ich wspólną cechą jest przede wszystkim wysoki koszt oraz czas reakcji pozwalający na pracę interakcyjną. Są to jednak rozwiązania oparte o standardowe, dostępne w handlu superkomputery, a różnice ograniczają się do zaimplementowanych algorytmów. Z nieco nowszych osiągnięć w tej dziedzinie warto wspomnieć implementację na 60-procesorowym superkomputerze z współdzieloną pamięcią SGI Origin 2000 [PMS+99]. Wieloprocesorowe superkomputery są jednak bardzo kosztowne i dostępne dla wąskiego grona użytkowników. Alternatywą może być zaprezentowany w pracach [WS01],[WSB01],[WSBW01] system zbudowany z popularnych komputerów osobistych wykorzystujący obecne w dzisiejszych procesorach zmiennoprzecinkowe instrukcje wektorowe [AMD00],[RPK00]. Interesujące są także wyniki badań [CC02] zmierzających w kierunku zmniejszenia złożoności obliczeniowej dzięki uwzględnieniu własności percepcyjnych ludzkiego narządu wzroku.

Metody polegające na wykorzystaniu obecnie produkowanych GPU (*ang. Graphic Processing Unit*) do niedawna nie dawały zadowalających wyników, zarówno ze względu na niewielką wydajność, jak i niską jakość otrzymywanych obrazów, m.in. z powodu zbyt małego zakresu dynamicznego stosowanych formatów reprezentacji danych. Poprawę efektów przyniosła dopiero ewolucja architektury GPU w kierunku wydajnych procesorów wektorowych z rozbudowaną, ortogonalną listą rozkazów [PBMH02]. Wyczerpujący przegląd aplikacji wykorzystujących nowoczesne GPU do implementacji różnorodnych problemów obliczeniowych, w tym AOG, zaprezentowano w pracy [OLG+05].

Znane są także próby wspomagania procesu wyznaczania przecięć promieni z obiektami przy użyciu matryc FPGA [SL02],[TL01]. Ze względu na niedostępność w owym czasie układów o wystarczającej pojemności są to jednak rozwiązania o niskiej wydajności oraz ograniczonym zbiorze zaimplementowanych operacji.

Najlepsze efekty uzyskano projektując specjalizowane procesory optymalizowane dla potrzeb algorytmów oświetlenia globalnego. Głównymi problemami utrudniającymi efektywne implementacje są: intensywne używanie rekursji, duża ilość skomplikowanych obliczeń wysokiej precyzji oraz częsty i nieregularny dostęp do pamięci wymuszający stosowanie magistral o wielkiej przepustowości. Można tu porównać następujące rozwiązania - układ SaarCOR opracowany na Uniwersytecie Saarland [Saa07],[SWS02], architekturę 3DCGiRAM [KiSK$^+$01],[KiSSO02] oraz komercyjny procesor AR350 firmy Advanced Rendering Technology [Hal01], [CRR04]. Pierwsze dwa z powyższych są przeznaczone do generowania obrazów w czasie rzeczywistym, ostatni służy natomiast jedynie jako jednostka wspomagająca tworzenie obrazów statycznych. Należy jednak zaznaczyć, że zarówno SaarCOR, jak i 3DCGiRAM nie zostały zaimplementowane. Dla układu SaarCOR wykonano prototyp o ograniczonej wydajności z użyciem układów FPGA [Sch06].

We wszystkich przypadkach głównym zadaniem projektowanych procesorów jest wspomaganie algorytmów wykorzystujących śledzenie promieni. Podstawową jednostkę stanowi więc blok generowania promieni i wyznaczania ich przecięć z obiektami. Układy SaarCOR i 3DCGiRAM obsługują tylko trójkąty, a użycie pamięci podręcznej umożliwia ominięcie ograniczeń związanych z obciążeniem magistrali komunikacji z pamięcią zewnętrzną. Dla układu AR350 istnieje możliwość ograniczonego wspomagania obliczeń algorytmów energetycznych, ponieważ jednostka testowania przecięć może posłużyć do wyznaczania współczynników sprzężenia metodami probabilistycznymi. Jest to jednak algorytm mało efektywny i podatny na zakłócenia (szumy kwantyzacji). Lepsze wyniki uzyskano dla projektowanego układu 3DCGiRAM, lecz nie został on jeszcze zaimplementowany. Wspomniane wyżej procesory można także łatwo łączyć w wieloelementowe siatki, przy czym osiągany jest prawie liniowy wzrost wydajności.

Wydajność niezbędną do renderingu w czasie rzeczywistym zapewniają jedynie implementacje wykorzystujące wieloprocesorowe superkomputery lub specjalizowane procesory. Przydatność praktyczna systemów implementowanych z użyciem superkomputerów jest ograniczona, głównie ze względu na wysokie koszty, wymiary i pobór mocy uniemożliwiające przenośność. Wykorzystanie AOG na szeroką skalę stanie się więc możliwe w momencie wprowadzenia niewielkich, wydajnych i

pobierających małą moc procesorów dedykowanych.

Jedynym rozwiązaniem o wydajności pozwalającej na generowanie obrazów w czasie rzeczywistym jest procesor opisany w [Sch06], który wymaga implementacji ASIC. Zaimplementowany w układach FPGA prototyp nie zapewnia wystarczającej wydajności. Nie istnieje zatem obecnie rozwiązanie procesora wspomagającego obliczenia AOG dostępnego dla szerokiego kręgu użytkowników. Ograniczeniem dotychczasowych projektów jest też brak kompleksowego rozwiązania procesora pozwalającego na wspomaganie obliczeń zarówno dla algorytmów śledzenia promieni, jak i metod energetycznych. Najwyższą jakość generowanego obrazu oferują rozwiązania hybrydowe korzystające z obu wymienionych metod. Konieczne jest zatem opracowanie technik pozwalających na implementację układów sprzętowego wspomagania AOG bez konieczności posiadania kosztownego zaplecza technologicznego.

1.2 Układy reprogramowalne

Układy specjalizowane (ASIC) umożliwiają konstrukcję złożonych systemów cyfrowych o wysokiej wydajności. Niestety, do ich wytwarzania konieczne jest posiadanie niezwykle kosztownych urządzeń i surowców. Co więcej, długi cykl produkcyjny i niemożność wprowadzania modyfikacji gotowego produktu przyczyniają się do wzrostu ryzyka oraz kosztów projektowych. W związku z tym doskonałym rozwiązaniem dla niewielkich jednostek produkcyjno–badawczych jest stosowanie logiki rekonfigurowalnej.

Obecnie produkowane układy reprogramowalne [Xil03a, Xil03b, Xil04, Xil05] pozwalają na konstrukcję systemów cyfrowych o złożoności liczonej w milionach bramek i taktowanych częstotliwościami sięgającymi setek MHz. Dostępne są dla nich wysokiej klasy narzędzia projektowania wspomaganego komputerowo (*ang.* ***C****omputer* ***A****ided* ***D****eisgn, CAD*). Istnieje rozwinięty rynek gotowych podzespołów systemów cyfrowych w postaci syntezowalnych opisów, nazywany rynkiem *własności intelektualnych* (*ang.* ***I****ntelectual* ***P****roperty, IP*). Umożliwia to konstrukcję kompletnych systemów cyfrowych o dużej złożoności w krótkim czasie i przy zachowaniu niskich kosztów projektowych. Masowa produkcja układów reprogramowalnych powoduje spadek kosztów pojedynczego układu do poziomu umożliwiającego stosowanie ich w urządzeniach produkowanych seryjnie. Wiele układów reprogramowalnych może być rekonfigurowanych w pracującym systemie (*ang.* ***I****n* ***S****ystem* ***P****rogramming, ISP*), dzięki czemu można poszerzać funkcjonalność systemu lub usuwać błędy po-

wstałe w fazie projektowania. Cechy te sprawiają, że logika reprogramowalna jest stosowana w różnorodnych aplikacjach.

Układy reprogramowalne znajdują szczególnie duże zastosowanie w systemach cyfrowego przetwarzania sygnałów. Sprzętowa implementacja wielu algorytmów pozwala na osiągnięcie wydajności wielokrotnie przewyższającej najszybsze procesory DSP przy podobnych nakładach. Dzięki łatwości stosowania układów reprogramowalnych proces projektowania sprzętu stał się obecnie podobny do procesu tworzenia oprogramowania.

Istnieje wiele odmian układów reprogramowalnych różniących się strukturą wewnętrzną i udostępnianymi zasobami. Struktura wewnętrzna większości układów reprogramowalnych zawiera zbiór jednostek funkcyjnych łączonych za pomocą programowalnej sieci połączeń. W procesie programowania układu przeprowadza się rekonfigurację sieci połączeń oraz funkcji realizowanych przez jednostki funkcyjne. Każda jednostka funkcyjna jest elementem niepodzielnym i pozwala na implementację nietrywialnej funkcji logicznej. Z tego powodu klasyczne metody minimalizacji logicznej na poziomie prostych bramek nie pozwalają na osiągnięcie optymalnej struktury układu.

Struktura układów reprogramowalnych z reguły zawiera elementy umożliwiające efektywną implementację wielu podstawowych bloków logicznych (układy arytmetyczne, rejestry, pamięci, itd.). Proces projektowania efektywnego systemu cyfrowego wymaga stosowania tych podstawowych elementów jako jednostek niepodzielnych. Umożliwia to narzędziom syntezy użycie wydajnych implementacji dedykowanych dla konkretnej rodziny układów reprogramowalnych. Struktura, parametry i ograniczenia podstawowych bloków narzucają istotne ograniczenia na strukturę i parametry systemu docelowego. Konieczne jest zatem zaprojektowanie konkretnego systemu w sposób umożliwiający efektywne wykorzystanie bloków logicznych dedykowanych dla konkretnej rodziny układów reprogramowalnych.

1.3 Resztowe systemy liczbowe

Wydajność systemu cyfrowego implementującego algorytmy obliczeniowe zależy od szybkości realizacji sprzętowej operacji arytmetycznych. W większości systemów cyfrowych liczby są reprezentowane za pomocą wektorów bitowych. Liczbę bitów danego wektora nazywa się jego *szerokością*. Zgodnie z modelem teoretycznym opracowanym przez Winograda szybkość wykonywania operacji arytmetycznych jest funkcją szerokości operandów. Zwiększenie wydajności systemu cyfrowego jest

zatem możliwe przez ograniczenie liczby bitów wektorów reprezentujących dane. Jest to możliwe dzięki użyciu resztowych systemów liczbowych (*ang.* ***R****esidue* ***N****umber* ***S****ystem, RNS*).

Reprezentacją liczby całkowitej w RNS jest wektor jej reszt modulo liczby stanowiące *bazę* systemu, nazywane *modułami*. Moduły bazy muszą być liczbami wzajemnie pierwszymi. Szerokość przedziału liczb, dla których istnieje unikatowa reprezentacja w RNS, nazywa się *zakresem dynamicznym* systemu. Zakres dynamiczny jest określony przez iloczyn modułów. Wartość pojedynczego modułu może być znacznie mniejsza od zakresu dynamicznego całego systemu. Na tak zdefiniowanej reprezentacji dodawanie, odejmowanie i mnożenie może być wykonywane dla wszystkich reszt równocześnie i niezależnie. Umożliwia to dekompozycję pojedynczego kanału obliczeniowego na szereg równoległych kanałów operujących na wektorach o znacznie mniejszej szerokości. Dzięki temu uzyskuje się zwiększenie szybkości układu przy równoczesnym ograniczeniu zajmowanego obszaru i poboru mocy.

Niestety, resztowe systemy liczbowe mają szereg cech utrudniających szerokie stosowanie w systemach obliczeniowych ogólnego przeznaczenia. Podstawowymi problemami są konieczność stosowania *konwersji* pomiędzy RNS a systemem pozycyjnym oraz niezwykle kosztowne algorytmy wykonywania niektórych operacji arytmetycznych. Korzyści z użycia RNS są najbardziej widoczne w systemach, gdzie podstawową operacją jest wielokrotnie wykonywane mnożenie akumulacyjne. Przykładem mogą być algorytmy DSP takie jak filtrowanie cyfrowe czy transformaty numeryczne.

Wydajność jednostki arytmetycznej wykorzystującej RNS jest ograniczona przez czas wykonania operacji w najwolniejszym kanale. Najkrótsze opóźnienia są wprowadzane przez układy dla modułów niewielkich. Istnieją także specyficzne wartości modułów, np. 2^k, dla których możliwe jest zbudowanie efektywnych układów arytmetycznych pomimo dużej wartości modułu. W przypadku konieczności konstruowania RNS o dużym zakresie dynamicznym użycie niewielkich modułów powoduje wzrost liczby modułów stanowiących bazę systemu. Dodatkowo wymaganie wzajemnej względnej pierwszości modułów znacznie utrudnia wybór bazy spośród liczb, dla których znane są efektywne implementacje jednostek arytmetycznych. Duża liczba modułów utrudnia także wykonywanie niektórych operacji arytmetycznych oraz konwersję pomiędzy RNS a systemem pozycyjnym.

Pomysłem pozwalającym na zmniejszenie szerokości kanałów obliczeniowych w RNS przy zachowaniu prostych algorytmów konwersji jest użycie hierarchicznych resztowych systemów liczbowych (*ang.* ***H****ierarchical* ***R****esidue* ***N****umber* ***S****ystem, HRNS*) [AJ68], [Yas92], [SA99]. W HRNS kolejne reszty są zapisywane z użyciem RNS o zakresie dynamicznym znacznie mniejszym od zakresu ca-

łego systemu. Konwersja może być przeprowadzana wielopoziomowo, z wykorzystaniem systemów pośrednich o małym zakresie dynamicznym i niewielkiej liczbie modułów. Dzięki temu możliwe jest zmniejszenie złożoności konwerterów. Obliczenia w poszczególnych kanałach przeprowadzane są dla modułów o niewielkich wartościach. Niestety, tematyka dotycząca HRNS jest dość słabo zbadana, a dobór bazy systemu jest problemem nietrywialnym.

Niezależnie od metod wyboru bazy RNS wydajność systemu może być zwiększona przez użycie wydajnych struktur resztowych jednostek arytmetycznych dla poszczególnych modułów. Problem konstrukcji wydajnych resztowych jednostek arytmetycznych był przedmiotem intensywnych badań. Główny nacisk położono na układy realizujące dodawanie i mnożenie modulo, zarówno dla dowolnej postaci modułu, jak i dla przypadków szczególnych, np.: liczb postaci $2^n \pm 1$. Niestety, większość opracowanych rozwiązań jest dedykowana dla układów ASIC i z tego względu ich implementacja w układach reprogramowalnych jest nieefektywna. Oczywiście istnieją publikacje na temat wykorzystania arytmetyki resztowej w układach reprogramowalnych [MBGT01], [RGMBL02], [RGTL03], [Beu03], [Ven96], [BM04], [MFAA98], [Beu02]. Zaobserwowano w nich, że stosowanie RNS w układach reprogramowalnych pozwala ominąć niektóre ograniczenia tych układów, zwiększyć wydajność operacji arytmetycznych oraz zmniejszyć zajmowany obszar.

Podstawową wadą dotychczasowych struktur resztowych jednostek arytmetycznych dedykowanych dla układów FPGA jest wykorzystywanie pamięci o dużej pojemności. Pomimo łatwej implementacji pamięci w układach FPGA wykładnicza zależność zajmowanego obszaru od szerokości wektorów reprezentujących reszty powoduje, że rozwiązania te są bardzo kosztowne dla modułów o średniej i dużej szerokości. Stanowi to poważną barierę utrudniającą konstruowanie resztowych jednostek arytmetycznych dla dużych zakresów dynamicznych. Należy zatem opracować struktury resztowych jednostek arytmetycznych przeznaczonych dla układów reprogramowalnych, które będą wolne od tego ograniczenia.

1.4 Teza i cel pracy

Na podstawie analizy dotychczasowych rozwiązań sprzętowego wspomagania AOG oraz metod zwiększania wydajności układów arytmetycznych zaobserwowano, że:

1. Znane rozwiązania układów sprzętowego wspomagania AOG mają zbyt małą wydajność lub charakteryzują się wysokimi kosztami. Zakres realizowanych zadań jest ograniczony do wspo-

magania algorytmów wykorzystujących śledzenie promieni. Dodatkowo ich implementacja wymaga posiadania kosztownego zaplecza technologicznego niedostępnego dla małych i średnich jednostek produkcyjno–badawczych.

2. Algorytmy oświetlenia globalnego są bardzo dobrze skalowalne na maszynach równoległych. Korzystne jest zatem dążenie do zmniejszenia obszaru zajmowanego przez jednostki obliczeniowe, co pozwoli na umieszczenie większej ich liczby w pojedynczym układzie i w konsekwencji spowoduje wzrost wydajności.

3. Wykorzystanie układów reprogramowalnych umożliwia konstrukcję złożonych systemów cyfrowych bez konieczności posiadania kosztownego wyposażenia. Powstały system charakteryzuje się niskim kosztem, podatnością na modyfikacje i krótkim czasem od pomysłu do projektu. Układy reprogramowalne stanowią zatem bardzo atrakcyjną platformę dla wielu systemów cyfrowych, także o dużej złożoności.

4. Zasoby udostępniane przez układy reprogramowalne pozwalają na realizację układów o złożoności liczonej w milionach bramek, jednak w porównaniu z układami ASIC są nadal wielokrotnie mniejsze. Z tego powodu konieczne jest poszukiwanie metod pozwalających na ograniczenie zajmowanego obszaru przy zachowaniu wysokiej wydajności. W układach arytmetycznych można to osiągnąć przez użycie arytmetyki resztowej.

5. Dotychczasowe struktury resztowych układów arytmetycznych w FPGA charakteryzują się nieefektywną implementacją dla średnich i dużych wartości modułu, co utrudnia konstrukcję RNS o dużym zakresie dynamicznym. Zakres dynamiczny wymagany w AOG jest rzędu kilkudziesięciu bitów, konieczne jest zatem opracowanie nowych, dedykowanych dla FPGA rozwiązań sprzętowej implementacji resztowych operacji arytmetycznych, które będą użyteczne w szerokim zakresie wartości modułu.

6. Skonstruowanie wydajnej resztowej jednostki arytmetycznej wymaga ograniczenia operacji kosztownych w RNS (np.: konwersji czy dzielenia) w stosunku do liczby dodawań i mnożeń. W dotychczasowych konstrukcjach konieczny był kompromis pomiędzy wydajnością toru obliczeniowego w RNS a kosztem operacji trudnych. Rozwiązaniem mogą być HRNS, które dzięki wielopoziomowej, uproszczonej strukturze konwerterów umożliwiają użycie niewielkich modułów, dla których istnieją efektywne implementacje jednostek arytmetycznych.

7. Działania w kanałach resztowych powinny być ograniczone do dodawania i mnożenia. Wiele problemów AOG spełnia te kryteria. W niektórych przypadkach konieczna jest reorganizacja obliczeń wyprowadzająca operacje trudne poza tor resztowy, co umożliwi ich efektywną implementację przy zachowaniu korzyści wynikających z stosowania RNS.

Powyższe wnioski pozwalają postawić tezę, że:

Wykorzystanie arytmetyki resztowej w układach sprzętowego wspomagania obliczeń algorytmów oświetlenia globalnego z użyciem logiki reprogramowalnej umożliwia zwiększenie wydajności.

W celu wykazania tezy rozwiązano szereg problemów cząstkowych, takich jak:

- opracowanie nowej metody generowania struktur resztowych jednostek arytmetycznych w układach FPGA,
- zastosowanie nowej klasy HRNS z bazą złożoną z modułów postaci $2^k, 2^k \pm 1$ w celu uproszczenia struktury konwerterów przy zachowaniu wysokiej wydajności układów arytmetycznych,
- zaprojektowanie architektury układu sprzętowego wspomagania AOG, w którym dzięki wyodrębnieniu operacji trudnych w RNS możliwe jest zwiększenie wydajności,
- implementacja kilku kluczowych operacji występujących w AOG z użyciem RNS.

1.5 Struktura pracy

Przedmiotem pracy są techniki sprzętowej akceleracji obliczeń arytmetycznych z wykorzystaniem resztowych systemów liczbowych oraz przykład ich zastosowania w układach sprzętowego wspomagania obliczeń AOG. Platformą docelową proponowanych rozwiązań są reprogramowalne matryce bramkowe (*ang. **F**ield **P**rogrammable **G**ate **A**rray, FPGA*). Eksperymenty przeprowadzono w układach rodziny Spartan 2 firmy Xilinx.

Zaproponowano metodę konstrukcji resztowych jednostek arytmetycznych charakteryzujących się większą wydajnością oraz mniejszym zajmowanym obszarem od dotychczasowych rozwiązań. Przedstawiono sposób doboru bazy resztowych systemów liczbowych pozwalający na uproszczenie struktur konwerterów pomiędzy systemem liczbowym resztowym a pozycyjnym przy zachowaniu wysokiej wydajności układów arytmetycznych. Podano także przykład implementacji kilku istotnych

operacji występujących w AOG z użyciem opisanych technik. Istotną cechą zaprezentowanych metod sprzętowej akceleracji obliczeń jest ich uniwersalny charakter umożliwiający zastosowanie w różnorodnych aplikacjach, także spoza obszaru AOG.

Opis realizacji celu pracy przedstawiają kolejne rozdziały rozprawy. W rozdziale drugim zawarto niezbędne w dalszych rozważaniach podstawy teoretyczne z zakresu AOG i arytmetyki resztowej. Zaproponowano nowy algorytm szybkiej detekcji znaku dla wybranej klasy RNS. Przedstawiono analizę własności nowej klasy hierarchicznych resztowych systemów liczbowych oraz podano równania umożliwiające konstrukcję konwerterów o niewielkiej złożoności. W rozdziale trzecim zaprezentowano nowe struktury resztowych jednostek arytmetycznych wraz z opisem metody ich tworzenia. Przedmiotem rozdziału czwartego jest krytyczna analiza wyników implementacji resztowych jednostek arytmetycznych w FPGA, resztowych kanałów obliczeniowych wykorzystujących nową klasę HRNS, algorytmu generacji resztowych jednostek arytmetycznych oraz procesora wspomagającego obliczenia w AOG.

Rozdział 2

Podstawy teoretyczne

2.1 Algorytmy oświetlenia globalnego

Modelowanie oświetlenia w algorytmach oświetlenia globalnego sprowadza się do rozwiązania równania oświetlenia zaproponowanego po raz pierwszy w pracy [Kaj86]. Równanie to jest adaptacją formuł stosowanych w zagadnieniach wymiany ciepła [SH81]. Równanie oświetlenia, nazywane także równaniem renderingu, jest uproszczeniem w stosunku do równań Maxwella opisujących propagację fal elektromagnetycznych i nie obejmuje np. zależności fazowych. Opisuje ono przepływ energii pomiędzy punktami na powierzchniach obiektów sceny. Współrzędne dowolnego punktu mogą być interpretowane jako wierzchołek wektora o ustalonym punkcie zaczepienia, dlatego też w dalszej części pracy punkty są traktowane tak, jak wektory. W równaniu oświetlenia przyjęto, że promieniowanie docierające z punktu $\mathbf{A}'$ do punktu $\mathbf{A}$ składa się z:

- promieniowania emitowanego z punktu $\mathbf{A}'$,
- promieniowania pochodzącego z wszystkich pozostałych punktów $\mathbf{A}''$ i odbitego od $\mathbf{A}'$.

Treścią równania oświetlenia jest określenie *natężenia promieniowania* I, nazywanego także *intensywnością promieniowania*. Przez natężenie promieniowania rozumie się moc promieniowania wysyłaną w jednostkowy kąt bryłowy. Jednostką natężenia promieniowania jest $\left[\frac{W}{sr}\right]$. Grafika komputerowa dotyczy zjawisk związanych z propagacją światła widzialnego, dlatego też natężenie promieniowania jest określane dla widzialnej części widma fal elektromagnetycznych i w związku z tym nazywane niekiedy *natężeniem światła*.

Zgodnie z modelem zaproponowanym w pracy [Kaj86], natężenie promieniowania przekazywanego z punktu $\mathbf{A}'$ do punktu $\mathbf{A}$ jest opisane jako

$$I(\mathbf{A}, \mathbf{A}') = g(\mathbf{A}, \mathbf{A}') \left[\epsilon(\mathbf{A}, \mathbf{A}') + \int_S f(\mathbf{A}, \mathbf{A}', \mathbf{A}'') I(\mathbf{A}', \mathbf{A}'') d\mathbf{A}'' \right]. \tag{2.1}$$

Czynnik $g(\mathbf{A}, \mathbf{A}')$, zwany *czynnikiem geometrycznym* (*ang. geometric term*), jest funkcją wzajemnego położenia punktów $\mathbf{A}$ i $\mathbf{A}'$. Wartość $\epsilon(\mathbf{A}, \mathbf{A}')$ opisuje natężenie światła emitowanego z punktu $\mathbf{A}'$. Natężenie to jest różne od zera np. dla źródeł światła. Funkcja $f(\mathbf{A}, \mathbf{A}', \mathbf{A}'')$ zależy od natężenia światła z punktu $\mathbf{A}''$ odbitego za pośrednictwem punktu $\mathbf{A}'$ do punktu $\mathbf{A}$. Całkowanie w równaniu (2.1) przeprowadzane jest po wszystkich powierzchniach występujących na scenie.

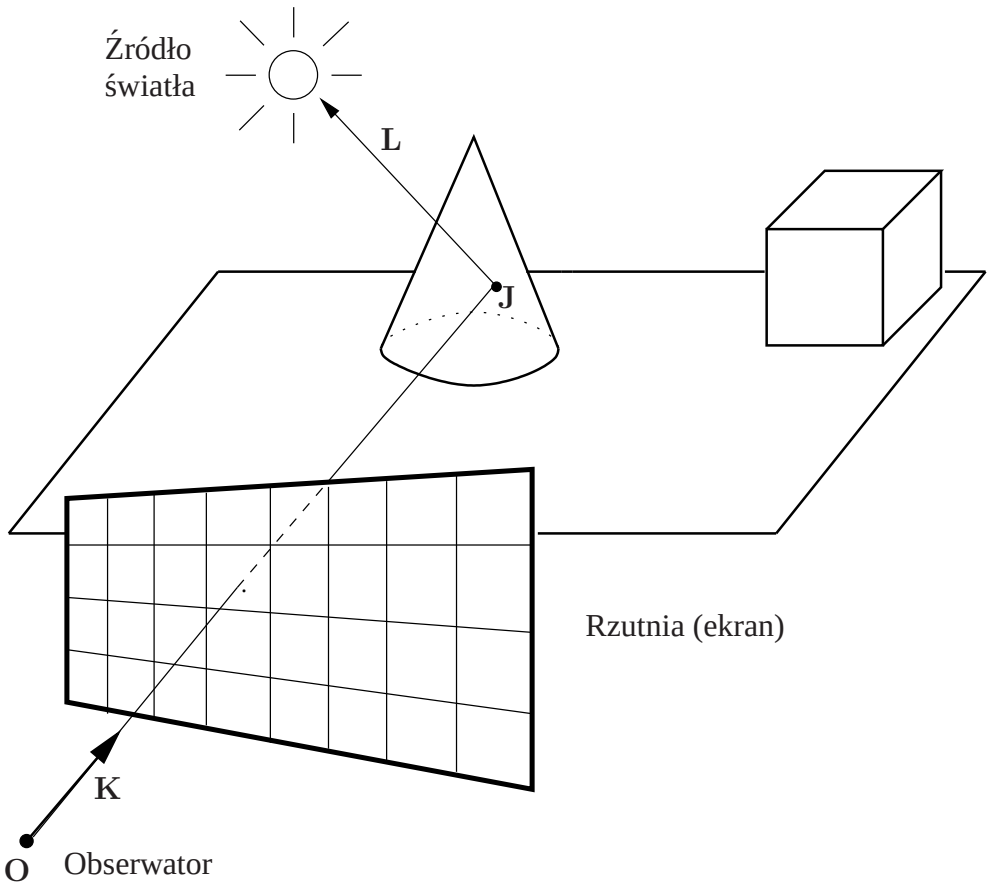
Równanie (2.1) opisuje statyczny rozkład energii świetlnej w danym otoczeniu. Analityczne rozwiązanie jest możliwe jedynie w przypadku bardzo prostych scen, stosowany jest zatem szereg metod numerycznych pozwalających znaleźć rozwiązanie przybliżone. W grafice komputerowej stosowane są dwie podstawowe koncepcje przybliżonego rozwiązania równania oświetlenia: rekursywne śledzenie promieni i metody energetyczne. Metody te wzajemnie się uzupełniają, dlatego w systemach oferujących najwyższą jakość stosowane są rozwiązania hybrydowe, wykorzystujące obie koncepcje równocześnie.

2.1.1 Rekursywne śledzenie promieni

W algorytmie rekursywnego śledzenia promieni transport energii świetlnej jest modelowany za pomocą umownych *promieni światła*. Promienie te podlegają zjawiskom odbicia, załamania i pochłaniania przez otoczenie, co umożliwia modelowanie wielu zjawisk obserwowanych w przyrodzie (np. odbicia lustrzane, cienie, mgły). Rekursywne śledzenie promieni jest rozwinięciem metody śledzenia promieni, której podstawy po raz pierwszy sformułował Albrecht Dürer, malarz żyjący na przełomie XV i XVI wieku. Pierwsze doniesienia o wykorzystaniu metody śledzenia promieni w grafice komputerowej zaprezentowano w [App68], [Mat68]. W algorytmie rekursywnego śledzenia promieni rozkład energii świetlnej jest obliczany z punktu widzenia obserwatora, tak więc zmiana położenia kamery wymaga ponownego przeprowadzenia obliczeń.

Śledzenie promieni

Algorytm śledzenia promieni powstał jako metoda tworzenia dwuwymiarowych obrazów trójwymiarowego świata. Ideę algorytmu śledzenia promieni przedstawiono na rys. 2.1. Obraz wynikowy jest



Rysunek 2.1. Idea algorytmu śledzenia promieni.

tworzony na dwuwymiarowej rzutni. Przed właściwym algorytmem należy wybrać położenie obserwatora i jego pole widzenia określające rzutnię. Na rzutni generowana jest siatka punktów określająca rozdzielczość docelowego obrazu.

Dla każdego punktu na rzutni tworzony jest co najmniej jeden promień rozpoczynający się w punkcie położenia obserwatora. Promień jest reprezentowany przez półprostą o początku **O** i wektorze kierunkowym **K**. Dla każdego promienia wyszukiwany jest zbiór obiektów przecinanych przez daną półprostą i obliczane są współrzędne punktów przecięcia. Spośród wyznaczonych punktów wybierany jest punkt **J** najbliższy obserwatora. Proces poszukiwania najbliższego punktu przecięcia dla zadanego promienia nazywany jest *rzucaniem promieni* (*ang. ray casting*).

Po wyznaczeniu punktu przecięcia z najbliższym obiektem generowany jest zbiór promieni z punktu **J** do źródeł światła. Służą one do sprawdzenia, czy pomiędzy punktem widzianym przez obserwatora a źródłami światła znajdują się jakieś przeszkody. Po wyznaczeniu wszystkich źródeł światła oświetlających punkt **J**, obliczane jest natężenie oświetlenia I dla tego punktu. Natężenie

oświetlenia punktu **J** określa jasność odpowiedniego punktu na rzutni (obrazie wynikowym). Z reguły natężenie oświetlenia jest wyznaczane według jednego z klasycznych, uproszczonych modeli oświetlenia. Jednym z najprostszych jest model dla odbicia rozproszonego

$$I = \rho \cdot \sum_i (\mathbf{N} \cdot \mathbf{L_i}) \cdot I_i, \tag{2.2}$$

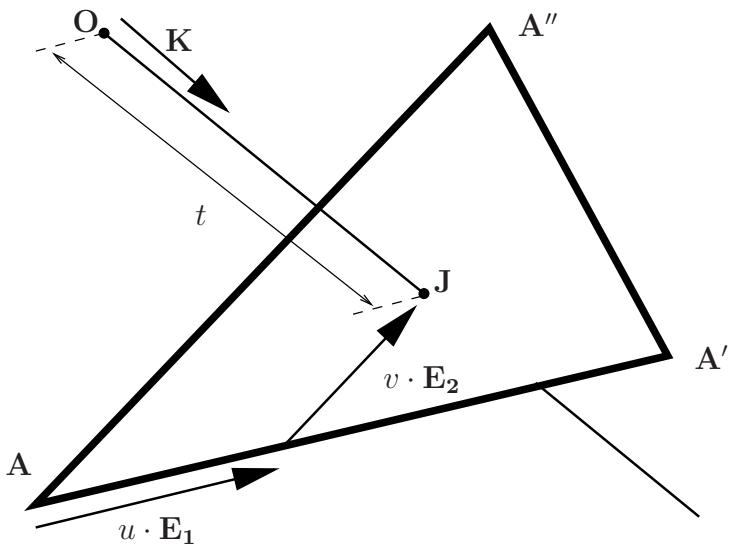
gdzie ρ oznacza współczynnik odbicia materiału, $\mathbf{L_i}$ jest wektorem kierunkowym promienia wyznaczania cienia, $\mathbf{N}$ oznacza wektor normalny powierzchni w punkcie **J**, a I_i jest natężeniem światła (światłością) i–tego, punktowego źródła światła. Przegląd różnych modeli oświetlenia można znaleźć np. w [FvDF$^+$01].

W algorytmie śledzenia promieni podstawową operacją jest wyznaczanie współrzędnych punktów przecięcia półprostych z obiektami na scenie. Czas potrzebny na obliczenia zależy od liczby półprostych, złożoności samej procedury obliczania współrzędnych punktu przecięcia półprostej z konkretnym obiektem, oraz liczby obiektów testowanych dla każdej półprostej. Liczba półprostych jest rezultatem przyjętej rozdzielczości obrazu wynikowego i liczby źródeł światła. Ponieważ rozdzielczość i model świata są ustalone przed wykonaniem algorytmu, należy szukać innych sposobów zwiększania wydajności algorytmu śledzenia promieni.

Wyznaczanie współrzędnych punktu przecięcia

Wyznaczenie współrzędnych punktu przecięcia półprostej z obiektem następuje poprzez rozwiązanie układu równań złożonego z równania prostej i równania opisującego powierzchnię obiektu. Pomijając najprostsze kształty, np. sfery, analityczne formuły opisujące powierzchnie modelowanych obiektów (np. twarz ludzką lub drzewo) są niezwykle złożone. Z tego powodu w grafice komputerowej skomplikowane powierzchnie są bardzo często modelowane za pomocą siatek wielokątów, najczęściej trójkątów. Dotyczy to także algorytmów oświetlenia globalnego, dlatego też podstawową operacją w algorytmie śledzenia promieni jest wyznaczanie współrzędnych punktu przecięcia półprostej z trójkątem.

Opracowano wiele algorytmów wyznaczania punktu wspólnego prostej i trójkąta [BA88], [Bad90], [Eri97], [MT97], [SF01]. Jednym z najwydajniejszych jest algorytm opisany w [MT97], który polega na zapisaniu położenia punktu przecięcia w układzie współrzędnych określonym przez dwa boki trójkąta i następnie sprawdzeniu, czy punkt przecięcia zawiera się wewnątrz badanego trójkąta. W



Rysunek 2.2. Wyznaczanie punktu przecięcia według algorytmu z [MT97].

algorytmie tym (rys. 2.2) półprosta jest opisana równaniem parametrycznym

$$R(t) = \mathbf{O} + t \cdot \mathbf{K}, \tag{2.3}$$

gdzie $\mathbf{O}$ jest punktem początkowym a $\mathbf{K}$ jest wektorem kierunkowym prostej. Badany trójkąt jest dany przez trzy wierzchołki $\mathbf{A}, \mathbf{A}', \mathbf{A}''$. Punkt $\mathbf{J}$ przecięcia półprostej i trójkąta jest opisany przez współrzędne barycentryczne $u, v : u \geq 0, v \geq 0, u + v \leq 1$ jako kombinacja liniowa wektorów określonych przez wierzchołki $\mathbf{A}, \mathbf{A}', \mathbf{A}''$

$$\mathbf{J}(u, v) = (1 - u - v) \cdot \mathbf{A} + u \cdot \mathbf{A}' + v \cdot \mathbf{A}''. \tag{2.4}$$

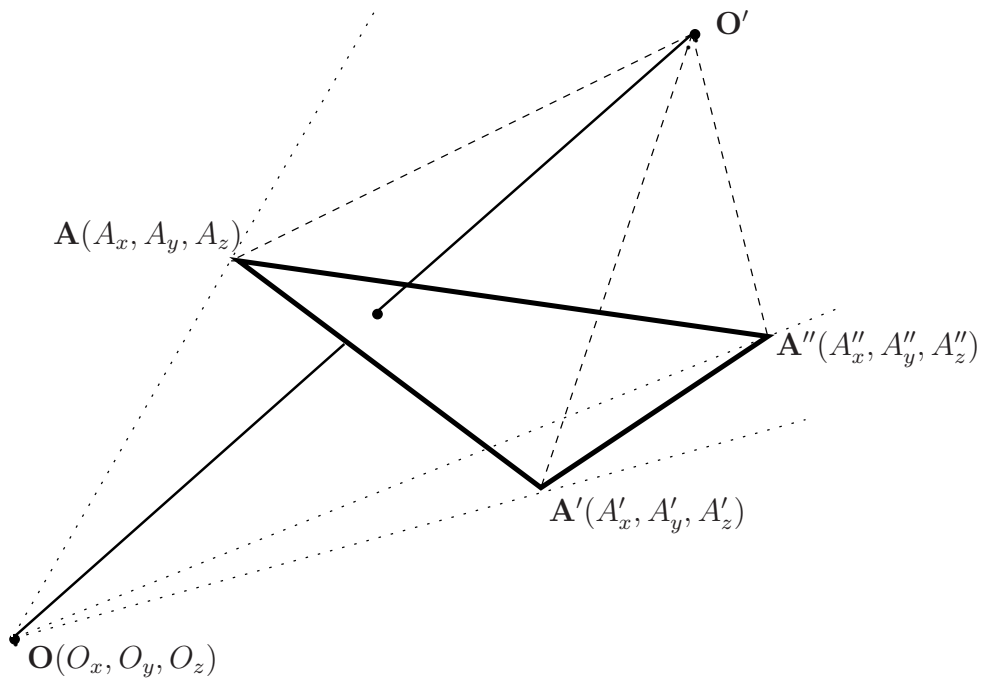
Punkt wspólny $\mathbf{J}$ półprostej (2.3) i trójkąta $\mathbf{A}\mathbf{A}'\mathbf{A}''$ jest rozwiązaniem równania wektorowego

$$\mathbf{O} + t \cdot \mathbf{K} = (1 - u - v) \cdot \mathbf{A} + u \cdot \mathbf{A}' + v \cdot \mathbf{A}'', \tag{2.5}$$

gdzie u, v określają jego położenie, a t odległość od początku półprostej $\mathbf{O}$. W opisywanym algorytmie równanie (2.5) zostało przekształcone do postaci

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\mathbf{S} \cdot \mathbf{E_1}} \begin{bmatrix} \mathbf{Q} \cdot \mathbf{E_2} \\ \mathbf{S} \cdot \mathbf{T} \\ \mathbf{Q} \cdot \mathbf{K} \end{bmatrix}, \tag{2.6}$$

gdzie $\mathbf{E_1} = \mathbf{A}' - \mathbf{A},\ \mathbf{E_2} = \mathbf{A}'' - \mathbf{A},\ \mathbf{T} = \mathbf{O} - \mathbf{A},\ \mathbf{S} = \mathbf{K} \times \mathbf{E_2},\ \mathbf{Q} = \mathbf{T} \times \mathbf{E_1}$. W pracy [MT97] zaprezentowano także sekwencję obliczeń pozwalająca znaleźć rozwiązanie układu równań (2.6).



Rysunek 2.3. Wyznaczanie punktu przecięcia według algorytmu z [SF01].

Sekwencja ta przedstawiona jest w alg. 1 na str. 30. Jej cechą charakterystyczną jest przesunięcie wszystkich dzieleń na koniec algorytmu.

Zaprezentowany algorytm pozwala na efektywne wyznaczenie współrzędnych punktu przecięcia prostej z trójkątem. Dodatkową jego zaletą jest możliwość określenia położenia punktu przecięcia w układzie współrzędnych określonym przez boki trójkąta, co jest niezwykle przydatne w późniejszych etapach tworzenia obrazu, np. przy teksturowaniu. Jeśli jednak nie ma potrzeby obliczenia współrzędnych punktu przecięcia, a jedyną interesującą informacją jest to, czy dana prosta przecina lub nie dany trójkąt, wydajniejszym rozwiązaniem jest zastosowanie algorytmu opisanego w [SF01].

W algorytmie zaprezentowanym w pracy [SF01] (rys. 2.3) półprosta zastąpiona jest odcinkiem $\mathbf{OO}'$. Podstawową operacją jest obliczanie objętości znakowanej czworościanów. Objętość znakowana czworościanu $\mathbf{OAA}'\mathbf{A}''$ jest zdefiniowana jako

$$[\mathbf{OAA}'\mathbf{A}''] = \frac{1}{6} \cdot \begin{vmatrix} A_x - O_x & A_y - O_y & A_z - O_z \\ A'_x - O_x & A'_y - O_y & A'_z - O_z \\ A''_x - O_x & A''_y - O_y & A''_z - O_z \end{vmatrix}, \tag{2.7}$$

gdzie (A_x, A_y, A_z) oznaczają współrzędne punktu $\mathbf{A}$.

W algorytmie tym wykorzystywany jest fakt, że znak objętości zależy od orientacji czworościanu,

Algorytm 1 Sekwencja działań w algorytmie wyznaczania punktu przecięcia prostej z trójkątem

Dane wejściowe: $\mathbf{O}, \mathbf{K}, \mathbf{A}, \mathbf{A}', \mathbf{A}''$

1: $\mathbf{E_1} = \mathbf{A}' - \mathbf{A}$

2: $\mathbf{E_2} = \mathbf{A}'' - \mathbf{A}$

3: $\mathbf{S} = \mathbf{K} \times \mathbf{E_2}$

4: $\delta = \mathbf{S} \cdot \mathbf{E_1}$

5: **if** $(\delta = 0)$ **then**

6: **return**

7: **end if**

8: $\mathbf{T} = \mathbf{O} - \mathbf{A}$

9: $u' = \mathbf{T} \cdot \mathbf{S}$

10: **if** $(u' < 0)$ **or** $(u' > \delta)$ **then**

11: **return**

12: **end if**

13: $\mathbf{Q} = \mathbf{T} \times \mathbf{E_1}$

14: $v' = \mathbf{K} \cdot \mathbf{Q}$

15: **if** $(v' < 0)$ **or** $(u' + v' > \delta)$ **then**

16: **return**

17: **end if**

18: $t = \mathbf{E_2} \cdot \mathbf{Q}$

19: $t = \frac{t'}{\delta}$

20: $u = \frac{u'}{\delta}$

21: $v = \frac{v'}{\delta}$

22: **return** t, u, v

tj. od kolejności trzech wierzchołków widzianych z czwartego. Czworościan $\mathbf{OAA'A''}$ jest dodatnio zorientowany, jeśli wierzchołki $\mathbf{AA'A''}$ obserwowane z wierzchołka $\mathbf{O}$ są widziane zgodnie z kierunkiem ruchu wskazówek zegara. Można więc określić, czy odcinek $\mathbf{OO'}$ przecina trójkąt $\mathbf{AA'A''}$ badając znaki objętości znakowanych odpowiednich czworościanów. W [SF01] udowodniono, że odcinek $\mathbf{OO'}$ przecina trójkąt $\mathbf{AA'A''}$ wtedy i tylko wtedy, gdy objętości czworościanów $\mathbf{O'AA'O}$, $\mathbf{O'A''A'O}$ i $\mathbf{O'AA''O}$ są nieujemne.

Wybór obiektów testowanych dla pojedynczego promienia

Wydajność algorytmu śledzenia promieni można zwiększyć nie tylko przez stosowanie szybkich procedur wyznaczających punkt przecięcia promienia z obiektem, ale także przez zmniejszenie liczby obiektów testowanych dla pojedynczego promienia. W algorytmie śledzenia promieni należy znaleźć najbliższy obserwatora obiekt przecinany przez dany promień. Najbardziej naiwną metodą jest wyznaczenie punktów przecięcia dla wszystkich obiektów na scenie, a następnie wybranie spośród nich jednego. Zdecydowana większość testów okaże się jednak zupełnie zbędna, ponieważ z reguły pojedyncza prosta przecina jedynie niewielki podzbiór wszystkich elementów sceny. Korzystne jest zatem zastosowanie metody wstępnego typowania potencjalnych kandydatów przekazywanych do procedury wyznaczania punktu przecięć.

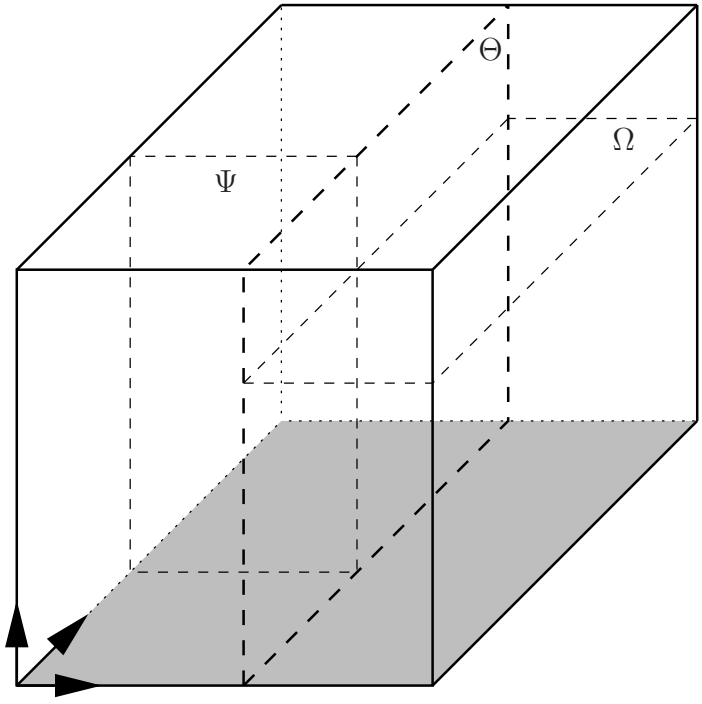
Często stosowanymi metodami eliminacji obiektów przecinających daną prostą są algorytmy wykorzystujące podział przestrzeni. W tych algorytmach przestrzeń sceny jest dzielona na elementy (komórki) zawierające w sobie pewną liczbę sąsiednich obiektów. Punkty przecięcia z prostą dla obiektów wchodzących w skład danej komórki są wyznaczane dopiero wtedy, gdy komórka ta jest przecinana przez badaną prostą. Rozmiar komórek może być adaptowany do lokalnej gęstości obiektów na scenie, tj. fragmenty przestrzeni z niewielką liczbą obiektów dzielone są na komórki o dużej objętości, a fragmenty z dużą liczbą obiektów na komórki o niewielkiej objętości. Rozmiar poszczególnych komórek może wynikać np. z ograniczeń na minimalną i maksymalną liczbę obiektów zawartych w pojedynczym elemencie.

Z poszczególnych elementów przestrzeni buduje się często struktury hierarchiczne, w których elementy o mniejszej objętości zawierają się w elementach o większej objętości. Powstaje w ten sposób struktura drzewiasta, w której kolejne poziomy zawierają elementy o coraz mniejszej objętości. Przykładem takiej struktury może być drzewo binarnego podziału przestrzeni (drzewo BSP, *ang. Binary Space Partitioning tree*), w którym poszczególne węzły są dzielone na dwa elementy. Istotną cechą struktur drzewiastych jest logarytmiczna zależność głębokości od liczby liści drzewa. Pozwala to na efektywne implementacje algorytmów wyszukiwania w drzewie obiektów przecinanych przez prostą.

Algorytm konstrukcji drzewa BSP jest algorytmem rekurencyjnym. W wyniku jego wykonania powstaje drzewo, którego korzeniem jest cała przestrzeń sceny. W pierwszym kroku algorytmu wybierana jest płaszczyzna dzieląca całą dostępną przestrzeń sceny na dwa elementy. Elementy te stają się potomkami korzenia drzewa. Z każdym elementem wiązany jest zbiór obiektów zawierających się po odpowiedniej stronie płaszczyzny dzielącej. Następnie procedura ta jest powtarzana rekuren-

cyjnie dla każdego powstałego elementu. Warunkiem zakończenia rekursji jest osiągnięcie zadanej głębokości drzewa bądź granicznej liczby obiektów w każdym elemencie.

Szczególnym przypadkiem drzew BSP są drzewa kd, w których płaszczyzny dzielące są zawsze prostopadłe do osi układu współrzędnych. Dzięki temu badanie warunków w algorytmach budowania i przeglądania takich drzew może być przeprowadzane bez konieczności wykonywania kosztownych obliczeń. Z tego względu drzewa kd są chętnie stosowane w sprzętowych implementacjach algorytmów oświetlenia globalnego. Przykład drzewa kd przedstawiono na rys. 2.4. Przestrzeń sceny jest podzielona płaszczyzną Θ, a lewy i prawy potomek odpowiednio płaszczyznami Ψ i Ω.



Rysunek 2.4. Drzewo kd.

W procesie renderingu scen trójwymiarowych problem budowania drzew BSP może zostać pominięty, ponieważ w dużej części przypadków drzewo może zostać skonstruowane na etapie modelowania sceny. Znacznie istotniejszym problemem jest wytypowanie cel przecinanych przez dany promień. Opracowano wiele algorytmów przeglądania drzew BSP, zaczynając od [Arv88]. Bardziej kompletny przegląd wielu możliwych rozwiązań zaprezentowano w [Bit99]. Jednym z najszybszych jest algorytm przeglądania drzew kd z [HKBZ97]. Idea algorytmu polega na sprawdzeniu wzajemnego położenia punktów przecięcia promienia z płaszczyznami ograniczającymi daną celę oraz z

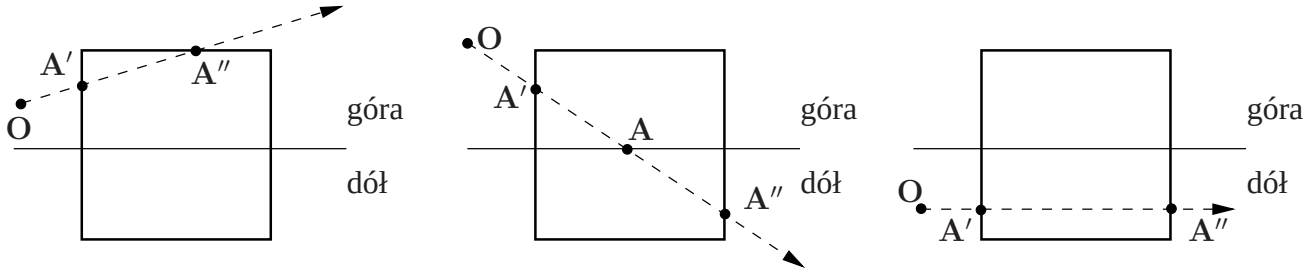
płaszczyzną dzielącą. Ponieważ płaszczyzny są prostopadłe do osi układu współrzędnych, problem sprowadza się do porównania pojedynczych współrzędnych. Dodatkowo, kolejność testów jest tak dobrana, aby przypadki występujące najczęściej były obsługiwane z najmniejszymi opóźnieniami.

Zapis działań metody opisanej w [HKBZ97] przedstawiono w alg. 2. Przez A'_* i A''_* oznaczono współrzędne punktów przecięcia promienia z płaszczyznami ograniczającymi badany węzeł. Indeks współrzędnych zależy od orientacji płaszczyzny dzielącej komórkę – odpowiada on osi prostopadłej do płaszczyzny dzielącej. Jako $\mathbf{A}'$ oznaczono punkt znajdujący się bliżej początku promienia (punkt wejściowy do komórki), jako $\mathbf{A}''$ oznaczono punkt dalszy (wyjściowy). Lewy węzeł zawiera obiekty *poniżej* płaszczyzny dzielącej, prawy węzeł obejmuje obszar *powyżej* płaszczyzny dzielącej. Przeglądanie drzewa odbywa się od korzenia w kierunku liści. Korzeń drzewa zawiera całą przestrzeń sceny. Na rys. 2.5 przedstawiono schematycznie kilka z możliwych położeń promienia i węzłów drzewa.

Algorytm 2 Przeglądanie drzewa kd według algorytmu z pracy [HKBZ97].

Dane wejściowe: Współrzędne punktów przecięcia $\mathbf{A}', \mathbf{A}''$ prostej z płaszczyznami ograniczającymi komórkę i płaszczyzną dzielącą (punkt $\mathbf{A}$).

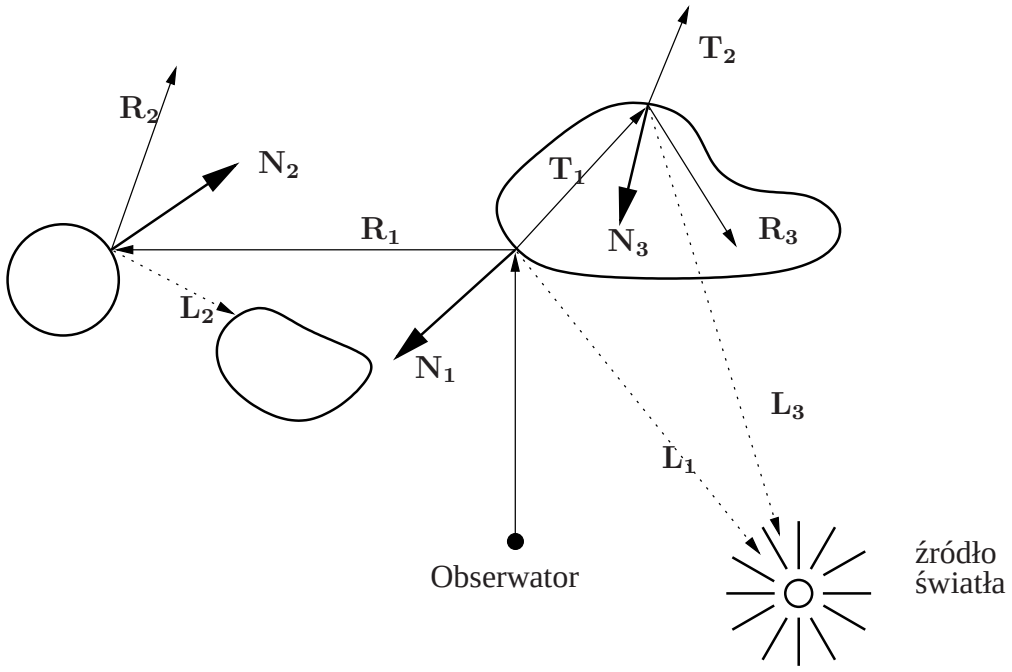
1: **if** $(A'_* \leq A_*)$ **then**
2: **if** $(A''_* < A_*)$ **then**
3: Sprawdź lewy węzeł
4: **else**
5: **if** $(A''_* = A_*)$ **then**
6: Sprawdź dowolny węzeł
7: **else**
8: Sprawdź lewy i prawy węzeł
9: **end if**
10: **end if**
11: **else**
12: **if** $(A''_* > A_*)$ **then**
13: Sprawdź prawy węzeł
14: **else**
15: Sprawdź prawy i lewy węzeł
16: **end if**
17: **end if**



Rysunek 2.5. Wybrane położenia promienia przy przeglądaniu drzew kd.

Rekursywne śledzenie promieni

Klasyczny algorytm śledzenia promieni modeluje jedynie przepływ energii od źródeł światła do oświetlanych obiektów pomijając całkowicie wzajemną wymianę energii pomiędzy pozostałymi elementami sceny. Poprawne modelowanie rozkładu energii według równania (2.1) wymaga uwzględnienia wpływu oświetlenia światłem odbitym od elementów nie będących źródłami światła. Konieczne rozszerzenia podstawowego algorytmu zaproponowano po raz pierwszy w pracy [Whi80].



$\mathbf{N_i}$ - wektor normalny powierzchni
$\mathbf{R_i}$ - promień odbity
$\mathbf{L_i}$ - promień wyznaczania cienia
$\mathbf{T_i}$ - promień transmitowany

Rysunek 2.6. Idea algorytmu rekursywnego śledzenia promieni.

W algorytmie rekursywnego śledzenia promieni (rys. 2.6) zbiór promieni generowanych z punktu **J** jest poszerzony o promień odbity i promień transmitowany (dla obiektów przezroczystych). Dla każdego z tych promieni wyznaczany jest najbliższy przecinany obiekt, po czym z punktów dla znalezionych przecięć generowane są kolejne zbiory promieni. Kolejne zbiory promieni tworzą *drzewo promieni*. Procedura budowania drzewa promieni jest kontynuowana do momentu, gdy zostanie spełniony jeden z dwóch warunków:

- dla kolejnego promienia nie można znaleźć przecinanego obiektu,
- osiągnięto założoną głębokość drzewa.

Algorytm rekursywnego śledzenia promieni doskonale modeluje wszelkie zjawiska związane z odbiciami zwierciadlanymi. Niestety, nie bierze on pod uwagę powierzchni matowych, tj. odbijających światło we wszystkich kierunkach równomiernie. Powoduje to konieczność stosowania korekcji związanych z *oświetleniem otoczenia*, czyli światłem odbitym od pozostałych obiektów. Poprawną symulację wymiany światła pomiędzy powierzchniami matowymi zapewniają metody energetyczne, niestety ich wadą jest wysoki koszt obliczeń dla modelowania powierzchni zwierciadlanych.

2.1.2 Metody energetyczne

Metody energetyczne (*ang. radiosity*) wykorzystują inżynierskie modele wymiany ciepła stosowane m.in. przy obliczaniu elementów statków kosmicznych. Pierwsze doniesienia o wykorzystaniu tych algorytmów w grafice komputerowej zaprezentowano w pracach [GTGB84] i [NN85]. W metodzie energetycznej zakłada się, że całkowita energia świetlna emitowana lub odbijana od dowolnej powierzchni jest absorbowana lub odbijana przez inne powierzchnie. Rozkład energii świetlnej na scenie jest obliczany niezależnie od pozycji obserwatora, a więc po jego wyznaczeniu konieczna jest odpowiednia projekcja wyników.

Wprowadzenie

Podstawowym założeniem w metodzie energetycznej jest potraktowanie wszystkich powierzchni jako nieprzezroczystych i matowych. Powierzchnie takie odbijają światło jednakowo w każdym kierunku, a więc niezależnie od kąta patrzenia wydają się jednakowo jasne. Zostały one nazwane *powierzchniami lambertowskimi* od nazwiska Johanna Heinricha Lamberta, który prowadził nad nimi badania

w XVII wieku. Natężenie światła odbitego od powierzchni lambertowskiej jest określone *prawem Lamberta*

$$I_o = I_s \cdot \rho \cdot \cos(\alpha), \tag{2.8}$$

gdzie I_o oznacza natężenie światła odbitego zarejestrowane przez obserwatora, I_s jest natężeniem źródła światła, $\rho \in [0, 1]$ jest współczynnikiem odbicia, a α kątem pomiędzy kierunkiem obserwacji a wektorem normalnym do powierzchni. Dzięki temu, że stosunek natężenia światła odbitego do padającego zależy wyłącznie od kąta obserwacji i współczynnika odbicia, możliwe jest znaczne uproszczenie równań opisujących przepływ energii świetlnej pomiędzy elementami sceny.

Zadaniem metod energetycznych jest wyznaczenie dla każdego punktu na scenie jego *promienistości* (*ang. radiosity*) $B(\mathbf{A})$, czyli strumienia energii emitowanego z danej powierzchni. Jednostką promienistości, nazywanej także *natężeniem wypromieniowania*, jest $\left[\frac{W}{m^2}\right]$. Przy ograniczeniu wszystkich powierzchni na scenie do powierzchni lambertowskich, równanie (2.1) można uprościć do postaci [CW93]

$$B(\mathbf{A}) = E(\mathbf{A}) + \rho(\mathbf{A}) \int_S B(\mathbf{A}')G(\mathbf{A}, \mathbf{A}')d\mathbf{A}', \tag{2.9}$$

gdzie $B(\mathbf{A})$ i $B(\mathbf{A}')$ oznaczają odpowiednio promienistości w punktach $\mathbf{A}$ i $\mathbf{A}'$, $E(\mathbf{A})$ oznacza promienistość własną źródeł światła, $\rho(\mathbf{A})$ jest współczynnikiem odbicia, a $G(\mathbf{A}, \mathbf{A}')$ jest nazywane jądrem geometrycznym. Wartość $G(\mathbf{A}, \mathbf{A}')$ zależy zarówno od wzajemnego położenia punktów $\mathbf{A}$ i $\mathbf{A}'$, jak i od obecności przeszkód pomiędzy tymi punktami.

Dyskretyzacja równania oświetlenia

Pomimo uproszczeń, analityczne rozwiązanie równania (2.9) jest możliwe jedynie w bardzo prostych przypadkach. Stosowane są zatem metody probabilistyczne i numeryczne pozwalające znaleźć rozwiązanie przybliżone. Jedną z najczęściej stosowanych metod numerycznych jest metoda elementów skończonych (MES). Pozwala ona przybliżyć rozwiązanie równania (2.9) rozwiązaniem układu równań liniowych.

W metodzie elementów skończonych skomplikowana funkcja jest aproksymowana sumą ważoną prostych *funkcji bazowych*. Każda z funkcji bazowych określona jest na pewnym przedziale wybranym z dziedziny funkcji aproksymowanej. Suma przedziałów funkcji bazowych musi pokryć całą dziedzinę funkcji aproksymowanej. Współczynniki przy funkcjach bazowych są dobierane w ten sposób, aby wartości funkcji aproksymowanej i sumy ważonej funkcji bazowych były sobie rów-

ne w wybranych punktach dziedziny, nazywanych *węzłami*. Wartości współczynników w węzłach są niewiadomymi, które można znaleźć rozwiązując liniowy układ równań.

Idea zastosowania MES w metodzie energetycznej polega na aproksymacji ciągłej funkcji $B(\mathbf{A})$ za pomocą sumy

$$B(\mathbf{A}) \approx \sum_i B_i \xi_i(\mathbf{A}), \tag{2.10}$$

gdzie $\xi_i(\mathbf{A})$ oznacza i–tą funkcję bazową. Funkcjami bazowymi są z reguły funkcje wielomianowe przyjmujące wartości niezerowe jedynie w bliskim otoczeniu danego węzła, zaś węzłami są wybrane punkty na powierzchniach występujących na scenie. Każda funkcja bazowa jest określona dla pewnego przedziału dziedziny $B(\mathbf{A})$, czyli dla pewnego wycinka powierzchni. Poza wybranym przedziałem wartość $\xi_i(\mathbf{A})$ jest równa zero. Często używaną funkcją bazową jest

$$\xi_i(\mathbf{A}) = \begin{cases} 1 & \text{jeśli } \mathbf{A} \text{ należy do płatu } i \\ 0 & \text{jeśli } \mathbf{A} \text{ nie należy do płatu } i \end{cases}. \tag{2.11}$$

Metoda elementów skończonych pozwala zapisać formułę (2.9) jako układ równań liniowych

$$\mathbf{E} = (\mathbf{\Xi} - \mathbf{P} \cdot \mathbf{F}) \cdot \mathbf{B}, \tag{2.12}$$

gdzie $\mathbf{E}$ oznacza wektor promienistości własnych, elementy macierzy $\mathbf{\Xi}$ są splotem odpowiednich funkcji bazowych, macierz $\mathbf{P}$ jest macierzą diagonalną współczynników odbicia, macierz $\mathbf{F}$ jest macierzą współczynników konfiguracji (kształtu) zależnych od współczynników sprzężenia, a wektor $\mathbf{B}$ jest poszukiwanym wektorem promienistości. Dla stałych funkcji bazowych macierz $\mathbf{\Xi}$ jest macierzą jednostkową, a macierz $\mathbf{F}$ zawiera współczynniki sprzężenia. Wektor rozwiązań układu (2.12) wyznacza z założonym błędem wartości promienistości dla wybranego zbioru punktów stanowiących węzły.

Znalezienie układu równań liniowych aproksymującego (2.9) wymaga przeprowadzenia szeregu operacji. Pierwszą z nich jest podział wszystkich powierzchni występujących na scenie na niewielkie *płaty elementarne*. Dla każdego płatu należy wybrać zbiór punktów będących węzłami, w których obliczane są wartości promienistości. Z każdym węzłem należy związać wybraną funkcję bazową. Liczba i położenie węzłów zależy od postaci funkcji bazowych. Dla stałych funkcji bazowych wybierany jest zazwyczaj jeden punkt pośrodku płata elementarnego. Dla każdej pary płatów elementarnych należy także wyznaczyć wartość współczynnika sprzężenia, czyli wielkości opisującej względne położenie i obecność przeszkód ograniczających wzajemną widoczność. Znajomość powyższych danych jednoznacznie określa współczynniki układu równań (2.12).

Współczynniki konfiguracji

Współczynniki konfiguracji (kształtu) określają ważony wpływ energii emitowanej z dziedziny pewnej funkcji bazowej na dziedzinę innej funkcji bazowej. W najprostszym przypadku są równe stosunkowi energii docierającej z płatu i do płatu j do całkowitej energii emitowanej z płatu i. Współczynniki kształtu zależą wyłącznie od odległości i wzajemnej orientacji płatów elementarnych oraz od istnienia ewentualnych przeszkód pomiędzy nimi.

Wyznaczenie współczynnika sprzężenia dla pary płatów elementarnych ma dwa etapy. W pierwszym wyznacza się *funkcję widoczności*. Wartość funkcji widoczności zależy od obecności przeszkód częściowo lub całkowicie blokujących wymianę energii pomiędzy badanymi płatami. Z reguły stosuje się dwuwartościową funkcję widoczności równą 1, gdy pomiędzy płatami elementarnymi nie ma żadnych przeszkód, i równą 0 w przeciwnym przypadku.

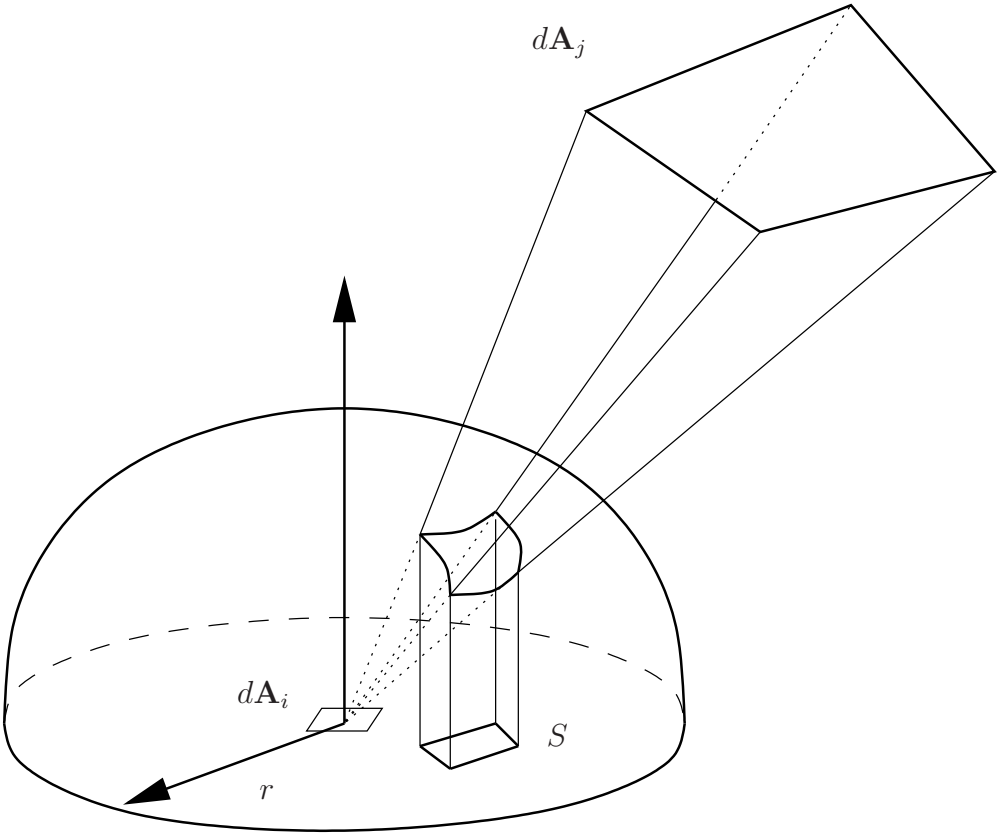
Do wyznaczania funkcji widoczności można posłużyć się algorytmem rzucania promieni. Konstruuje się promień o początku na jednym z płatów elementarnych i skierowany na drugi z płatów. Następnie poszukuje się obiektów przecinanych przez ten promień, które są położone pomiędzy początkiem promienia a drugim z badanych płatów. Wystarczy znaleźć jeden taki obiekt, aby określić wartość funkcji widoczności równą zero. Z powodu logarytmicznej złożoności algorytmu poszukiwania obiektów przecinanych przez promień rozwiązanie to jest dość efektywne.

Drugim krokiem procedury wyznaczania współczynnika sprzężenia jest określenie jego wartości zależnej od kształtu i wzajemnego położenia płatów elementarnych. Dla niektórych kształtów płatów elementarnych znane są analityczne formuły określające tę wartość, jednak z powodu ich złożoności są rzadko stosowane. Jednym z podstawowych sposobów obliczania wartości współczynnika kształtu jest metoda próbkowania półsfery (metoda Nusselta) [CW93].

Ideę metody Nusselta przedstawiono na rys. 2.7. Konstruowana jest umowna półsfera o środku położonym na jednym z płatów elementarnych. Drugi z płatów jest rzutowany na powierzchnię półsfery, a następnie na koło będące jej podstawą. Współczynnik kształtu jest równy stosunkowi pola zajętego przez rzut do pola całego koła

$$F_{d\mathbf{A}_i \to d\mathbf{A}_j} = \frac{S}{\pi r^2}. \tag{2.13}$$

Metoda Nusselta stanowi podstawę wielu algorytmów. Jedną z modyfikacji jest metoda próbkowania półsześcianu (rys. 2.8). Półsfera zastępowana jest półsześcianem, którego ściany są dzielone na niewielkie elementy. Z każdym elementem związana jest wartość jego współczynnika kształtu

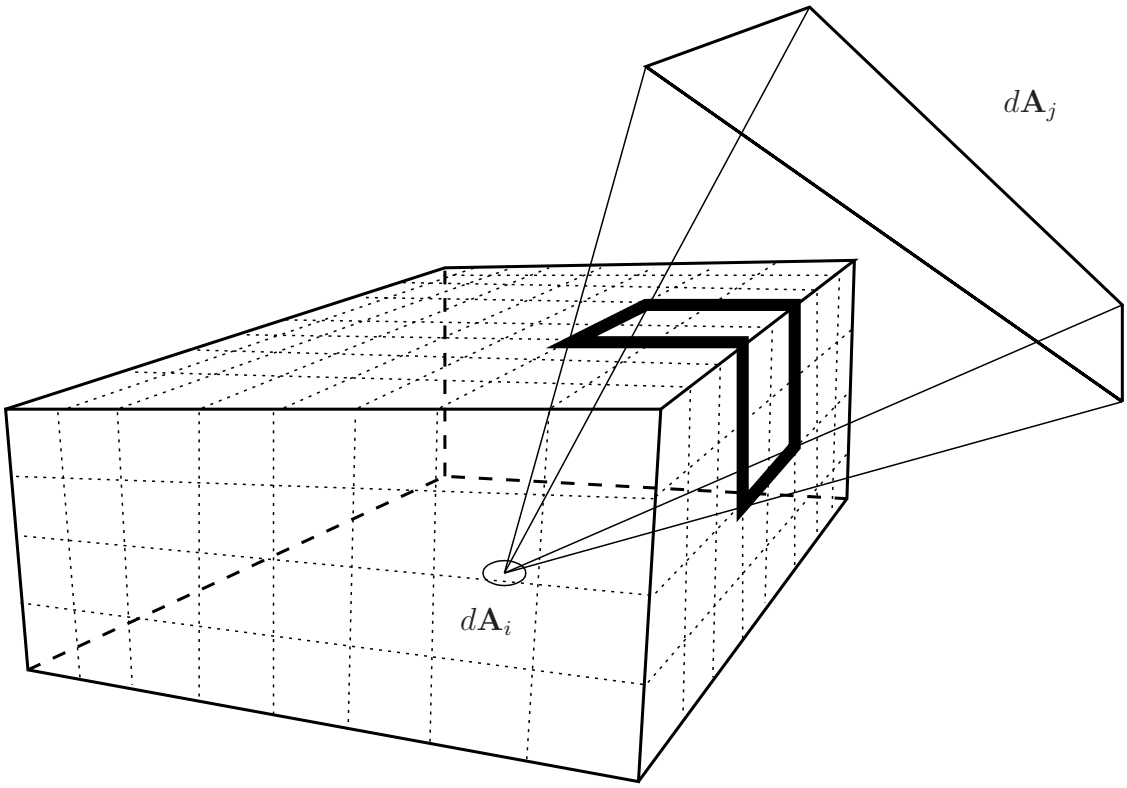


Rysunek 2.7. Wyznaczanie współczynników kształtu metodą Nusselta.

względem płatu $d\mathbf{A}_i$. Następnie badany płat $d\mathbf{A}_j$ jest rzutowany na powierzchnię półsześcianu i jego współczynnik kształtu jest obliczany jako suma współczynników "przykrytych" elementów.

Metoda próbkowania półsześcianu jest szczególnie podatna na implementację z wykorzystaniem klasycznych procesorów graficznych zawierających realizację sprzętową algorytmu wyznaczania obiektów widocznych, np. algorytm z–bufora lub inaczej bufora głębokości. Algorytm ten pozwala zrzutować wszystkie elementy sceny na płaszczyzny wyznaczone przez ściany półsześcianu. Przy rzutowaniu zachowana jest kolejność obiektów, tzn. płaty położone dalej od ścian półsześcianu zostaną przysłonięte przez elementy bliższe. Pozwala to na jednoczesne wyznaczenie funkcji widoczności. Czas wykonania algorytmu wykorzystującego bufor głębokości zależy liniowo od liczby elementów na scenie. Szereg implementacji wykorzystujących ideę próbkowania półsześcianu i klasyczne procesory graficzne zaprezentowano m.in. w pracach [Kel97], [BGZ97], [NC02], [Lan04], [CHL04].

Istnieją także inne algorytmy wyznaczania współczynników sprzężenia. Przykładem mogą być różnorakie metody probabilistyczne. Są one jednak rzadko stosowane ze względu na czasochłonność i niską jakość uzyskiwanych wyników. Wspomaganie metod energetycznych przy użyciu algorytmów



Rysunek 2.8. Wyznaczanie współczynników kształtu metodą próbkowania półsześcianu.

probabilistycznych zastosowano w procesorze AR350 [Hal01].

Rozwiązanie układu równań oświetlenia

Rozwiązanie układu równań (2.12) pozwala na znalezienie wartości promienistości dla zbioru punktów położonych na powierzchni obiektów występujących na scenie. Niestety, aby uzyskać wystarczającą dokładność, konieczne jest stosowanie odpowiednio gęstej siatki węzłów, czyli odpowiednio małych płatów elementarnych. Dla skomplikowanych scen liczba płatów elementarnych może przekroczyć 10^6 elementów. Duża liczba węzłów, a więc także zmiennych w układzie (2.12), wymaga ogromnych mocy obliczeniowych do znalezienia rozwiązania.

Rozmiar układu równań oświetlenia powoduje, że klasyczne metody (np. metoda Gaussa o złożoności $O(n^3)$) są praktycznie bezużyteczne. W praktycznych implementacjach stosowane są zazwyczaj relaksacyjne metody rozwiązywania układu równań, będące modyfikacjami metod iteracyjnych Jacobiego czy Gaussa–Seidla. Często stosowaną jest metoda *progresywnego ulepszania* (*ang. progressive refinement*) [Lan04].

W algorytmie progresywnego ulepszania (alg .3) z każdym płatem elementarnym i związana jest

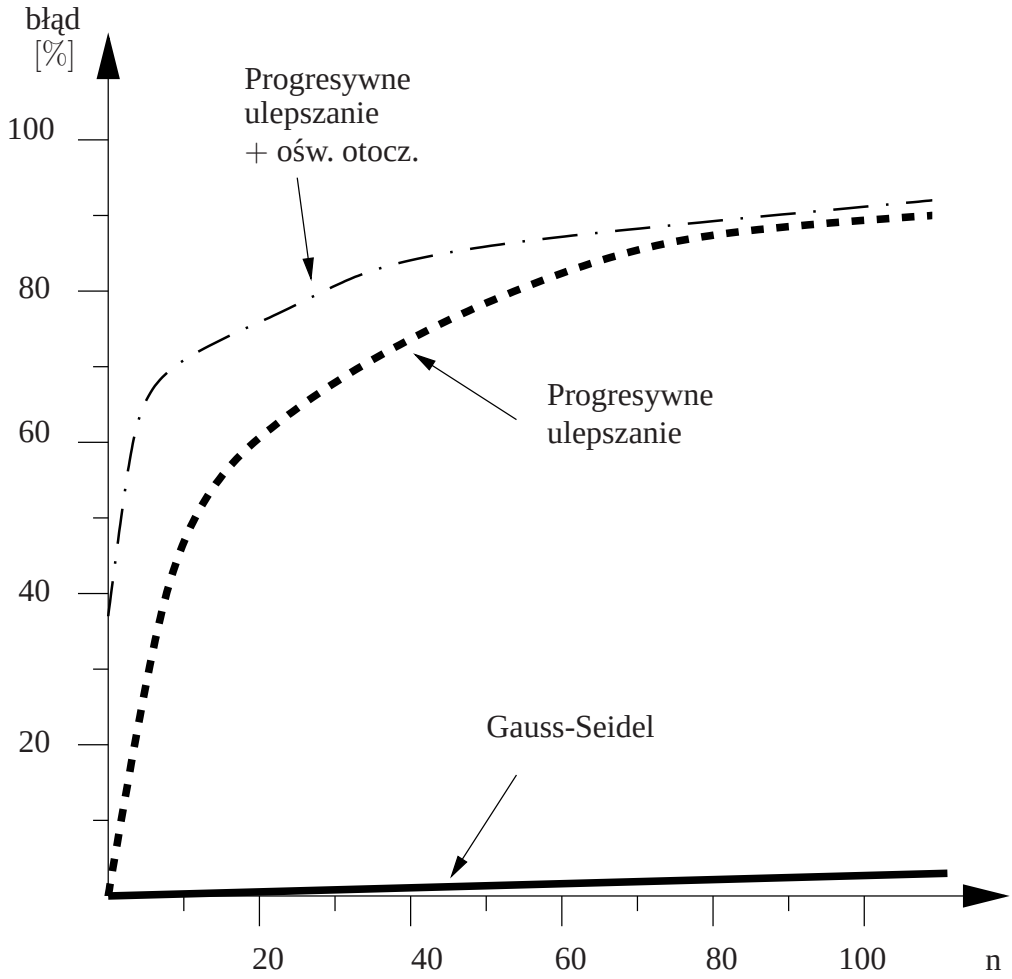
wartość dotychczas obliczonej promienistości B_i oraz promienistości, która jeszcze nie została wypromieniowana ΔB_i. Algorytm ten jest algorytmem iteracyjnym, w którym pojedyncza iteracja składa się z dwóch etapów. Etap pierwszy obejmuje wybór elementu z największą wartością niewypromieniowanej energii ΔB_i. W drugim kroku energia wszystkich płatów elementarnych jest zwiększana o wartość związaną z emisją energii ΔB_i.

Algorytm 3 Rozwiązanie układu równań oświetlenia metodą progresywnego ulepszania.

Dane wejściowe: Zbiór wszystkich płatów elementarnych, ich współczynników odbicia ρ_i i promienistości własnych E_i

1: **for** $i = 1$ **to** n **do**
2: $B_i = E_i$
3: $\Delta B_i = E_i$
4: **end for**
5: **while** przybliżenie mało dokładne **do**
6: Wybierz element i z największą niewypromieniowaną energią $\Delta B_i \cdot A_i$
7: **for** $j = 1$ **to** n **do**
8: $\Delta rad = \Delta B_i \cdot \rho_j \cdot F_{ji}$
9: $\Delta B_j = \Delta B_j + \Delta rad$
10: $B_j = B_j + \Delta rad$
11: **end for**
12: $\Delta B_i = 0$
13: **end while**

Podstawową zaletą metody progresywnego ulepszania jest szybka zbieżność kolejnych rozwiązań, szczególnie dla początkowych kroków. W wielu przypadkach wystarczającą dokładność osiąga się po niewielkiej liczbie iteracji. Ponieważ każda iteracja wymaga uaktualnienia wszystkich płatów elementarnych, złożoność całego algorytmu progresywnego ulepszania może być oszacowana jako $O(n)$, gdzie n oznacza liczbę płatów elementarnych. Dodatkowo dokładność wyników dla początkowych kroków można zwiększyć wprowadzając dodatkową korekcję związaną z oświetleniem otoczenia. Na rys. 2.9 przedstawiono porównanie błędu obliczonych wartości promienistości w funkcji liczby iteracji.



Rysunek 2.9. Porównanie błędu obliczonych wartości promienistości w funkcji liczby iteracji dla różnych metod rozwiązywania układu równań oświetlenia [CW93].

2.2 Resztowe systemy liczbowe

2.2.1 Wybrane zagadnienia teorii liczb

Reszta z dzielenia liczby całkowitej X przez liczbę naturalną $M > 1$ jest oznaczana jako $|X|_M$ lub $X \bmod M$. Wartość reszty $W = |X|_M$ jest zdefiniowana jako

$$W = |X|_M \implies X = \left\lfloor \frac{X}{M} \right\rfloor \cdot M + W. \tag{2.14}$$

Z równania (2.14) wynika, że wartość reszty jest nieujemna. Możliwe jest zdefiniowanie także reszt ujemnych, jednak w dalszej części pracy przypadek ten nie jest rozważany.

O dwóch liczbach całkowitych X, Y mówimy, że są *przystające* (kongruentne) modulo M, jeśli różnica $X - Y$ jest podzielna przez M, czyli $|X - Y|_M = 0$. Relację kongruencji oznacza się

symbolem $X \equiv Y \bmod M$. Bezpośrednio z definicji kongruencji wynika, że jeśli $X \equiv Y \bmod M$ i $Z \equiv W \bmod M$, to

$$(X \odot Z) \equiv (Y \odot W) \bmod M, \tag{2.15}$$

gdzie $\odot$ oznacza jedno z działań $\{+, -, \times\}$. Własność ta nazywana jest *niezmienniczością* relacji kongruencji względem dodawania, odejmowania i mnożenia.

Przy wykonywaniu operacji arytmetycznych modulo użyteczne jest posługiwanie się pojęciami *odwrotności addytywnej modulo* i *odwrotności multiplikatywnej modulo*. Odwrotnością addytywną liczby X modulo M jest taka liczba Y, że $|X + Y|_M = 0$. Odwrotność addytywną oznacza się jako $|-X|_M$. Odwrotnością multiplikatywną liczby X modulo M jest taka liczba Y, że $|X \cdot Y|_M = 1$. Warunkiem istnienia odwrotności multiplikatywnej jest wzajemna względna pierwszość X i M. Odwrotność multiplikatywna jest oznaczana jako $|X^{-1}|_M$.

Podstawą wielu algorytmów arytmetyki resztowej jest małe twierdzenie Fermata [GKP01, Kob95], które dla liczby pierwszej M i liczby całkowitej X względnie pierwszej z M stanowi

$$\left|X^{M-1}\right|_M = 1. \tag{2.16}$$

Poprawność (2.16) można łatwo uzasadnić. Ponieważ X jest względnie pierwsze z M, to reszty $|X|_M, |2 \cdot X|_M, \ldots, |(M-1) \cdot X|_M$ kolejnych krotności X są różne, czyli tworzą permutację ciągu $1, 2, 3, \ldots, M-1$. Iloczyn reszt tych krotności będzie więc równy iloczynowi kolejnych elementów ciągu $1, 2, 3, \ldots, M-1$, czyli $(M-1)!$. Wynika stąd, że

$$X^{M-1} \cdot (M-1)! \equiv (M-1)! \bmod M,$$

skąd bezpośrednio wynika wzór (2.16).

Na podstawie małego twierdzenia Fermata można wykazać ([Kob95], str. 52–53), że dla każdej liczby pierwszej M istnieje $\varphi(M-1)$ liczb naturalnych g, dla których reszty kolejnych $M-1$ potęg są różne, czyli generują permutację ciągu $1, 2, 3, \ldots, M-1$. Funkcja $\varphi(M-1)$ jest funkcją Eulera i jest określona jako liczba mniejszych od $M-1$ liczb naturalnych względnie pierwszych z $M-1$. Liczby g nazywane są *generatorami grupy multiplikatywnej*. Wynika stąd, że znając g dla danego M można jednoznacznie przekształcić liczby całkowite X z zakresu $[1, M-1]$ na odpowiadające im wykładniki $j_X \in [0, M-2]$, gdzie

$$X = \left|g^{j_X}\right|_M. \tag{2.17}$$

Wykładniki j_X nazywane są także *indeksami* i tworzą *grupę addytywną*, a całe przekształcenie określane jest *transformacją na izomorficzną grupę addytywną*. Transformacja opisana przez (2.17) pozwala na zastąpienie mnożenia modulo M dodawaniem modulo $M-1$ zgodnie z wzorem

$$|X \cdot Y|_M = \left|g^{|j_X + j_Y|_{M-1}}\right|_M, \tag{2.18}$$

gdzie $X = |g^{j_X}|_M$, $Y = |g^{j_Y}|_M$.

Uogólnieniem małego twierdzenia Fermata jest twierdzenie Eulera, które stanowi, że dla pary względnie pierwszych liczb naturalnych M, X

$$\left|X^{\varphi(M)}\right|_M = 1. \tag{2.19}$$

Konsekwencją (2.19) jest powtarzalność reszt kolejnych potęg X modulo M z okresem równym co najwyżej $\varphi(M)$. Przyjmując $X = 2$ można tę właściwość wykorzystać do efektywnego wykonywania operacji modulo M dla M nieparzystych na reprezentacjach binarnych, ponieważ reszty dla kolejnych cyfr o wagach 2^i będą się powtarzały okresowo [Pie94].

Najmniejsza odległość między identycznymi wartościami w ciągu reszt dla kolejnych potęg 2 modulo M nazywana jest *okresem*. Okres istnieje dla każdej liczby naturalnej $M > 1$ i jest podzielnikiem $\varphi(M)$. Dla niektórych wartości M można także wyróżnić *półokres* potęg 2 modulo M. Półokresem jest najmniejsza odległość pomiędzy resztami potęg 2, które są odwrotnościami addytywnymi, czyli ich suma modulo M wynosi 0. Jeśli istnieje półokres potęg 2 modulo M, jest on zawsze równy połowie okresu. Przykładem występowania okresu i półokresu może być ciąg reszt kolejnych potęg 2 modulo 9: $2^0 \bmod 9 = 1, 2^1 \bmod 9 = 2, 2^2 \bmod 9 = 4, 2^3 \bmod 9 = 8, 2^4 \bmod 9 = 7, 2^5 \bmod 9 = 5, 2^6 \bmod 9 = 1, \ldots$. Okres potęg 2 modulo 9 wynosi 6, a półokres jest równy 3.

Resztowy system liczbowy (RNS) [SJJT86], [ST67] jest określony przez zbiór n wzajemnie względnie pierwszych liczb naturalnych $M_i > 1$. Zakres dynamiczny systemu jest określony jako

$$M = \prod_{i=1}^{n} M_i. \tag{2.20}$$

Reprezentacją liczby całkowitej X w RNS$(M_1, M_2, \ldots, M_n)$ jest wektor $(X_1, X_2, \ldots, X_n)$ reszt $X_i = |X|_{M_i}$. Kolejne reszty X_i są także nazywane cyframi w RNS. Zgodnie z chińskim twierdzeniem o resztach (*Chinese Remainder Theorem, CRT*) wektor reszt $(X_1, X_2, \ldots, X_n)$ jest niepowtarzalny dla każdej liczby z przedziału $[Z, Z + M)$ dla Z całkowitego [Knu02], [Kob95].

Unikalność reprezentacji RNS można wykazać przez zaprzeczenie. Załóżmy, że dla różnych liczb $X, Y \in [Z, Z + M)$ ich reszty $X_i = |X|_{M_i}$ i $Y_i = |Y|_{M_i}$ są sobie równe dla każdego $i \in [1, n]$. Na

podstawie definicji kongruencji wynika stąd, że $|X - Y|_{M_i} = 0$ dla $i \in [1, n]$. Jeśli różnica $X - Y$ jest podzielna przez każdy moduł M_i z osobna, musi być także podzielna przez najmniejszą wspólną wielokrotność (NWW) tych modułów. Ponieważ moduły M_i są wzajemnie względnie pierwsze, ich NWW równa jest M, co daje $|X - Y|_M = 0$. Wynika stąd, że liczby X i Y muszą być równe lub co najmniej jedna z nich musi być spoza przedziału $[Z, Z + M)$, co przeczy początkowemu założeniu.

Zaletą RNS jest możliwość wykonywania operacji $\{+, -, \times\}$ niezależnie dla poszczególnych cyfr, zgodnie z równaniem

$$(X \odot Y) \equiv W \bmod M \Leftrightarrow (X_i \odot Y_i) \equiv W_i \bmod M_i, \tag{2.21}$$

gdzie $W_i = |W|_{M_i}$. Pozwala to na dekompozycję operacji na liczbach X, Y na zbiór niezależnych, równoległych kanałów, w których działania arytmetyczne przeprowadzane są modulo kolejne moduły M_i. Jeżeli wartości modułów M_i są znacznie mniejsze od wartości X, Y, koszt układu arytmetycznego złożonego z niezależnych jednostek dla poszczególnych modułów jest znacznie mniejszy od kosztu układu dla liczb X, Y. Niestety, istnieją operacje arytmetyczne, dla których implementacja w RNS jest niezwykle kosztowna. Poza tym, w większości obecnie stosowanych systemów cyfrowych używana jest arytmetyka pozycyjna, co wymusza konieczność *konwersji* pomiędzy RNS a systemami pozycyjnymi.

Przy analizie resztowych układów arytmetycznych przydatne jest także pojęcie *kompozycji* liczby naturalnej. Kompozycją $k \in \mathbb{N}$ nazywa się sposób podziału liczby k na sumę składników większych od 0. Cechą kompozycji jest to, że kolejność składników jest istotna. Dwie kompozycje o tych samych wartościach elementów, ale różnej ich kolejności, są różnymi kompozycjami tej samej liczby, np: $(2, 3)$ i $(3, 2)$ są różnymi kompozycjami liczby 5.

Niech $\Gamma(k) = \{(\mathbf{\Gamma}^i)\}$ oznacza zbiór wszystkich kompozycji k nie zawierających elementów równych 1. Pojedyncza, i–ta kompozycja jest opisana wektorem $\mathbf{\Gamma}^i$, przy czym $\sum_j \gamma_j^i = k$ i $\gamma_j^i > 1$. W pracy [Gri01] wykazano, że liczba takich kompozycji równa jest F_{k-1}, gdzie F_{k-1} oznacza $(k-1)$ element ciągu Fibonacciego. Elementy ciągu Fibonacciego zdefiniowane są za pomocą zależności rekurencyjnej

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_j &= F_{j-1} + F_{j-2} \quad \text{dla } j > 1 \end{aligned} \quad . \tag{2.22}$$

Wartość F_j można także opisać w postaci zwartej. Załóżmy, że wartość dowolnego wyrazu F_j jest

opisana wyrażeniem

$$F_j = \alpha \cdot \mu^j + \beta \cdot \delta^j \tag{2.23}$$

dla $\alpha, \beta, \mu, \delta$ rzeczywistych. Ze wzorów (2.22) i (2.23) wynika układ równań

$$\begin{cases} \alpha \cdot \mu^0 + \beta \cdot \delta^0 &= 0 \\ \alpha \cdot \mu^1 + \beta \cdot \delta^1 &= 1 \\ \alpha \cdot \mu^2 + \beta \cdot \delta^2 &= 1 \\ \alpha \cdot \mu^j + \beta \cdot \delta^j &= \alpha \cdot \mu^{j-1} + \beta \cdot \delta^{j-1} + \alpha \cdot \mu^{j-2} + \beta \cdot \delta^{j-2} \end{cases}, \tag{2.24}$$

który jest spełniony dla

$$\begin{cases} \alpha &= -1/\sqrt{5} \\ \beta &= 1/\sqrt{5} \\ \mu &= \frac{1-\sqrt{5}}{2} \\ \delta &= \frac{1+\sqrt{5}}{2} \end{cases}. \tag{2.25}$$

Wartość dowolnego elementu ciągu Fibonacciego wynosi zatem

$$F_j = \frac{1}{\sqrt{5}} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^j - \left(\frac{1-\sqrt{5}}{2} \right)^j \right). \tag{2.26}$$

Wzór (2.26) można przekształcić do prostszej postaci. Ponieważ

$$\lim_{j \to \infty} \left(\frac{1-\sqrt{5}}{2} \right)^j = 0,$$

więc j-ty element ciągu jest bliski $\frac{\phi^j}{\sqrt{5}}$, gdzie ϕ oznacza liczbę Fidiasza, czyli $\frac{1+\sqrt{5}}{2}$. Co więcej, różnica pomiędzy F_j a $\frac{\phi^j}{\sqrt{5}}$ nie przekracza $1/2$, ponieważ dla $j \geq 0$ jest

$$\left| \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2} \right)^j \right| < \frac{1}{2}.$$

Wynika stąd, że liczba kompozycji liczby k nie zawierających elementów równych 1 wynosi

$$|\Gamma(k)| = F_{k-1} = \left\lfloor \frac{\phi^{k-1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor. \tag{2.27}$$

Najprostszą metodą wyznaczenia zbioru $\Gamma(k)$ jest wygenerowanie wszystkich $k/2$ cyfrowych liczb w systemie pozycyjnym o podstawie k i wybór tych, dla których suma cyfr wynosi k. Niestety, metoda ta jest niezwykle czasochłonna. Znacznie szybszy algorytm zaprezentowano w pracy [Gri01]. W algorytmie tym zbiór $\Gamma(k)$ dla $k > 3$ jest określony przez przekształcenie $\Gamma(k-1)$ i

$\Gamma(k-2)$, przy czym $\Gamma(2) = \{(2)\}$ i $\Gamma(3) = \{(3)\}$. Pierwszym krokiem przekształcenia jest dopisanie 2 na ostatniej pozycji każdego wektora ze zbioru $\Gamma(k-2)$. Drugim krokiem jest dodanie 1 do ostatniej pozycji każdego wektora ze zbioru $\Gamma(k-1)$. Zbiory $\Gamma(k)$ dla kilku kolejnych wartości k przyjmują więc postać

$$\begin{aligned} \Gamma(2) &= (2) \\ \Gamma(3) &= (3) \\ \Gamma(4) &= (2,2),(4) \\ \Gamma(5) &= (3,2),(2,3),(5) \\ \Gamma(6) &= (2,2,2),(4,2),(3,3),(2,4),(6) \\ &\dots \end{aligned} \tag{2.28}$$

2.2.2 Konwersja pomiędzy RNS a systemem pozycyjnym

Konwersja liczby zapisanej w systemie pozycyjnym do RNS wymaga wyznaczenia reszt modulo kolejne moduły M_i stanowiące bazę RNS. Konwersja do RNS jest z reguły przeprowadzana dla liczb zapisanych w systemach binarnych, może zatem być zrealizowana za pomocą wielooperandowego sumatora modulo M_i dodającego reszty obliczone dla poszczególnych bitów reprezentacji binarnej. Dla niektórych wartości modułów technika ta pozwala na bardzo efektywną implementację. Przykładem mogą być moduły postaci $2^k - 1$, gdzie wyznaczenie reszty sprowadza się do sumowania pól zawierających k sąsiednich bitów reprezentacji binarnej.

Proces konwersji reprezentacji w RNS do reprezentacji w systemie pozycyjnym nazywany jest *konwersją odwrotną*. Klasyczną techniką konwersji odwrotnej jest konwersja odwrotna według CRT przeprowadzana zgodnie z wzorem

$$X = \left| \sum_{i=1}^{n} \frac{M}{M_i} \cdot \left| \left(\frac{M}{M_i} \right)^{-1} \right|_{M_i} \cdot X_i \right|_M . \tag{2.29}$$

Idea równania (2.29) polega na rozwiązaniu problemu dla liczb reprezentowanych w RNS jako $(1,0,...)$, $(0,1,0,...)$ itd. Liczby te nazywane są *wagami*, a ich wartości wynoszą $\frac{M}{M_i} \cdot \left| \left(\frac{M}{M_i} \right)^{-1} \right|_{M_i}$. Ponieważ kongruencje można dodawać i mnożyć, wartość dowolnej liczby w RNS jest następnie dana jako

$$(X_1, X_2, ..., X_n) = |X_1 \cdot (1,0,...,0) + \ldots + X_n \cdot (0,...,1)|_M . \tag{2.30}$$

Iloczyny reszt z odpowiadającymi im wagami nazywane są *projekcjami ortogonalnymi*.

Drugą metodą konwersji odwrotnej jest wykorzystanie stowarzyszonego systemu z mieszanymi podstawami [BJ78], [Hua83], [MM98], [GJ99]. Systemy liczbowe nazywamy stowarzyszonymi wtedy, gdy posiadają identyczną bazę lub zbiór cyfr. Liczba w systemie z mieszanymi podstawami (*ang.* ***M****ixed* ***R****adix* ***S****ystem, MRS*) stowarzyszonym z RNS jest reprezentowana przez n cyfr a_i. Cyfry a_i są określone wzorem

$$a_i = |r_i|_{M_i}, \tag{2.31}$$

gdzie r_i opisane jest zależnością rekurencyjną

$$\begin{array}{rcl} r_1 & = & X \\ r_i & = & (r_{i-1} - a_{i-1}) \cdot |(M_{i-1})^{-1}|_{M_i} \end{array}. \tag{2.32}$$

Wartością liczby w MRS jest

$$X = \sum_{i=1}^{n} a_i \cdot (\prod_{j=1}^{i-1} M_i). \tag{2.33}$$

Reprezentacja w MRS jest szeroko stosowana do detekcji znaku, wykrywania przepełnień, wykrywania i korekcji błędów itd. Dla reprezentacji w MRS można wykonać wiele operacji, których implementacja w RNS jest kosztowna. Pozwala to uniknąć pełnej konwersji do systemu binarnego.

Konwersja odwrotna może także być przeprowadzona według koncepcji określanej jako nowe chińskie twierdzenie o resztach II [Wan00]. Zgodnie z twierdzeniem przedstawionym i udowodnionym w [Wan00] konwersja odwrotna dla liczby naturalnej $X \in [0, M_1 \cdot M_2)$ zapisanej w RNS o bazie (M_1, M_2) jako (X_1, X_2) dla $X_1 = |X|_{M_1}, X_2 = |X|_{M_2}$ może być przeprowadzona zgodnie z wzorem

$$X = X_2 + \left| \left| M_2^{-1} \right|_{M_1} (X_1 - X_2) \right|_{M_1} \cdot M_2. \tag{2.34}$$

Wzór (2.34) umożliwia przeprowadzenie konwersji odwrotnej także dla RNS o bazie złożonej z wielu modułów. Moduły te są grupowane w pary i dla każdej z par przeprowadzana jest konwersja odwrotna w systemie o bazie określonej przez tę parę. Wynik konwersji dla każdej z par jest następnie traktowany jako reszta modulo iloczyn modułów wchodzących w skład pary. Uzyskane reszty są następnie znów łączone w pary, dla których ponownie jest przeprowadzana konwersja odwrotna zgodnie z równaniem (2.34). Zaletą metody zaprezentowanej w [Wan00] jest ograniczenie wartości modułów, dla których wykonywane są operacje modulo.

Oprócz przedstawionych metod istnieje jeszcze szereg innych rozwiązań będących modyfikacją powyższych, zarówno dla dowolnych RNS, jak i dla ściśle określonych baz [WSA99], [Pre92], [Pre95], [Pre04], [CCS03], [WSAW03], [CRL98], [WSAW00]. Szczególnie duża liczba możliwych

struktur konwerterów odwrotnych została opracowana dla RNS o bazie złożonej z modułów $(2^k - 1, 2^k, 2^k + 1)$ [CN99], [BPS98], [Moh01], [IS88], [Pie95], [Ber85], [VR94], [Moh98], [WJM00], [GPS97], [BWAK04].

Sprzętowe implementacje układów konwersji

W sprzętowych implementacjach układów arytmetyki resztowej pojawia się często konieczność obliczania wartości skomplikowanych wyrażeń arytmetycznych wymagających wyznaczania reszty, odwrotności multiplikatywnych modulo bądź logarytmu dyskretnego. Jedną z możliwości implementacji tych operacji w szybkich układach cyfrowych jest użycie pamięci ROM. Wewnątrz pamięci zawarte są wyniki danego działania dla wszystkich kombinacji argumentów. Z wartości argumentów jest konstruowany adres komórki zawierającej gotowy wynik. Metoda ta pozwala na bardzo szybką implementację dowolnej operacji arytmetycznej. Podstawową jej wadą jest szybki wzrost rozmiaru pamięci w funkcji maksymalnej wartości adresu pamięci.

W implementacjach konwerterów wykorzystujących CRT pamięci ROM mogą być wykorzystane do wyznaczania wartości projekcji ortogonalnych dla reszt X_i. Projekcje ortogonalne są następnie sumowane za pomocą wielooperandowego sumatora modulo M. Na potrzeby implementacji sprzętowej wzór (2.29) jest często przekształcany do postaci

$$X = \left| \sum_{i=1}^{n} \frac{M}{M_i} \cdot \left| \left| \left(\frac{M}{M_i} \right)^{-1} \right|_{M_i} \cdot X_i \right|_{M_i} \right|_M . \tag{2.35}$$

W konwerterach wykorzystujących wzór (2.35) pamięci ROM zawierają wartości $\left| \left| \left(\frac{M}{M_i} \right)^{-1} \right|_{M_i} \cdot X_i \right|_{M_i}$. Wartości te są znacznie mniejsze od pełnych projekcji ortogonalnych określonych jako $\frac{M}{M_i} \cdot \left| \left(\frac{M}{M_i} \right)^{-1} \right|_{M_i} \cdot X_i$. Pozwala to na kilkukrotne zmniejszenie rozmiaru pamięci. Oczywiście rozmiar pamięci nadal zależy wykładniczo od szerokości poszczególnych modułów M_i, zarówno dla implementacji opartych o wzór (2.29), jak i (2.35).

W przypadku konwerterów wykorzystujących MRC pamięci ROM są wykorzystywane do wyznaczania wartości r_i oraz a_i. Umożliwia to zrealizowanie konwersji z RNS do MRS w postaci układu potokowego bez dodatkowych układów arytmetycznych.

W konwerterach zaprezentowanych w pracy [Wan00] stosowane są drzewiaste struktury zbudowane z sumatorów modulo. Liczba poziomów równa jest $\log_2(n)$, natomiast każdy z sumatorów

wykonuje operacje modulo liczby, których wartość rośnie na kolejnych poziomach od M_i do ok. $\sqrt{M}$. W porównaniu z CRT umożliwia to zmniejszenie złożoności układu dzięki wyeliminowaniu wielooperandowego sumatora modulo M.

Struktury konwerterów zaprojektowanych dla wybranych, ściśle określonych RNS mogą znacznie różnić się od rozwiązań uniwersalnych wykorzystujących bezpośrednio CRT bądź MRS. Konwertery opracowane dla pojedynczych klas RNS są także mniejsze i szybsze od struktur, które można zastosować dla dowolnego RNS. Dla przykładu, w RNS o bazie $(2^k - 1, 2^k, 2^k + 1)$ osiągnięcie dużego zakresu dynamicznego wymaga stosowania odpowiednio dużego k. Wykorzystanie pamięci ROM jest więc niepraktyczne ze względu na wymagany rozmiar. Z tego powodu opracowane układy konwersji odwrotnej dla RNS $(2^k - 1, 2^k, 2^k + 1)$ z reguły zawierają wyłącznie układy arytmetyczne [BPS98], [GPS97], [AA88], [IS88], [Pie95], [Dhu98], [WJM00], [WSAS02].

2.2.3 Resztowe układy arytmetyczne

Operacje arytmetyczne przeprowadzane na liczbach zapisanych w RNS dzieli się na dwie kategorie: *łatwe* i *trudne* w zależności od kosztu implementacji. Do operacji łatwych zalicza się dodawanie, odejmowanie i mnożenie, natomiast pozostałe działania, w tym dzielenie, skalowanie, wykrywanie nadmiaru addytywnego i multiplikatywnego i testowanie znaku należą do kategorii trudnych w RNS. Operacje łatwe w RNS wykonywane są dla każdej cyfry niezależnie, tak więc resztowe sumatory, subtraktory i układy mnożące składają się z zestawu odpowiednich układów arytmetycznych wykonujących operacje modulo poszczególne moduły M_i.

Problem sprzętowej implementacji sumatorów i układów mnożących modulo był wielokrotnie poruszany przez wielu autorów. Zdecydowana większość rozwiązań dotyczy jednak struktur dedykowanych do implementacji w układach ASIC, w związku z czym ich implementacja w układach FPGA jest nieefektywna. Część z istniejących rozwiązań dla ASIC może jednak zostać zaadaptowana na potrzeby implementacji w FPGA, jest to jednak możliwe jedynie w szczególnych przypadkach, np. dla specyficznych wartości modułu. Przykładem są jednostki mnożenia modulo $2^k \pm 1$ opisane w pracach [Beu02] i [Zim99]. Podstawą konstrukcji obu tych układów jest formuła

$$X \cdot Y = 2^k \cdot \left\lfloor \frac{X \cdot Y}{2^k} \right\rfloor + |X \cdot Y|_{2^k} \tag{2.36}$$

wynikająca bezpośrednio z definicji reszty z dzielenia (wzór (2.14)).

Algorytm mnożenia modulo $2^k - 1$ według pracy [Zim99] jest opisany wzorem

$$|X \cdot Y|_{2^k-1} = \left| |X \cdot Y|_{2^k} + \left\lfloor \frac{X \cdot Y}{2^k} \right\rfloor \right|_{2^k-1}. \tag{2.37}$$

Zgodnie z (2.37) mnożenie modulo 2^k-1 może być zaimplementowane za pomocą matrycy mnożącej i sumatora modulo. W pracy [Zim99] zaproponowano strukturę pozwalającą na redukcję modulo na etapie tworzenia iloczynów częściowych, dzięki czemu zmniejszono liczbę koniecznych sumatorów modulo. Niestety, rozwiązanie to jest bardzo nieefektywnie implementowane w układach FPGA, ponieważ wymaga wytwarzania iloczynów częściowych dla każdego bitu mnożnika osobno. W związku z tym na potrzeby tej pracy układy mnożenia modulo $2^k - 1$ są implementowane w postaci bezpośrednio wynikającej (2.37). Pozwala to na wykorzystanie obecnej w FPGA dedykowanej logiki zwiększającej efektywność implementacji układów arytmetycznych.

Układ mnożący modulo $2^k + 1$ zaprezentowany w pracy [Beu02] wykonuje operację

$$|X \cdot Y|_{2^k+1} = \left| |X \cdot Y|_{2^k} - \left\lfloor \frac{X \cdot Y}{2^k} \right\rfloor \right|_{2^k+1}. \tag{2.38}$$

Implementacja zaprezentowana w [Beu02] zawiera matrycę mnożącą, multiplekser i sumator modulo 2^k+1. W układzie z [Beu02] można pominąć multiplekser, co pozwala zmniejszyć zajmowany obszar i wprowadzane opóźnienia kosztem wprowadzenia podwójnej reprezentacji zera.

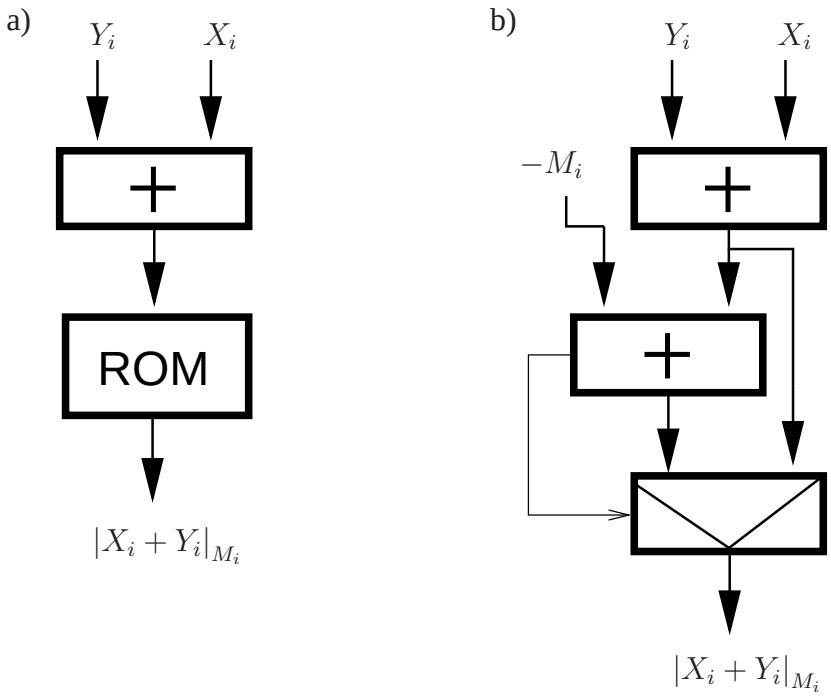
W ogólnym przypadku układy arytmetyczne dedykowane dla ASIC są optymalizowane na poziomie wykorzystania pojedynczych ogniw sumatora, bramek logicznych bądź nawet tranzystorów [PKS01]. Metody te nie dają zadowalających wyników dla FPGA, dlatego też opracowano szereg rozwiązań dedykowanych dla tej platformy. Poniżej zostaną przedstawione dotychczasowe propozycje resztowych układów arytmetycznych dla matryc FPGA.

Najprostszą koncepcyjnie metodą implementacji dodawania, odejmowania lub mnożenia modulo jest zastosowanie pojedynczej pamięci ROM zawierającej wyniki dla wszystkich kombinacji słów wejściowych. Rozmiar pamięci może zostać zmniejszony dzięki wykorzystaniu technik pozwalających znaleźć optymalną zawartość pamięci dla występujących kombinacji operandów wejściowych, zarówno dla układów resztowych [Ven96], jak i w ogólnym przypadku [Łub03]. Niestety, zależność rozmiaru pamięci ROM od liczby bitów użytych do reprezentacji modułu jest wykładnicza. Z tego powodu rozwiązanie to jest stosowane wyłącznie dla niewielkich wartości modułu.

Dodawanie i odejmowanie modulo można także zaimplementować za pomocą układów zaprezentowanych w [Beu03]. Układy te są strukturami dwupoziomowymi (rys. 2.10). Na pierwszym poziomie znajduje się zwykły sumator lub subtraktor, zadaniem drugiego poziomu jest znalezienie reszty

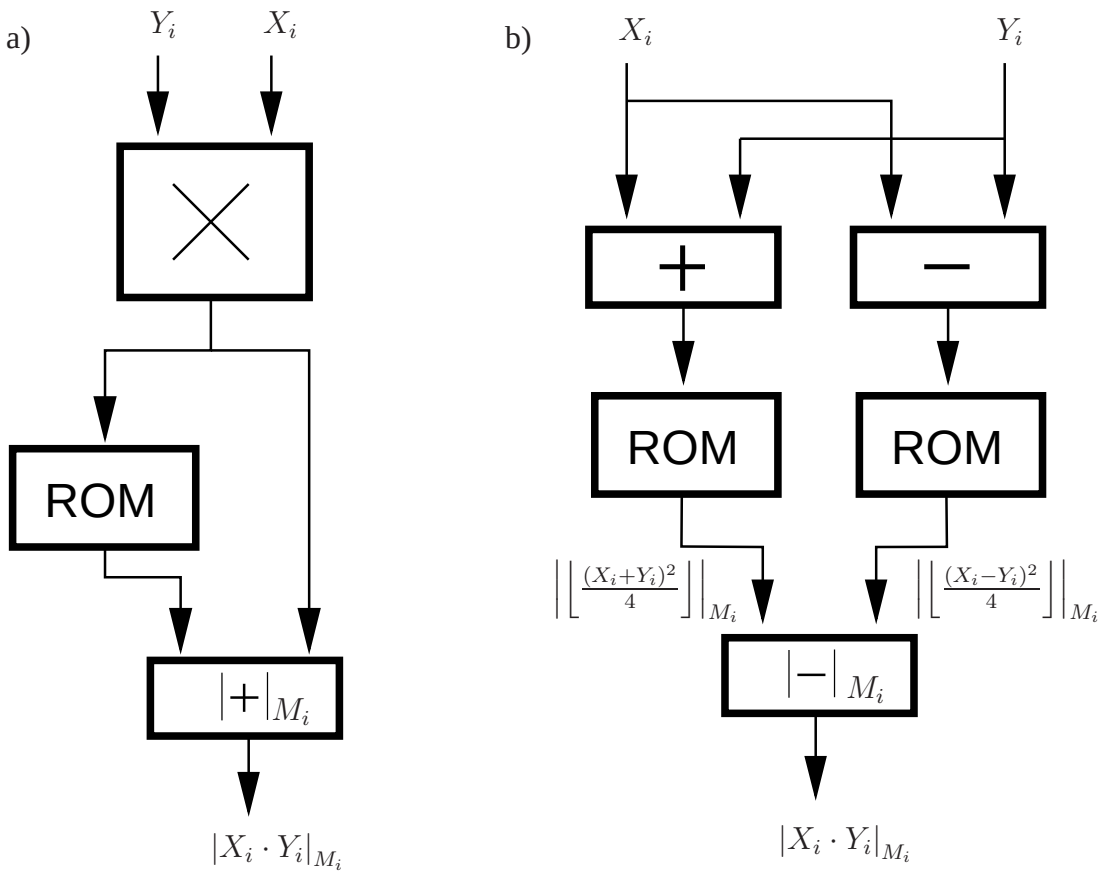
dla sumy lub różnicy obliczonej na pierwszym poziomie. W ogólnym przypadku reszty dla wszystkich wartości sumy/różnicy mogą być zapamiętane w pamięci ROM (rys. 2.10 a)), lecz możliwe jest także zastosowanie rozwiązania o mniejszej złożoności (rys. 2.10 b)). Wykorzystywany jest fakt, że dla dodawania lub odejmowania dwuoperandowego wynik nigdy nie przekroczy dwukrotnej wartości modułu. Można zatem wytworzyć dwa wyniki: pierwszy jest bezpośrednim rezultatem dodawania/odejmowania operandów, drugi wynik jest rezultatem dodawania/odejmowania skorygowanym o wartość modułu. Spośród tych dwóch wartości jest następnie wybierana ta, która mieści się w zakresie dopuszczalnych wartości dla reszty modulo. Operację tę dla dodawania można zapisać jako

$$|X_i + Y_i|_{M_i} = \begin{cases} X_i + Y_i & \text{dla} \quad X_i + Y_i < M_i \\ X_i + Y_i - M_i & \text{dla} \quad X_i + Y_i \geq M_i \end{cases} . \tag{2.39}$$



Rysunek 2.10. Implementacja sumatorów modulo w układach FPGA.

Dla układów mnożących opracowano wiele różnorodnych struktur układowych. W pracy [Beu03] mnożenie modulo realizowane jest za pomocą pełnej matrycy mnożącej, po której następuje korekcja wyniku realizowana za pomocą sumatora modulo i pamięci ROM wyznaczającej resztę dla wyższej części wyniku. Schemat układu znajduje się na rys. 2.11 a).



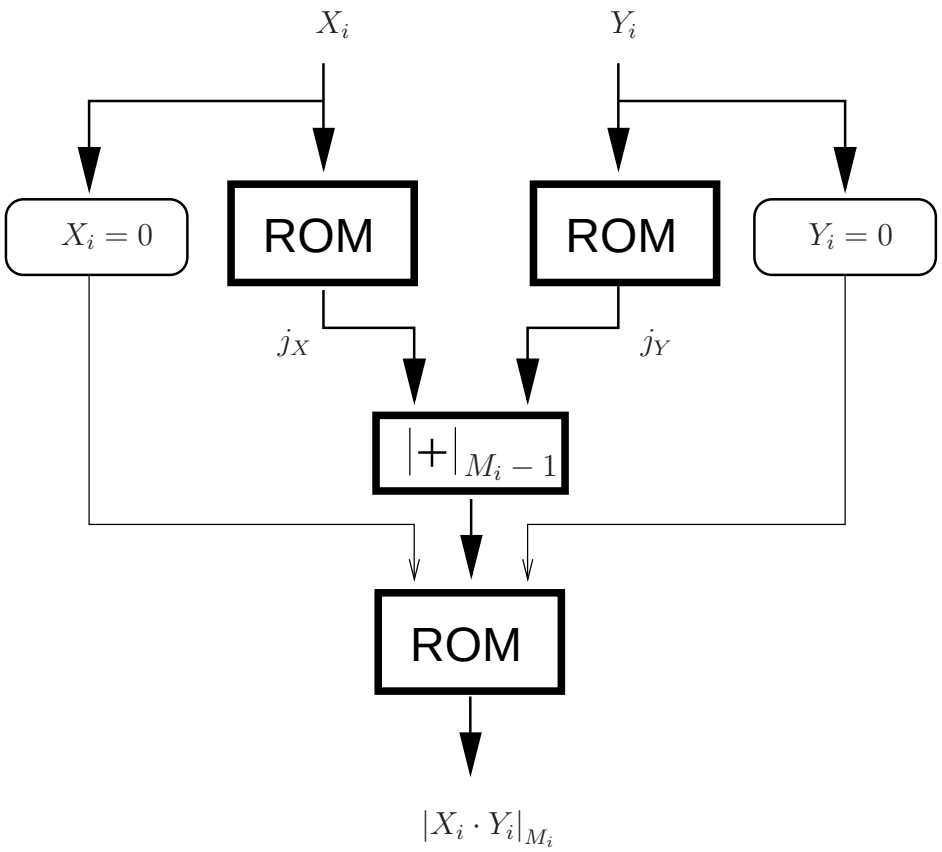
Rysunek 2.11. Implementacja układów mnożących modulo modulo w układach FPGA.

Dla modułów będących liczbami pierwszymi można wykorzystać wzór (2.18) wynikający z małego twierdzenia Fermata (str. 43). Wadą tej metody jest ograniczenie zbioru modułów do liczb pierwszych oraz wysoki koszt poszerzenia układu o możliwość dodawania. Układ mnożący zbudowany zgodnie z wzorem (2.18) zaprezentowano w pracy [MFAA98]. Schemat układu znajduje się na rys. 2.12. Składa się on z dwóch pamięci ROM przekodowujących operandy X_i i Y_i na odpowiadające im indeksy j_X, j_Y, sumatora modulo $M_i - 1$, końcowej tablicy ROM zawierającej wynik mnożenia dla obliczonej sumy indeksów oraz dwóch dodatkowych detektorów zera dla operandów wejściowych.

Kolejną metodą pozwalającą na zastąpienie mnożenia dodawaniem jest wykorzystanie prawa różnicy kwadratów (*ang. quarter-square multiplier*) [MBGT01]. W metodzie tej mnożenie przeprowadzane jest według wzoru

$$|X_i \cdot Y_i|_{M_i} = \left| \left| \left\lfloor \frac{(X_i + Y_i)^2}{4} \right\rfloor \right|_{M_i} - \left| \left\lfloor \frac{(X_i - Y_i)^2}{4} \right\rfloor \right|_{M_i} \right|_{M_i}. \tag{2.40}$$

Schemat układu realizującego mnożenie zgodnie z wzorem (2.40) przedstawiono na rys. 2.11 b).



Rysunek 2.12. Jednostka mnożąca modulo wykorzystująca małe twierdzenie Fermata.

Równanie (2.40) implementuje się z użyciem sumatora i subtraktora binarnego, dwóch pamięci ROM przekodowujących sumę i różnicę X_i i Y_i na $\left|\left\lfloor \frac{(X_i \pm Y_i)^2}{4} \right\rfloor\right|_{M_i}$ oraz końcowego subtraktora modulo M_i. Metoda ta jest pozbawiona wad układu z rys. 2.12.

Znane są także resztowe układy arytmetyczne w postaci struktur potokowych pozwalających na szeregowe wykonywanie operacji [MB01], [BM04]. Charakteryzują się one mniejszym obszarem przy ograniczonej wydajności.

2.2.4 Detekcja znaku w RNS

Z powodu złożoności implementacji operacje trudne w RNS powinny być wyeliminowane z resztowego toru obliczeniowego. Niestety, istnieją zastosowania RNS, w których całkowite usunięcie operacji trudnych jest niemożliwe. Przykładem może być arytmetyka dokładna (*ang. exact arithmetic*) lub geometria obliczeniowa (*ang. computational geometry*) [BEPP97]. Jednym z częściej stosowanych algorytmów trudnych w RNS jest detekcja znaku, która znajduje zastosowanie zarówno w operacjach

dotyczących samego RNS (dzielenie i porównywanie liczb), jak i wielu innych zastosowaniach.

Problem detekcji znaku w RNS był wielokrotnie poruszany przez wielu autorów. W najprostszej wersji może być zrealizowany poprzez konwersję odwrotną i porównanie otrzymanej wartości z odpowiednią liczbą, zazwyczaj równą połowie zakresu dynamicznego. Znane są także inne algorytmy, pozwalające na bardziej efektywną implementację. W pracy [Ulm83] funkcja detekcji znaku jest sumą modulo 2 cyfr reprezentacji w stowarzyszonym MRS. Wadą tej metody są ograniczenia na wartości modułów stanowiących bazę RNS. W pracy [Vu85] znak liczby jest określony przez najwyższy bit wielooperandowej sumy modulo 2^k argumentów uzyskanych z reszt w RNS. Koncepcja zaprezentowana w [AM98] wykorzystuje RNS o bazie rozszerzonej o dodatkowy moduł i wymaga zastosowania dwóch wielooperandowych sumatorów modulo i dwóch układów mnożących modulo. Opracowano także szereg algorytmów detekcji znaku implementowanych programowo, ale charakteryzują się one znacznie większą złożonością, np.: pomysł opisany w [DJM98] wymaga przeprowadzania obliczeń w rozszerzonym RNS o zakresie równym kwadratowi zakresu RNS podstawowego.

Z uwagi na znaczny koszt układu detekcji znaku jego stosowanie w RNS jest nieopłacalne. Wyjątkiem są pewne klasy systemów resztowych, np. RNS $(2^k-1, 2^k, 2^k+1)$, dla których możliwa jest efektywna implementacja niektórych operacji trudnych. Prezentowana poniżej nowa metoda detekcji znaku w porównaniu z dotychczasowymi rozwiązaniami pozwala ograniczyć obszar układu przy jednoczesnym zmniejszeniu długości ścieżki krytycznej [Tom05a]. Charakterystyki AT układu są lepsze od charakterystyk konwerterów odwrotnych dla RNS $(2^k-1, 2^k, 2^k+1)$.

Liczby ze znakiem ukrytym w RNS

Niech W oznacza dowolną liczbę całkowitą z przedziału $[Z, Z+M)$ dla Z całkowitego. Zgodnie z chińskim twierdzeniem o resztach istnieje unikatowa reprezentacja W w RNS o zakresie dynamicznym M. Konsekwencją jest możliwość wyboru dowolnego Z, a więc w szczególności także $Z = -\left\lfloor \frac{M}{2} \right\rfloor$. Dla tej wartości Z zakres reprezentowanych liczb W jest całkowicie symetryczny względem zera dla M nieparzystych, natomiast dla M parzystych nie ma możliwości reprezentacji jedynie odwrotności addytywnej dla $W = -\frac{M}{2}$. Uzyskany zakres $W \in [-\left\lfloor \frac{M}{2} \right\rfloor, M - \left\lfloor \frac{M}{2} \right\rfloor)$ umożliwia reprezentowanie liczb ze znakiem w RNS.

Reprezentacją dowolnej liczby całkowitej W w RNS jest wektor reszt $(W_1, W_2, \ldots, W_n)$ tej liczby modulo kolejne moduły M_i. Reprezentacja ta jest identyczna z reprezentacją $(X_1, X_2, \ldots, X_n)$ przystającej do W modulo M liczby naturalnej $X \in [0, M)$. Dla $W \in [-\left\lfloor \frac{M}{2} \right\rfloor, M - \left\lfloor \frac{M}{2} \right\rfloor)$ relacja

pomiędzy W i X jest opisana jako

$$X = \begin{cases} W & \text{dla} \quad W \in [0, M - \lfloor \frac{M}{2} \rfloor) \\ M + W & \text{dla} \quad W \in [-\lfloor \frac{M}{2} \rfloor, 0) \end{cases} . \tag{2.41}$$

W ogólnym przypadku zależność tę określa wzór

$$W = |X - Z|_M + Z. \tag{2.42}$$

Wzór (2.42) opisuje sposób reprezentacji liczb ze znakiem ukrytym w RNS. Konwersja wejściowa sprowadza się do wygenerowania dla liczby W dodatnich reszt modulo moduły M_i. Konwersja wyjściowa może być przeprowadzona w dwóch krokach. Pierwszy z nich obejmuje znalezienie wartości X, w drugim uzyskana liczba X jest konwertowana do W zgodnie z wzorem (2.42). Wartość X może być obliczona z użyciem jednej z opisanych dotychczas metod konwersji odwrotnej.

Dla tak zdefiniowanych reprezentacji badanie znaku liczby można sprowadzić do zbadania przynależności W lub X do odpowiedniego przedziału. Dla RNS o symetrycznym zakresie $\left(Z = -\lfloor \frac{M}{2} \rfloor\right)$ funkcją znaku jest

$$sgn(W) = \begin{cases} 0 & \text{for} \quad W \in \left[0, M - \lfloor \frac{M}{2} \rfloor\right) \\ 1 & \text{for} \quad W \in \left[-\lfloor \frac{M}{2} \rfloor, 0\right) \end{cases} \tag{2.43}$$

lub

$$sgn(W) = sgn(X) = \begin{cases} 0 & \text{for} \quad X \in \left[0, \lceil \frac{M}{2} \rceil\right) \\ 1 & \text{for} \quad X \in \left[\lceil \frac{M}{2} \rceil, M\right) \end{cases} . \tag{2.44}$$

Równanie (2.44) jest podstawą proponowanego algorytmu detekcji znaku dla RNS $(2^k - 1, 2^k, 2^k + 1)$.

Liczby ze znakiem w RNS $(2^k - 1, 2^k, 2^k + 1)$

Dla RNS o bazie $(2^k - 1, 2^k, 2^k + 1)$ zakres dynamiczny systemu wynosi

$$M = \left(2^k - 1\right) \cdot 2^k \cdot \left(2^k + 1\right) = 2^{3k} - 2^k. \tag{2.45}$$

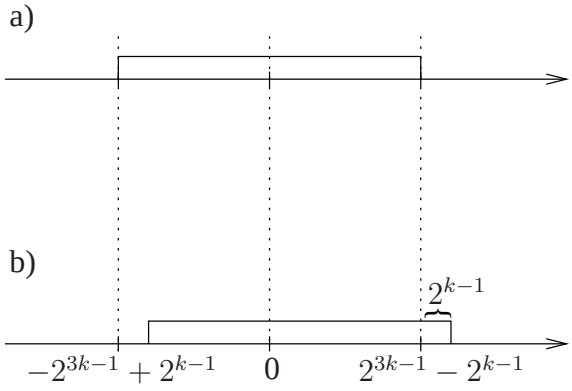
Przyjmując $Z = -2^{3k-1} + 2^{k-1}$, przedział reprezentowalnych liczb W jest określony jako $W : W \in \left[-2^{3k-1} + 2^{k-1}, 2^{3k-1} - 2^{k-1}\right)$. Funkcją znaku jest

$$sgn(X) = \begin{cases} 0 & \text{for} \quad X \in \left[0, 2^{3k-1} - 2^{k-1}\right) \\ 1 & \text{for} \quad X \in \left[2^{3k-1} - 2^{k-1}, 2^{3k} - 2^k\right) \end{cases} . \tag{2.46}$$

Podział zakresu dynamicznego M na liczby dodatnie i ujemne można przeprowadzić na wiele innych sposobów. Jedną z możliwości jest $Z = -2^{3k-1} + 2^k$, skąd zakres reprezentowalnych liczb $W \in \left[-2^{3k-1} + 2^k, 2^{3k-1}\right)$. Funkcja znaku

$$sgn'(X) = \left[X \geq 2^{3k-1}\right] \tag{2.47}$$

odpowiada wartości najwyższego bitu X. Podział taki ma jednak dwie wady. Po pierwsze otrzymany zakres W jest niesymetryczny (rys. 2.13), ponieważ nie obejmuje odwrotności addytywnych dla $W \in \left(2^{3k-1} - 2^k, 2^{3k-1} - 2^{k-1}\right)$. Po drugie, rozmiar konwertera odwrotnego obliczającego X jest większy od prezentowanego poniżej układu detekcji znaku.



Rysunek 2.13. Zakresy dynamiczne RNS dla a) $Z = -2^{3k-1} + 2^{k-1}$, b) $Z = -2^{3k-1} + 2^k$.

Jako przykład można rozważyć RNS$(2^4 - 1, 2^4, 2^4 + 1)$. Zakres dynamiczny systemu wynosi $M = 15 \cdot 16 \cdot 17 = 4080$. Jeśli funkcja znaku jest równa wartości najwyższego bitu X, to wartości $X \in [0, 2047]$ reprezentują liczby nieujemne W, a wartości $X \in [2048, 4079]$ reprezentują liczby ujemne $W \in [-2032, -1]$. Dla liczb $W \in [2033, 2047]$ nie można zapisać ich odwrotności addytywnych. Bardziej symetryczny podział uzyskuje się dla $Z = -2040$. W tym przypadku liczby nieujemne $W \in [0, 2039]$ są reprezentowane przez $X \in [0, 2039]$, a liczbom ujemnym $W \in [-2040, -1]$ odpowiadają wartości $X \in [2040, 4079]$.

Porównując równania (2.46) i (2.47) można zauważyć, że funkcje $sgn(X)$ i $sgn'(X)$ różnią się wartościami wyłącznie dla argumentów z zakresu

$$X \in \left[2^{3k-1} - 2^{k-1}, 2^{3k-1}\right). \tag{2.48}$$

Obserwacja ta została wykorzystana do konstrukcji funkcji znaku jako modyfikacji równania opisującego wartość najwyższego bitu X.

Algorytm detekcji znaku

Podstawą proponowanego w rozprawie algorytmu detekcji znaku dla RNS $(2^k - 1, 2^k, 2^k + 1)$ są równania konwersji odwrotnej wynikające z tzw. nowego chińskiego twierdzenia o resztach 2 (2.34). Niech $M_1 = 2^k - 1$, $M_2 = 2^k$, $M_3 = 2^k + 1$ oraz $X_i = |X|_{M_i}$. Pierwszym krokiem konwersji odwrotnej według (2.34) jest znalezienie reszty modulo $M_1 \cdot M_3$ równej

$$|X|_{M_1 \cdot M_3} = X_3 + \left| \left| M_3^{-1} \right|_{M_1} \cdot (X_1 - X_3) \right|_{M_1} \cdot M_3, \tag{2.49}$$

po czym wartość $X \in [0, M_1 \cdot M_2 \cdot M_3)$ dana jest jako

$$X = X_2 + \left| \left| M_2^{-1} \right|_{M_1 \cdot M_3} \cdot (|X|_{M_1 \cdot M_3} - X_2) \right|_{M_1 \cdot M_3} \cdot M_2. \tag{2.50}$$

Po podstawieniu odpowiednich wartości za M_i oraz rozwinięciu we wzorze (2.50) symbolu $|X|_{M_1 \cdot M_3}$ według (2.49) wartość X wynosi

$$X = X_2 + \left| 2^k \cdot \left(X_3 - X_2 + \left| 2^{k-1} \cdot (X_1 - X_3) \right|_{2^k - 1} \cdot \left(2^k + 1 \right) \right) \right|_{2^{2k-1}} \cdot 2^k. \tag{2.51}$$

Dla uproszczenia (2.51) korzystne jest wprowadzenie symboli

$$Y = \left| 2^{k-1} (X_1 - X_3) \right|_{2^k - 1}, \tag{2.52}$$

$$V = X_3 - X_2 + Y \left(2^k + 1 \right) \tag{2.53}$$

i

$$R = \left| 2^k \cdot V \right|_{2^{2k} - 1}. \tag{2.54}$$

Dzięki temu ostateczna wersja formuły opisującej wartość X może być zapisana jako

$$X = X_2 + 2^k \cdot R. \tag{2.55}$$

Ponieważ $X_2 < 2^k$, najwyższe $2k$ bitów X jest zależne wyłącznie od R. Najwyższy bit X może więc być określony jako *MSB*(R). Wzór (2.54) opisujący wartość R można zaimplementować jako rotację cykliczną w lewo o k bitów wektora reprezentującego

$$|V|_{2^{2k} - 1} = V + C. \tag{2.56}$$

Korekcja C występująca w równaniu (2.56) jest taką krotnością $2^{2k} - 1$, którą należy dodać do V, aby obliczyć $|V|_{2^{2k} - 1}$. Wartość C jest równa

$$C = - \left(2^{2k} - 1 \right) \cdot \left\lfloor \frac{V}{2^{2k} - 1} \right\rfloor. \tag{2.57}$$

Reprezentacją $|V|_{2^{2k}-1}$ jest wektor bitowy o szerokości co najwyżej $2k$ bitów. Po rotacji cyklicznej w lewo o k bitów, $(k-1)$ bit wektora reprezentującego $|V|_{2^{2k}-1}$ stanie się najwyższym bitem wektora reprezentującego R. Korzystając z (2.53), formułę (2.56) można rozwinąć do postaci

$$|V|_{2^{2k}-1} = X_3 - X_2 + Y + 2^k \cdot Y + C. \tag{2.58}$$

Wartość zapisana na k najniższych bitach wektora reprezentującego $|V|_{2^{2k}-1}$ jest więc równa $T + |C|_{2^k}$ dla

$$T = X_3 - X_2 + Y. \tag{2.59}$$

Funkcja określająca najwyższy bit X może być zatem zdefiniowana jako

$$\mathit{MSB}(X) = \mathit{MSB}(R) = \mathit{MSB}(|T + |C|_{2^k}|_{2^k}) = \left[|T + |C|_{2^k}|_{2^k} \geq 2^{k-1}\right]. \tag{2.60}$$

Formuła (2.60) pozwala określić wartość funkcji $sgn'(X)$ zdefiniowanej wzorem (2.47). Bardziej symetryczny podział zakresu zapewnia poszukiwana funkcja znaku $sgn(X)$, różniąca się od $sgn'(X)$ wyłącznie dla argumentów z przedziału danego przez (2.48). Poniżej zostanie wykazane, że jeżeli wartość C zostanie ustalona jako równa 0, wartości funkcji opisanej równaniem (2.60) ulegną zmianie wyłącznie dla argumentów z przedziału (2.48). Pozwoli to zdefiniować funkcję znaku $sgn(X)$ jako

$$sgn\,(X) = \left[|T|_{2^k} \geq 2^{k-1}\right]. \tag{2.61}$$

Twierdzenie 2.2.1. *Niech X będzie dowolną liczbą całkowitą, k liczbą naturalną większą od 1, $X_1 = |X|_{2^k-1}$, $X_2 = |X|_{2^k}$, $X_3 = |X|_{2^k+1}$, a $M = 2^k \cdot (2^k - 1) \cdot (2^k + 1)$. Reszta $|X|_M$ jest mniejsza od $\left\lceil \frac{M}{2} \right\rceil$ wtedy i tylko wtedy, gdy*

$$\left| X_3 - X_2 + \left|2^{k-1} \cdot (X_1 - X_3)\right|_{2^k-1} \right|_{2^k} < 2^{k-1}. \tag{2.62}$$

Dowód. Lewa strona nierówności (2.62) jest równa wartości T określonej przez (2.59), a więc nierówność (2.62) zależy od $\mathit{MSB}(|T|_{2^k})$. Z wzoru (2.60) wiadomo, że $\mathit{MSB}(|T + |C|_{2^k}|_{2^k}) = \mathit{MSB}(X)$, stąd $\mathit{MSB}(|T|_{2^k})$ i $\mathit{MSB}(X)$ są równe dla $C = 0$. Należy więc zbadać przypadki, w których $C \neq 0$.

Z wzoru (2.57) wynika, że korekcja C we wzorze (2.56) przyjmuje wartości różne od zera wyłącznie dla $V \notin [0, 2^{2k} - 1)$. Zakres możliwych wartości V zdefiniowanego wzorem (2.53) w zależności od Y jest dany jako:

1. jeśli $0 < Y < 2^k - 1$, to $V \in \left[2, 2^{2k} - 2\right]$:

$$\begin{aligned} V &\leq X_{3,max} - X_{2,min} + Y_{max} \cdot (2^k + 1) \\ V &\leq 2^k - 0 + \left(2^k - 2\right) \cdot \left(2^k + 1\right) \\ V &\leq 2^{2k} - 2 \end{aligned}$$

i

$$\begin{aligned} V &\geq X_{3,min} - X_{2,max} + Y_{min} \cdot \left(2^k + 1\right) \\ V &\geq 0 - \left(2^k - 1\right) + 1 \cdot \left(2^k + 1\right) \\ V &\geq 2, \end{aligned}$$

2. jeśli $Y = 0$, to $V \in \left(-2^k, 2^k\right]$:

   (a) jeśli $X_3 - X_2 \geq 0$, to $0 \leq V \leq 2^k$,

   (b) jeśli $X_3 - X_2 < 0$, to $-2^k < V < 0$.

Wartość $C \neq 0$ wystąpi zatem jedynie dla przypadku 2b), w którym $C = 2^{2k} - 1$.

We wzorze (2.60) wymagana jest reszta $|C|_{2^k} = |-1|_{2^k}$, tak więc dla przypadku 2b) wartość $\mathit{MSB}(X) = \mathit{MSB}(|T-1|_{2^k})$. Reprezentacje binarne T i $T-1$ różnią się jedynie na pozycjach obejmujących ciąg najmniej znaczących zer i pierwszej jedynki wektora reprezentującego T zgodnie z wzorem

$$\ldots t_{j+3} t_{j+2} t_{j+1} 1 \underbrace{00\ldots0}_{j}{}_2 - 1 = \ldots t_{j+3} t_{j+2} t_{j+1} 0 \underbrace{11\ldots1}_{j}{}_2, \tag{2.63}$$

gdzie t_j oznacza bit wektora $\mathbf{T}$ o wadze 2^j. Ponieważ różnica wartości T i $T + |C|_{2^k}$ może wystąpić wyłącznie w przypadku 2b), czyli dla $Y = 0$, warunek

$$\mathit{MSB}\left(|T|_{2^k}\right) \neq \mathit{MSB}\left(\left|T + |C|_{2^k}\right|_{2^k}\right) \tag{2.64}$$

wystąpi dla

$$X_3 - X_2 = 1\ldots1\underbrace{0\ldots0}_{k}{}_2 \tag{2.65}$$

lub

$$X_3 - X_2 = 1\ldots1\underbrace{10\ldots0}_{k}{}_2. \tag{2.66}$$

Sytuacja opisana przez (2.65) jest niemożliwa, ponieważ

$$X_3 \geq 0 \cap X_2 < 2^k \quad \Rightarrow \quad X_3 - X_2 \neq -2^k. \tag{2.67}$$

Wartość różnicy $X_3 - X_2 = -2^{k-1}$ określona przez (2.66) może wystąpić wyłącznie dla $X_2 \in \left[2^{k-1}, 2^k - 1\right]$. Mając zakres wartości X_2 oraz wartość różnicy $X_3 - X_2$, można z wzoru (2.55) znaleźć zakres wartości X, dla których zachodzi warunek (2.64):

$$X = X_2 + 2^k \cdot \left|2^k \left(X_3 - X_2\right)\right|_{2^{2k-1}} = X_2 + 2^k \cdot \left|2^k \left(-2^{k-1}\right)\right|_{2^{2k-1}} = X_2 + 2^{3k-1} - 2^k. \quad (2.68)$$

Ponieważ $X_2 \in \left[2^{k-1}, 2^k - 1\right]$, tak więc dla

$$X \in \left[2^{3k-1} - 2^{k-1}, 2^{3k-1} - 1\right] \quad (2.69)$$

warunek (2.64) jest spełniony, czyli MSB$(X) \neq$ MSB$(|T|_{2^k})$. W pozostałych przypadkach MSB$(X) =$ MSB$(|T|_{2^k})$. Ponieważ *MSB*$(X) = 0$ dla $X \in [0, 2^{3k-1})$, dla X z przedziału (2.69) najstarszy bit T będzie 1. Wynika stąd, że wartość $\left|X_3 - X_2 + \left|2^{k-1} \cdot (X_1 - X_3)\right|_{2^k-1}\right|_{2^k}$ jest mniejsza od 2^{k-1} dla $X < 2^{3k-1} - 2^{k-1}$. □

Struktura układu

Wartość funkcji znaku zdefiniowanej (2.61) jest określona przez $(k-1)$ bit T opisanego wyrażeniem (2.59). Na podstawie równań (2.59) i (2.52) wartość T jest określona formułą

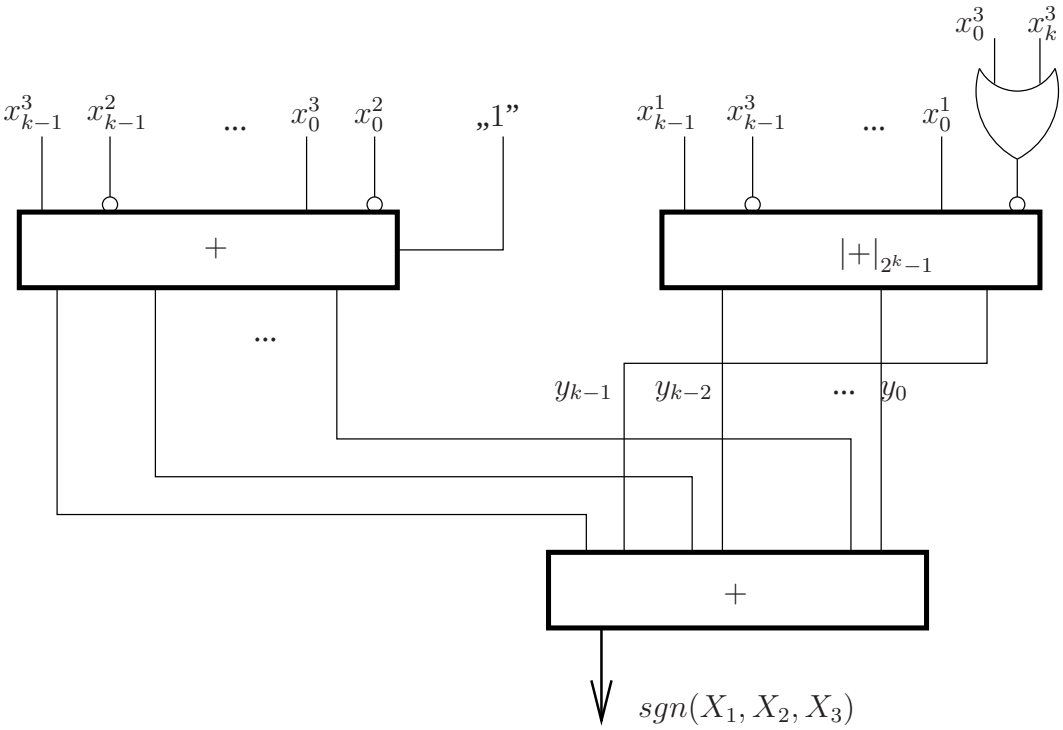
$$T = X_3 - X_2 + \left|2^{k-1} \left(X_1 - X_3\right)\right|_{2^k-1}. \quad (2.70)$$

Wzór (2.70) może zostać zaimplementowany za pomocą trzech sumatorów k–bitowych, z których jeden jest sumatorem modulo $2^k - 1$. Ponieważ warunkiem poprawności prezentowanej funkcji znaku jest $Y < 2^k - 1$, zastosowany sumator modulo $2^k - 1$ nie może generować podwójnej reprezentacji zera. W przypadku użycia sumatora z przeniesieniem okrężnym niezbędny jest więc dodatkowy komparator wykrywający wynik postaci $\underbrace{11\ldots1}_{k}{}_2$.

Na potrzeby implementacji warto dodatkowo przekształcić wyrażenie (2.52) opisujące wartość Y do postaci

$$Y = \left|2^{k-1}(X_1 - X_3)\right|_{2^k-1} = \left|2^{k-1} \cdot \left|X_1 - |X_3|_{2^n-1}\right|_{2^k-1}\right|_{2^k-1}. \quad (2.71)$$

Ponieważ $0 \geq X_3 \geq 2^k$, wartość $|X_3|_{2^k-1}$ można wyznaczyć zastępując najmłodszy bit wektora reprezentującego X_3 sumą logiczną tego bitu i bitu najstarszego. Następnie różnicę $X_1 - |X_3|_{2^k-1}$ można zastąpić sumą X_1 i odwrotności addytywnej $|X_3|_{2^k-1}$ modulo $2^k - 1$, czyli negacji $|X_3|_{2^k-1}$. Po wyznaczeniu wartości $\left|X_1 - |X_3|_{2^k-1}\right|_{2^k-1}$, wartość Y jest określona jako rotacja cykliczna otrzymanego wyniku w lewo o $k-1$ bitów. Należy jedynie pamiętać, aby w wyniku $\left|X_1 - |X_3|_{2^k-1}\right|_{2^k-1}$ wyeliminować podwójne zero.



Rysunek 2.14. Schemat układu detekcji znaku dla RNS $(2^k - 1, 2^k, 2^k + 1)$.

Na rys. 2.14 przedstawiono schemat układu implementującego operację opisaną równaniami (2.70) i (2.71). Struktura układu zawiera elementy stosowane w szybkich konwerterach odwrotnych dla RNS$(2^k - 1, 2^k, 2^k + 1)$. Na układ składają się dwa pracujące równolegle sumatory obliczające różnice $X_3 - X_2$ i wartość Y według równania (2.71) oraz wyjściowy sumator obliczający wartość T. Wszystkie sumatory operują na wektorach k–bitowych. Sumator obliczający Y jest pełnym sumatorem modulo $2^k - 1$ z korekcją podwójnej reprezentacji zera.

Struktura układu z rys. 2.14 może być w prosty sposób zaimplementowana w matrycach FPGA. Jako sumatory k–bitowe należy użyć sumatorów RCA wykorzystujących wbudowane w FPGA kanały szybkiej propagacji przeniesień. Sumator modulo $2^k - 1$ może zostać zrealizowany jako kaskadowe połączenie dwóch sumatorów k–bitowych i dodatkowy komparator wykrywający podwójną reprezentację zera. Pierwszy sumator dodaje wektory reprezentujące X_1 oraz odwrotność addytywną $|X_3|_{2^k-1}$, zadaniem drugiego jest dodanie do wektora wyjściowego pierwszego sumatora sumy logicznej przeniesienia z pierwszego sumatora i sygnału z komparatora.

Obszar zajęty przez układ z rys. 2.14 jest sumą obszarów czterech sumatorów k–bitowych i komparatora wykrywającego kombinację $\underbrace{11\ldots11}_{k}{}_2$. Dla układów Spartan 2 implementacja czterech su-

matorów wymaga $4k$ tablic LUT. Komparator może zostać zaimplementowany jako kaskadowe połączenie $k/3$ tablic LUT lub w postaci drzewa o głębokości $\lceil \log_4(k) \rceil$ poziomów. Kompilator opisu w języku VHDL może dodatkowo zoptymalizować strukturę komparatora korzystając z dedykowanej logiki obecnej w FPGA. Dla wartości $k \approx 30$ rozmiar komparatora nie przekracza 10 tablic LUT. Długość ścieżki krytycznej układu z rys. 2.14 jest określona przez sumę opóźnień wprowadzanych przez 3 sumatory k–bitowe i komparator.

Dotychczas opracowane układy detekcji znaku dla RNS wykorzystują pamięci ROM adresowane słowem o szerokości modułu, tak więc zajmują wielokrotnie większy obszar od prezentowanego układu. Z tego powodu układ z rys. 2.14 zostanie porównany z konwerterami odwrotnymi dla RNS $(2^k - 1, 2^k, 2^k + 1)$. Znak po konwersji może zostać zbadany za pomocą dodatkowego komparatora lub zgodnie z równaniem (2.47).

Obszar zajęty przez najlepszy obecnie konwerter odwrotny zaprezentowany w pracy [WSAS02] jest sumą obszarów sumatora $2k$–bitowego i sumatora modulo $2^{2k} - 1$. Stosując jednakową postać sumatora modulo w obydwu porównywanych układach obszar wymagany przez konwerter wynosi $6k$ tablic LUT plus obszar komparatora $2k$–bitowego. Opóźnienie wprowadzane przez konwerter odwrotny jest sumą opóźnień dwóch sumatorów $2k$–bitowych i komparatora. Zaprezentowany układ detekcji znaku jest więc mniejszy zapewniając dodatkowo symetrię zakresu reprezentowanych liczb.

2.3 Hierarchiczne resztowe systemy liczbowe

Zyski wynikające z zastosowania RNS w układach arytmetycznych są szczególnie widoczne w przypadku RNS o bazie złożonej z niewielkich modułów. Niestety, zmniejszenie wartości modułów powoduje wzrost liczności bazy dla RNS o tym samym zakresie dynamicznym. Duża liczba modułów w bazie RNS komplikuje algorytmy konwersji i przeprowadzania operacji trudnych, może także być przyczyną znacznych różnic w ich szerokości, co jest zjawiskiem niekorzystnym. Konieczny jest zatem kompromis pomiędzy szybkością wykonywania operacji arytmetycznych a złożonością konwersji i operacji trudnych.

Pomysłem pozwalającym na zachowanie prostej struktury konwerterów przy niewielkiej wartości modułów w bazie jest zastosowanie hierarchicznych resztowych systemów liczbowych (*ang.* ***H****ierarchical* ***R****esidue* ***N****umber* ***S****ystem*, *HRNS*). Hierarchiczne resztowe systemy liczbowe są specyficzną klasą RNS. Jeśli implementacja przeprowadzana jest w systemach cyfrowych wykorzystujących lo-

gikę dwuwartościową, w klasycznych RNS wartości cyfr są z reguły zapisywane za pomocą systemu naturalnego binarnego (NB). W HRNS wartości wszystkich, lub tylko niektórych, cyfr są zapisywane za pomocą RNS niższego poziomu o odpowiednio dobranym zakresie dynamicznym. Proces ten może być kontynuowany rekurencyjnie na kolejnych poziomach. System binarny jest stosowany dopiero dla RNS na najniższym poziomie.

Rozróżnia się dwa podejścia do problemu wyboru bazy systemów niższego poziomu. Pierwsze, zaproponowane w pracach [AJ68], [Yas92] wymaga, aby zakres systemu niższego poziomu był wystarczający do zapisania wyników pośrednich, np.: dla pojedynczego mnożenia zakres ten musi być większy od kwadratu wartości modułu w nim zapisanego. Prowadzi to do szybkiego wzrostu zakresów systemów niższych poziomów oraz implikuje konieczność użycia wielopoziomowych, rozbudowanych konwerterów pomiędzy tymi systemami.

Cechą szczególną tego rozwiązania jest możliwość wielokrotnego wykorzystania tych samych modułów w różnych RNS niższego poziomu. Niech RNS_1 pierwszego poziomu zawiera moduły $(17, 19, 20, 21)$, a wykonywaną operacją będzie pojedyncze mnożenie. Maksymalna wartość reszt dla największego modułu wynosi 20. Jeśli wykonywaną operacją jest pojedyncze mnożenie, wartość iloczynu dwóch reszt modulo 21 nie przekroczy 20^2. Zakresy RNS niższego poziomu dla poszczególnych cyfr muszą więc wynosić co najmniej $16^2 + 1, 18^2 + 1, 19^2 + 1$ i $20^2 + 1$, czyli 257, 325, 362 i 401. Dla RNS drugiego poziomu o takich zakresach przekroczenie zakresu nie wystąpi dla pojedynczego mnożenia. Można więc we wszystkich przypadkach użyć podobnego zbioru modułów, czyli np: $\text{RNS}_2 = (3, 4, 5, 7)$ o zakresie 420. Czterokrotne zastosowanie RNS_2 dla poszczególnych reszt modulo 17, 19, 20 i 21 pozwala na wykonywanie operacji arytmetycznych w systemie o zakresie $17 \cdot 19 \cdot 20 \cdot 21 = 135660$, czyli ponad 2^{17}, przy użyciu modułów o szerokości nie przekraczającej 3 bitów. Niestety, podstawową wadą tego rozwiązania jest konieczność wykonania konwersji pomiędzy RNS_1 a RNS_2 przed i po pojedynczej operacji arytmetycznej

Druga metoda konstruowania systemów hierarchicznych [SA99] sprowadza się do wyboru bazy systemu podstawowego spośród liczb rozkładalnych na odpowiednio małe czynniki, z których tworzy się następnie bazę systemu niższego poziomu. Ważną jej cechą jest niewielki wzrost zakresu systemów na kolejnych poziomach. Nie ma także potrzeby częstego przeprowadzania konwersji pomiędzy kolejnymi poziomami, ponieważ zakres RNS niższego poziomu jest równy odpowiedniemu modułowi wyższego poziomu. Niestety, zalety te są okupione trudnością znalezienia bazy systemu. W pracy [SA99] rozwiązano ten problem kodując moduły postaci $2^{2k} - 1$ za pomocą systemu o bazie

$(2^k - 1, 2^k + 1)$. Pozwala to na zachowanie prostoty zarówno kanałów obliczeniowych, jak i konwerterów pomiędzy poziomami. Podejście takie ma jednak dwie wady: wartości niektórych modułów są zbliżone do zakresu dynamicznego całego systemu, a pomiędzy wartościami poszczególnych modułów występują duże różnice. Zjawiska te wynikają z warunku wzajemnej pierwszości modułów i są szczególnie niewygodne dla systemów o dużym zakresie dynamicznym.

Poniżej zostanie zaprezentowana nowa propozycja nowej klasy HRNS [Tom05b]. W klasie tej bazą RNS najwyższego poziomu jest zbiór modułów $(2^k - 1, 2^k, 2^k + 1)$, w którym moduły $2^k \pm 1$ są rozkładalne na małe czynniki. Zasadniczą zaletą proponowanych HRNS jest uproszczenie struktur konwerterów z/na system pozycyjny przy zachowaniu niewielkiej szerokości modułu. Poza tym możliwe jest efektywne przeprowadzanie niektórych operacji trudnych w RNS dzięki możliwości wykorzystania istniejących rozwiązań dla RNS$(2^k-1, 2^k, 2^k+1)$. Przykładem może być zaprezentowana w rozdz. 2.2.4 nowa metoda szybkiej detekcji znaku. Ze względu na niski koszt konwersji zaproponowane HRNS mogą być szczególnie użyteczne dla krótkich kanałów obliczeniowych operujących na liczbach o szerokości kilkudziesięciu bitów.

Opisywane HRNS są środkiem pozwalającym na zwiększenie wydajności jednostki mnożenia akumulacyjnego wykorzystywanej w dalszej części pracy. Z tego względu w poniższym opisie główny nacisk położony jest na podzbiór HRNS wybrany pod kątem wspomnianego zastosowania.

2.3.1 Wprowadzenie

W proponowanym systemie zakresy RNS niższego poziomu są równe odpowiednim modułom wyższego poziomu. Cechą charakterystyczną tego HRNS jest użycie liczb postaci $2^k \pm 1$ rozkładalnych na niewielkie czynniki jako modułów RNS wyższego poziomu. Zbiór liczb mogących tworzyć bazę proponowanych HRNS jest ograniczony ze względu na warunek rozkładalności modułów $2^k \pm 1$ na niewielkie czynniki.

Liczby postaci $2^k - 1$ nie są liczbami pierwszymi, jeśli k nie jest liczbą pierwszą, ponieważ

$$2^{ij} - 1 = (2^i - 1)(2^{0 \cdot i} + 2^{1 \cdot i} + ^{2 \cdot i} + \ldots + 2^{(j-1) \cdot i}). \tag{2.72}$$

Istnieją także liczby $2^k - 1$ dla k będącego liczbą pierwszą, które nie są liczbami pierwszymi, np. $2^{11} - 1 = 23 \cdot 89$. Liczby postaci $2^k + 1$ są rozkładalne, gdy k nie jest liczbą pierwszą ani potęgą dwójki, ponieważ dla j nieparzystego

$$2^{ij} + 1 = (2^i + 1)(2^{0 \cdot i} - 2^{1 \cdot i} + 2^{2 \cdot i} - \ldots + 2^{(j-1) \cdot i}). \tag{2.73}$$

Tabela 2.1. Pary modułów postaci $2^k \pm 1$ rozkładalnych na małe czynniki.

moduł	czynniki	maks. szer	moduł	czynniki	maks. szer
2^6-1	7 9	4	$2^{14}+1$	5 29 113	7
2^6+1	5 13	4	$2^{15}-1$	7 31 151	8
2^9-1	7 73	7	$2^{15}+1$	9 11 331	9
2^9+1	19 27	5	$2^{18}-1$	7 19 27 73	7
$2^{10}-1$	3 11 31	5	$2^{18}+1$	5 13 37 109	7
$2^{10}+1$	25 41	6	$2^{24}-1$	5 9 7 13 17 241	8
$2^{12}-1$	5 7 9 13	4	$2^{24}+1$	97 257 673	10
$2^{12}+1$	17 241	8	$2^{30}-1$	7 9 11 31 151 331	9
$2^{14}-1$	3 43 127	7	$2^{30}+1$	13 25 41 61 1321	11

Sam warunek rozkładalności liczb $2^k \pm 1$ nie gwarantuje jeszcze utworzenia wydajnego HRNS, ponieważ dla niektórych czynników obszar i opóźnienie układów arytmetycznych mogą być zbyt duże. Przykładem może być liczba $2^{26} - 1 = 3 \cdot 22369621$. Jednostka mnożąca modulo 22369621 jest bardziej skomplikowana od jednostki mnożącej modulo $2^{26} - 1$.

Przykładowe wartości modułów postaci $2^k \pm 1$ dla $k \leq 102$ przydatnych do konstrukcji baz HRNS przedstawiono w tabelach 2.1 i 2.2. Tabela 2.2 nie zawiera iloczynów odpowiednich par modułów z tabeli 2.1. Analizując kolejne wiersze tabeli 2.2 można zauważyć, że systemy te stają się szczególnie atrakcyjne dla dużych zakresów dynamicznych. Przykładem mogą być moduły $2^{60} - 1$, $2^{84} - 1$ czy $2^{102} - 1$, dla których można maksymalne szerokości czynników są znacznie mniejsze od k.

Korzystając z danych w tabelach 2.1 i 2.2 można budować hierarchiczne systemy resztowe na wiele różnych sposobów. Konstrukcja systemu analizowanego w rozprawie polega na wytypowaniu z tab. 2.1 pary modułów $2^k \pm 1$ i stworzeniu RNS pierwszego poziomu postaci $\left(2^k - 1, 2^k, 2^k + 1\right)$. Rozwiązanie to pozwala na użycie szybkich konwerterów odwrotnych dla RNS $\left(2^k - 1, 2^k, 2^k + 1\right)$. Korzystną cechą jest także zastosowanie modułu postaci 2^k, dla którego implementacja operacji arytmetycznych jest prosta i wydajna. Dodatkową zaletą prezentowanego HRNS jest możliwość skalowania i detekcji znaku bez konieczności wykonywania pełnej konwersji odwrotnej. Tabela 2.1 zawiera wszystkie użyteczne bazy dla opisywanych HRNS o zakresie dynamicznym od ok. 18 do 90 bitów. Dla pozostałych liczb $2^k \pm 1$ z podanego zakresu duże wartości czynników utrudniają konstrukcję wydajnych jednostek arytmetycznych.

Tabela 2.2. Moduły postaci $2^k - 1$ rozkładalne na małe czynniki.

moduł	czynniki	maks. szer.
$2^{32} - 1$	3 5 17 257 65537	17
$2^{42} - 1$	9 43 49 127 337 5419	13
$2^{44} - 1$	3 5 23 89 397 683 2113	12
$2^{50} - 1$	3 11 31 251 601 1801 4051	12
$2^{52} - 1$	3 5 53 157 1613 2731 8191	13
$2^{72} - 1$	5 7 13 17 19 27 37 73 109 241 433 38737	16
$2^{84} - 1$	5 9 13 29 43 49 113 127 337 1429 5419 14449	15
$2^{100} - 1$	3 11 31 41 101 125 251 601 1801 8101 4051 268501	19
$2^{102} - 1$	7 9 103 307 2143 2857 6529 11119 43691 131071	17

Proponowany w rozprawie HRNS jest systemem dwupoziomowym. Reszty dla modułów pierwszego poziomu równych $2^k \pm 1$ są zapisywane w RNS o bazach określonych przez czynniki tych modułów. Zakresy RNS drugiego poziomu są identyczne z wartością odpowiednich modułów na pierwszym poziomie. Wykonywanie operacji arytmetycznych w RNS drugiego poziomu jest więc równoważne wykonywaniu tych operacji modulo odpowiedni moduł RNS pierwszego poziomu. Z tego względu opisywany HRNS można także traktować jak zwykły RNS, którego bazę stanowią czynniki modułów $2^k \pm 1$ i moduł 2^k. Cechą szczególną tego RNS jest możliwość konstrukcji konwerterów w postaci hierarchicznych struktur o niskiej złożoności. Konwersja przeprowadzana jest dwuetapowo: najpierw do pośredniego RNS $(2^k - 1, 2^k, 2^k + 1)$, a następnie do systemu docelowego.

Dotychczas opracowane struktury konwerterów dla RNS $(2^k - 1, 2^k, 2^k + 1)$ mają znacznie mniejszą złożoność od konwerterów dla pozostałych RNS o porównywalnym zakresie dynamicznym. Niestety, niewielka liczba modułów o dużej wartości zwiększa koszt wykonywania operacji arytmetycznych w RNS $(2^k - 1, 2^k, 2^k + 1)$. Proponowane HRNS łączą możliwość zastosowania wydajnych konwerterów dla RNS $(2^k - 1, 2^k, 2^k + 1)$ z niską złożonością wykonywania operacji arytmetycznych dzięki użyciu modułów o małej wartości.

2.3.2 Konwersja pomiędzy HRNS a systemem pozycyjnym

Poniżej zostaną zaprezentowane podstawy teoretyczne struktur konwerterów pomiędzy HRNS i systemem pozycyjnym. Konwersja przeprowadzana jest w dwóch krokach. W pierwszym etapie liczby

są konwertowane do RNS o bazie $(2^k - 1, 2^k, 2^k + 1)$, drugi krok obejmuje konwersję do systemu wyjściowego. Dzięki takiemu rozwiązaniu operacje na liczbach bliskich zakresowi systemu są przeprowadzane wyłącznie za pomocą niewielkich i wydajnych konwerterów dla RNS $(2^k-1, 2^k, 2^k+1)$. Konwersja z HRNS do RNS $(2^k - 1, 2^k, 2^k + 1)$ jest oparta na koncepcji zaprezentowanej w pracy [Wan00]. Konwersja ta przeprowadzana jest dla RNS o zakresie znacznie mniejszym od zakresu pełnego HRNS, dzięki czemu uzyskuje się zmniejszenie obszaru i długości ścieżki krytycznej konwerterów. Na potrzeby rozprawy równania opublikowane w [Wan00] zostały przekształcone do postaci umożliwiającej zminimalizowanie obszaru zajmowanego przez konwerter. Przekształcenia te umożliwiają obniżenie kosztu pojedynczego konwertera do poziomu pozwalającego na użycie proponowanych HRNS nawet dla bardzo krótkich kanałów obliczeniowych.

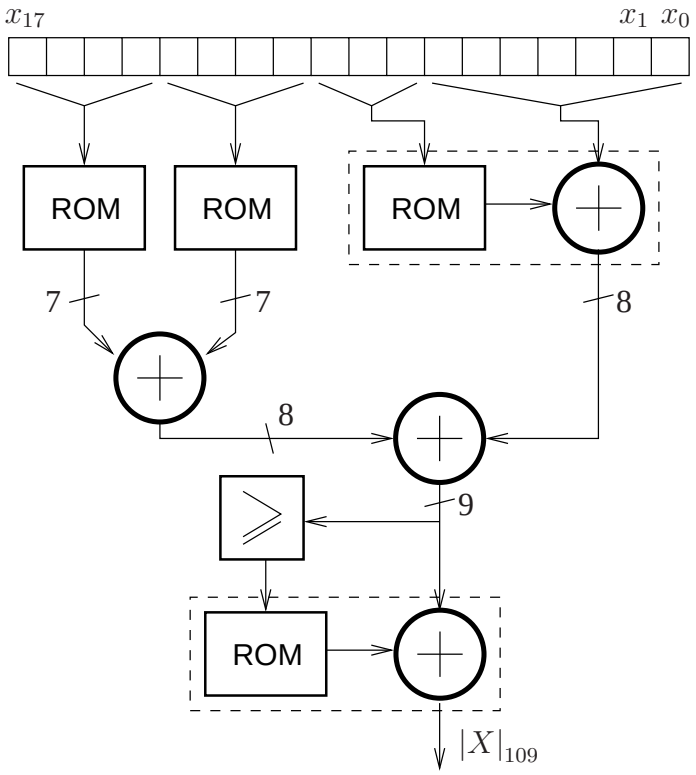
Konwersja z systemu pozycyjnego do HRNS

Konwersja liczby całkowitej do systemu resztowego wymaga wyznaczenia dla tej liczby reszt modulo kolejne składniki bazy systemu resztowego. W prezentowanym konwerterze konwersja z systemu pozycyjnego do RNS $(2^k-1, 2^k, 2^k+1)$ nie jest wymagana, ponieważ założono, że szerokość operandów wejściowych jest ograniczona do k bitów. Dla szerszych operandów można zastosować sumator dodający k–bitowe pola wydzielone z wektora wejściowego [Pie94].

Po konwersji do RNS $(2^k - 1, 2^k, 2^k + 1)$ należy dla słów k i $(k + 1)$–bitowego obliczyć reszty modulo czynniki liczb $2^k \pm 1$. Struktura generatora reszt zależy od wartości okresu bądź półokresu potęg 2 modulo rozpatrywany czynnik. Dla czynników o dużej wartości okresu/półokresu stosowany jest układ, w którym dla wydzielonych pól ze słowa wejściowego obliczane są reszty za pomocą pamięci ROM. Reszty te są następnie sumowane w kaskadzie sumatorów szeregowych (sumator z propagacją przeniesień, *ang. ripple–carry adder, RCA*). Reszta dla wyniku sumowania jest obliczana przez odjęcie krotności modułu zależnej od wartości sumy. Dla małych szerokości modułów (ok. 4–5 bitów) operację tę można przeprowadzić za pomocą pojedynczej pamięci ROM. W pozostałych przypadkach badanie zakresu wyniku jest realizowane za pomocą zestawu komparatorów, a pamięć ROM jest wykorzystywana jedynie do przechowywania odejmowanych stałych. Przykładem może być pokazany na rys. 2.15 generator modulo 109 dla słowa 18–bitowego.

Charakterystyki AT generatora modulo można zmniejszyć, jeśli wartość okresu/półokresu potęg 2 modulo dany czynnik jest odpowiednio mała. Możliwa jest wtedy wstępna redukcja szerokości słowa wejściowego generatora. Redukcja szerokości jest przeprowadzana według koncepcji wykorzy-

stującej małe twierdzenie Fermata (str. 43). Generator dla pojedynczego czynnika poprzedzony jest wielooperandowym sumatorem z przeniesieniem okrężnym (*ang. end–around carry, EAC*) [Pie94]. Szerokość sumatora równa jest okresowi bądź półokresowi potęg 2 modulo dany czynnik.



Rysunek 2.15. Przykład układu wyznaczania reszty słowa 18–bitowego modulo 109.

Obszar sumatora z EAC można dodatkowo zredukować. Niech $P(M_i)$ oznacza wartość okresu potęg 2 modulo M_i, a $HP(M_i)$ wartość półokresu potęg 2 modulo M_i. Wartości okresów lub półokresów potęg 2 modulo wybrane czynniki liczb $2^k \pm 1$ są często równe wielokrotnościom okresów lub półokresów dla innych czynników. Możliwe jest więc skonstruowanie fragmentów sumatora EAC jako wspólnych dla kilku czynników. W tab. 2.3 zamieszczono wartości okresów/półokresów potęg 2 modulo czynniki wybranych liczb $2^k \pm 1$.

Idea wykorzystania części sumatora jako wspólnej dla kilku czynników zostanie pokazana na przykładzie fragmentu układu wyznaczającego resztę liczby 18–bitowej modulo czynniki liczb $2^{18} \pm 1$. Jeżeli dla modułów M_1, M_2 zachodzi zależność $HP(M_1) = HP(M_2)$ lub $P(M_1) = i \cdot P(M_2)$ dla $i = 1, 2, 3, \ldots$, to sumator redukujący szerokość operandu do $P(M_1)$ lub $HP(M_1)$ bitów może być wspólny dla modułów M_1, M_2. W opisywanym przykładzie sytuacja taka zachodzi dla modułów

Tabela 2.3. Wartości okresów i półokresów potęg 2 modulo czynniki liczb $2^k \pm 1$

$k = 18$			$k = 24$			$k = 30$		
M_i	$HP(M_i)$	$P(M_i)$	M_i	$HP(M_i)$	$P(M_i)$	M_i	$HP(M_i)$	$P(M_i)$
7		3	5	2	4	7		3
19	9	18	7		3	9	3	6
27	9	18	9	3	6	11	5	10
73		9	13	6	12	31		5
5	2	4	17	4	8	151		15
13	6	12	241	12	24	331	15	30
37	18	36	97	24	48	13	6	12
109	18	36	257	8	16	25	10	20
			673	24	48	41	10	20
						61	30	60
						1321	30	60

5,7,13. Ponieważ $P(13) = 4 \cdot P(7) = 3 \cdot P(5)$, sumator z EAC dla tych modułów może zawierać część wspólną składającą się z sumatora redukującego szerokość operandu wejściowego do 12 bitów. W związku z tym szerokość słowa wejściowego generatorów modulo można zmniejszyć z 18 do 12 bitów. Podobna sytuacja zachodzi także dla modułów 19 i 27, gdzie elementem wspólnym jest sumator z przeniesieniem okrężnym redukujący szerokość operandu wejściowego z wykorzystaniem $HP(19) = HP(27) = 9$. Właściwość ta umożliwia znaczne zmniejszenie obszaru wymaganego przez generator reszt, szczególnie dla szerokich operandów.

Wykorzystanie części sumatora dla kilku modułów równocześnie jest możliwe także w sytuacji, gdy $HP(M_1) = P(M_2)$. Należy wtedy podzielić operand wejściowy na pola o szerokości $P(M_2)$ bitów i użyć trzech sumatorów wielooperandowych. Dwa z nich posłużą do obliczenia sum pól o indeksach odpowiednio parzystych i nieparzystych. Trzeci sumator zostanie użyty do zsumowania zanegowanych pól o indeksach nieparzystych. Następnie, dodając sumę pól o indeksach parzystych do sumy zanegowanych bądź nie pól o indeksach nieparzystych, uzyskuje się redukcję operandu modulo odpowiednio $(2^{HP(M_1)} + 1)$ i $(2^{P(M_2)} - 1)$. Pozwala to zmniejszyć obszar o ok. 25% w stosunku do układu złożonego z dwóch niezależnych sumatorów o szerokości $P(M_2)$ i $HP(M_1)$.

Konwersja dla liczb ze znakiem

Przedstawione konwertery są łatwo modyfikowalne do postaci pozwalającej na wyznaczanie dodatnich reszt dla liczb ujemnych zapisanych w kodzie U2. Modyfikacja może dotyczyć zarówno sumatora z przeniesieniem okrężnym, jak i generatora modulo.

Modyfikacja generatora modulo nie wymaga zmian w strukturze układu. W proponowanym rozwiązaniu reszty modulo czynnik dla poszczególnych pól bitowych są obliczane za pomocą pamięci ROM. Wystarczy więc dla pamięci dekodującej pole zawierające najstarszy bit słowa wejściowego zwracać reszty dla liczb ujemnych zgodnie z wzorem

$$R = \left| -1 \cdot 2^{k-1} \cdot x_{k-1} + \sum_{j=l}^{k-2} 2^j \cdot x_j \right|_{M_i}, \tag{2.74}$$

gdzie x_i oznacza kolejne bity pola, a $R \geq 0$ określa zawartość komórki adresowanej wektorem bitowym $x_{k-1}x_{k-2}\ldots x_l$. Zawartość pamięci używanej na rys. 2.15 do dekodowania najstarszego pola pokazano w tab. 2.4.

Tabela 2.4. Zawartość pamięci dekodującej pole zawierające bit o wadze -2^{17}

Bity adresu	Zawartość komórki
0000	0
0001	$\left\|2^{14}\right\|_{109} = 34$
...	...
1000	$\left\|-2^{17}\right\|_{109} = 55$
1001	$\left\|-2^{17} + 2^{14}\right\|_{109} = 89$
...	...

Uwzględnienie liczb ujemnych w sumatorach z EAC wymaga niewielkich zmian w strukturze połączeń. W sumatorze dodającym pola o szerokości okresu wykorzystywany jest fakt, że kolejne $P(M_i)$–bitowe pola operandu zawierają bity o tych samych wagach modulo M_i. Operując na liczbach w kodzie U2, jedyna różnica dotyczy bitu na najstarszej pozycji, którego waga jest traktowana jako wartość ujemna. Należy zatem dla tego bitu wyznaczyć dodatnią resztę modulo M_i i zsumować z resztami modulo M_i dla pozostałych pól operandu wejściowego.

Jedną z metod jest potraktowanie bitu znaku jako dodatkowego sygnału przekazywanego do generatora modulo. Dla takiego rozwiązania zmiany w strukturze sumatora z EAC nie są wymagane. W przypadku stosowania pamięci ROM w generatorze reszt można bit o wadze ujemnej doprowadzić do

dodatkowej linii adresowej dowolnej pamięci, a wartości zapisane w pamięci poszerzyć o dodatkowe wyniki. Jest to koncepcja prosta i łatwa w implementacji, niestety wiąże się ze wzrostem rozmiaru pamięci ROM.

Drugą metodą pozwalającą na skonstruowanie układu dla liczb w kodzie U2 jest wprowadzenie modyfikacji do sumatora wielooperandowego. Pierwszym krokiem powinno być lewostronne rozszerzenie znakowane słowa wejściowego do szerokości, dla której bit o wadze ujemnej będzie na pozycji o jeden większej od wielokrotności okresu/półokresu. Następnie należy zauważyć, że sumatory wielooperandowe dodające pola o szerokości $P(M_i)$ i $HP(M_i)$ są sumatorami modulo $2^{P(M_i)} - 1$ i $2^{HP(M_i)} + 1$. Można więc wykorzystać jedną z trzech zależności

$$\left|-1 \cdot 2^{j \cdot k}\right|_{2^k-1} = -1, \tag{2.75}$$

$$\left|-1 \cdot 2^{2 \cdot j \cdot k}\right|_{2^k+1} = -1, \tag{2.76}$$

$$\left|-1 \cdot 2^{2 \cdot j \cdot k+1}\right|_{2^k+1} = 1, \tag{2.77}$$

które są prawdziwe dla k i j naturalnych.

Modyfikacja sumatora wielooperandowego dodającego pola o szerokości okresu potęg 2 modulo M_i jest przeprowadzana na podstawie (2.75). Uwzględnienie bitu znaku wymaga więc odjęcia wartości tego bitu. Można to zrealizować przez dodanie odwrotności addytywnej 1 modulo $2^{P(M_i)} - 1$. Niestety, wymaga to użycia dodatkowego sumatora dodającego słowo, w którym wszystkie bity są równe bitowi znaku.

Korzystniejszym rozwiązaniem może być implementacja według wzoru

$$-x^i_{j \cdot P(M_i)} = -1 + \overline{x^i}_{j \cdot P(M_i)}, \tag{2.78}$$

gdzie $\overline{x^i}_{j \cdot P(M_i)}$ oznacza negację bitu na pozycji $j \cdot P(M_i)$. Wzór (2.78) oznacza, że zamiast odjęcia wartości bitu $x^i_{j \cdot P(M_i)}$ odejmowana jest zawsze 1, a dodawany jest bit zanegowany. Ponieważ dodawany jest tylko jeden bit o wadze 1, można go doprowadzić do przeniesienia wejściowego dowolnego sumatora. Z reguły wybierany jest sumator na pierwszym poziomie kaskady RCA, ponieważ przeniesienia wejściowe pozostałych sumatorów są wykorzystane przez przeniesienia okrężne. Stała -1 może być dodana w dowolnym miejscu całego generatora resztowego, można ją więc wkomponować w jedną z pamięci ROM stosowanych w dalszej części generatora modulo.

Dla sumatorów dodających pola o szerokości półokresu potęg 2 możliwe są dwa przypadki. Jeśli bit znaku znajduje się na pozycji bezpośrednio następującej po *nieparzystej* wielokrotności półokresu,

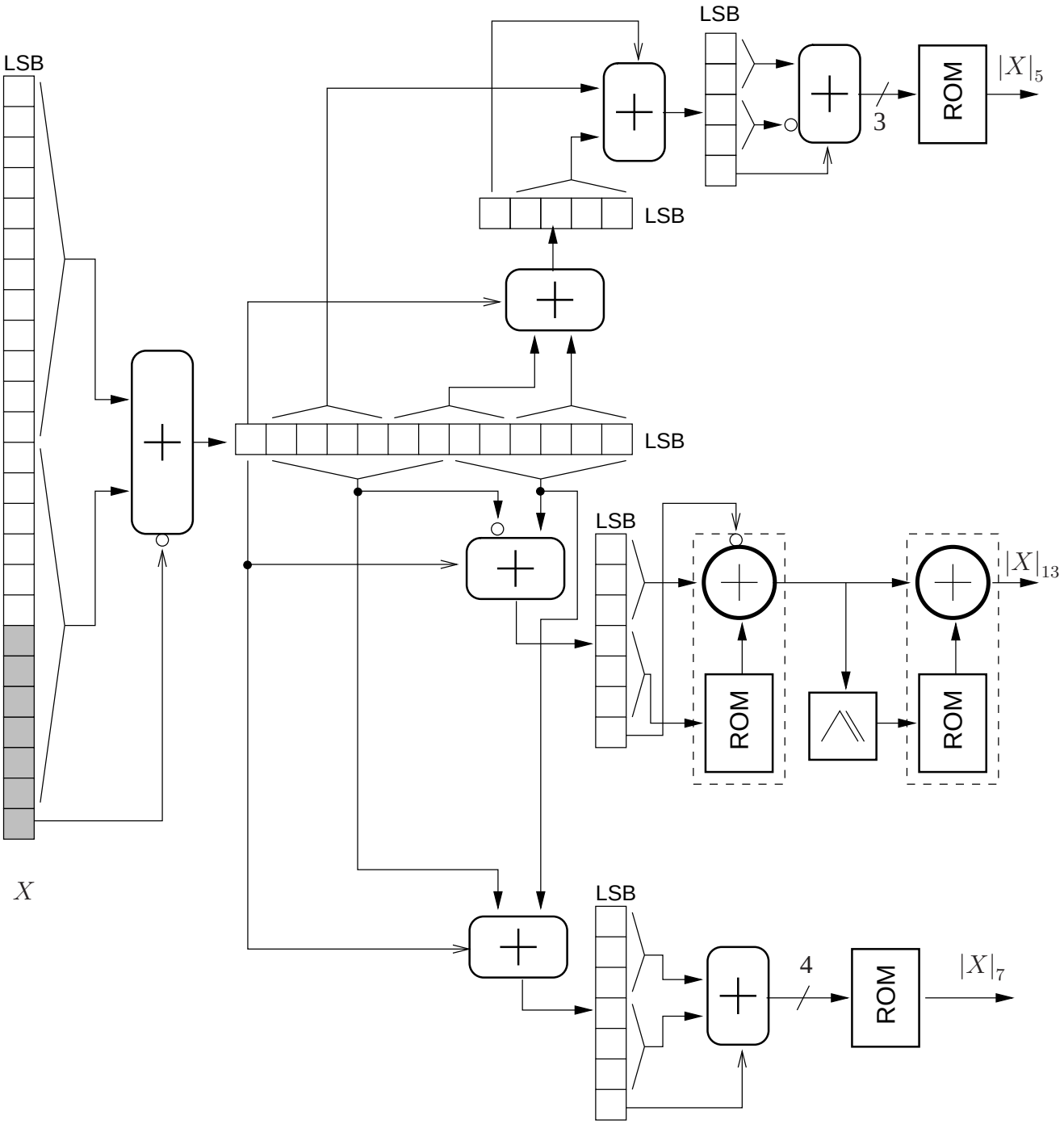
wykorzystywany jest wzór (2.77). Wystarczy więc doprowadzić bit znaku do przeniesienia wejściowego dowolnego sumatora kaskady RCA. Jeśli bit znaku jest na pozycji bezpośrednio następującej po *parzystej* wielokrotności półokresu, modyfikacja sumatora wielooperandowego jest określona przez wzór (2.76). Uwzględnienie znaku liczby wymaga więc odjęcia bitu znaku. Realizowane jest to podobnie, jak dla sumatorów dodających pola o szerokości okresu potęg 2 modulo M_i.

Struktura układu

Struktura kompletnego konwertera z systemu pozycyjnego do HRNS zależy od wzajemnych relacji pomiędzy czynnikami modułów $2^k \pm 1$. Podanie ogólnych wytycznych gwarantujących uzyskanie optymalnego układu jest niezmiernie trudne. Podczas konstrukcji konkretnego konwertera należy sprawdzić kilka możliwych rozwiązań i wybrać to o najbardziej pożądanych parametrach. Różnice pomiędzy rozwiązaniami będą dotyczyły przede wszystkim wyboru czynników, dla których budowany jest wspólny sumator dodający pola o szerokości okresów lub półokresów tych czynników. Przykład układu wyznaczania reszty modulo 5,7 i 13 dla liczby całkowitej X reprezentowanej 18–bitowym słowem w kodzie U2 przedstawiono na rys. 2.16.

Pierwszym elementem układu jest 12–bitowy sumator przeprowadzający wstępną redukcję modulo $2^{12} - 1$. Słowo wejściowe jest rozszerzone do szerokości 24 bitów, a zanegowany bit znaku doprowadzony do przeniesienia wejściowego sumatora. Wyjściem sumatora jest już liczba dodatnia, przy czym w dalszych stopniach układu należy zastosować korekcję -1 związaną z dodawaniem zanegowanego bitu znaku zgodnie z wzorem (2.78). Wyjście 12–bitowego sumatora jest przekazywane do zestawu układów obliczających reszty modulo 5,7 i 13. Każdy z tych układów zawiera sumator dodający pola o szerokości odpowiedniego okresu bądź półokresu potęg 2 oraz pamięci ROM w stopniu wyjściowym. Każda z pamięci przekodowuje słowo wejściowe na odpowiednie reszty skorygowane o sumę stałych wynikających z konstrukcji sumatorów z EAC i -1. Wzory opisujące wartość korekcji dla zadanej struktury sumatora EAC przedstawiono w [Pie94].

Układ z rys. 2.16 można zbudować także w inny sposób. Słowo wejściowe może być podzielone na trzy 6–bitowe pola, z których zostaną obliczone reszty modulo $2^6 \pm 1$. Dla tych reszt można następnie obliczyć wartości $|X|_7$ i $|X|_{13}$. W tym przypadku sumator 12–bitowy można zastąpić sumatorem 6–bitowym. Niestety, układ redukcji modulo 5 byłby całkowicie niezależny, a szerokość jego słowa wejściowego wzrosła by do 18 bitów. Konieczne więc byłoby użycie dwóch dodatkowych sumatorów 4–bitowych w torze redukcji modulo 5. Układ z rys. 2.16 wymaga więc mniejszej liczby ogniw FA.



Rysunek 2.16. Przykład układu wyznaczania reszty modulo 5,7 i 13 dla słowa 18–bitowego w kodzie U2. Kolorem szarym oznaczono dodane bity rozszerzenia znakowanego.

Konwersja z HRNS do systemu pozycyjnego

Konwersja z HRNS do systemu pozycyjnego przeprowadzana jest dwuetapowo. Pierwszy etap obejmuje konwersję z HRNS do RNS $(2^k-1, 2^k, 2^k+1)$, drugi etap to konwersja z RNS do systemu pozycyjnego. W większości przypadków konwersja pomiędzy HRNS a RNS oparta jest na twierdzeniu opublikowanym w pracy [Wan00] nazwanym przez autorów nowym chińskim twierdzeniem o resztach II. Podstawy teoretyczne konwertera opisano w rozdz. 2.2 na str. 48. Jako konwerter z RNS $(2^k-1, 2^k, 2^k+1)$ do systemu pozycyjnego może zostać użyte dowolne z wielu dotychczas opracowanych rozwiązań, np. układ zaprezentowany w pracy [WSAS02] zawierający jeden sumator modulo $2^{2k}-1$.

Konwersja odwrotna według koncepcji z pracy [Wan00] przeprowadzana jest zgodnie z wzorem (2.34). Dla bazy RNS złożonej z czynników liczb postaci $2^k \pm 1$ możliwy jest dobór modułów w pary, dla których implementacja operacji opisanej przez (2.34) jest szczególnie prosta. Wynika to zarówno z wartości modułów, dla których wykonywane są operacje, jak i z postaci stałych występujących we wzorze (2.34). Poniżej zostaną przedstawione przekształcenia równania konwersji odwrotnej oraz wskazówki dotyczące implementacji dla RNS zdefiniowanych przez czynniki wybranych liczb postaci $2^k \pm 1$. Przedstawione równania były tworzone z myślą o efektywnej implementacji, dlatego też należy je rozpatrywać w powiązaniu ze sposobem ich sprzętowej realizacji.

Konwersja dla $2^{18}-1$

Czynniki liczby $2^{18}-1$ można pogrupować w dwie pary, których iloczyny są liczbami postaci $2^9 \pm 1$. Pary te to $7 \cdot 73 = 511$ i $19 \cdot 27 = 513$. Konwersja odwrotna może być przeprowadzona za pomocą dwupoziomowego układu. Pierwszy poziom jest odpowiedzialny za znalezienie reszt modulo 511 i 513 zgodnie z wzorem

$$\begin{aligned} |X|_{511} &= |X|_{73} + \left| \underbrace{\left|73^{-1}\right|_7}_{5} \cdot (|X|_7 - |X|_{73}) \right|_7 \cdot 73 \\ |X|_{513} &= |X|_{27} + \left| \underbrace{\left|27^{-1}\right|_{19}}_{12} \cdot (|X|_{19} - |X|_{27}) \right|_{19} \cdot 27 \end{aligned} . \tag{2.79}$$

Operacje modulo są ograniczone do małych liczb, dla których odpowiednie układy są względnie proste. Po obliczeniu $|X|_{511}$ i $|X|_{513}$ wartość reszty modulo $2^{18}-1$ jest określona jako

$$|X|_{2^{18}-1} = |X|_{513} + \left| \underbrace{\left|513^{-1}\right|_{511}}_{256} \cdot (|X|_{511} - |X|_{513}) \right|_{511} \cdot 513. \tag{2.80}$$

Wzór (2.80) może zostać zaimplementowany za pomocą sumatora modulo 511 obliczającego różnicę $|X|_{511} - |X|_{513}$ i końcowego sumatora 18–bitowego. Pozostałe operacje sprowadzają się do odpowiednich obrotów i przesunięć bitów. Iloczyn $||X|_{511} - |X|_{513}|_{511} \cdot 256$ modulo 511 opisuje rotację cykliczną w lewo o 8 bitów. Mnożenie liczby z zakresu $[0, 511)$ przez 513 sprowadza się do konkatenacji dwóch wektorów 9–bitowych. Pełny konwerter może zatem zostać zaimplementowany za pomocą kilku sumatorów modulo małe moduły i niewielkich układów mnożących przez stałe.

Konwersja dla $2^{18} + 1$

Czynniki liczby $2^{18} + 1$ można pogrupować w pary $5 \cdot 13 = 65$ i $37 \cdot 109 = 4033$. Jeden iloczyn jest równy 2^6+1, iloczyn drugiej pary wynosi $2^{12}-2^5+2^0$, co umożliwia efektywną implementację układu mnożącego. Podobnie jak w poprzednim przypadku, konwersja odwrotna może być przeprowadzona w dwupoziomowym układzie. Zadaniem pierwszego poziomu jest obliczenie reszt modulo 65 i 4033 zgodnie z wzorem

$$\begin{aligned} |X|_{65} &= |X|_{13} + \left| \underbrace{\left|13^{-1}\right|_5}_{2} \cdot (|X|_5 - |X|_{13}) \right|_5 \cdot 13 \\ |X|_{4033} &= |X|_{109} + \left| \underbrace{\left|109^{-1}\right|_{37}}_{18} \cdot (|X|_{37} - |X|_{109}) \right|_{37} \cdot 109 \end{aligned} . \tag{2.81}$$

Operacje modulo we wzorze (2.81) są ograniczone do niewielkich modułów. Następnie wartość reszty modulo $2^{18} + 1$ można wyznaczyć na podstawie $|X|_{65}$ i $|X|_{4033}$ zgodnie z formułą

$$|X|_{2^{18}+1} = |X|_{4033} + \left| \underbrace{\left|4033^{-1}\right|_{65}}_{22} \cdot (|X|_{65} - |X|_{4033}) \right|_{65} \cdot \underbrace{4033}_{111111000001_2} . \tag{2.82}$$

Występujące we wzorze (2.82) mnożenie przez stałą modulo 65 może zostać zaimplementowane z użyciem sumatora o szerokości półokresu potęg 2 modulo 65. Mnożenie przez stałą 4033 sprowadza się do użycia pojedynczego subtraktora, ponieważ $4033 = 2^{12} - 2^5 + 2^0$, a wynik mnożenia liczby z zakresu $[0, 65)$ przez $2^{12} + 2^0$ można wyznaczyć poprzez złożenie dwóch wektorów. Obszar i opóźnienie kompletnego konwertera są w tym przypadku nieco większe niż dla $2^{18} - 1$ ze względu na konieczność wyznaczenia reszty modulo 65 dla $|X|_{4033}$ i użycia szerokiego subtraktora realizującego mnożenie przez 4033.

Konwersja dla $2^{24} - 1$

Ponieważ liczbę $2^{24} - 1$ można rozłożyć na sześć czynników, konwersja odwrotna według wzoru

(2.34) wymaga układu trójpoziomowego. Pomimo tego opóźnienie wprowadzane przez układ jest niewielkie ze względu na niezwykle proste operacje dla niektórych par czynników. Czynniki liczby $2^{24}-1$ można pogrupować w trzy pary: $5 \cdot 13 = 65$, $7 \cdot 9 = 63$ i $17 \cdot 241 = 4097$. Układ konwersji odwrotnej składa się z kanału dla pary $17 \cdot 241$ oraz pracującego równolegle kanału obliczającego resztę modulo 4095 z reszt dla par $5 \cdot 13$ oraz $7 \cdot 9$. Konwersja dla pary $17 \cdot 241$ przeprowadzana jest zgodnie z formułą

$$|X|_{4097} = |X|_{241} + \left| \underbrace{\left|241^{-1}\right|_{17}}_{6} \cdot (|X|_{17} - |X|_{241}) \right|_{17} \cdot \underbrace{241}_{11110001_2} . \tag{2.83}$$

W implementacji sprzętowej należy użyć generatora reszty modulo 17 dla $|X|_{241}$, wielooperandowego sumatora obliczającego różnicę $|X|_{17} - |X|_{241}$ przemnożoną przez 6 modulo 17 oraz subtraktor realizujący mnożenie przez $241 = 2^8 - 2^3 + 2^0$. Niezbędny jest także sumator dodający do wyniku operacji modulo wartość $|X|_{241}$.

W kanale obliczającym resztę modulo 4095 konieczne jest zastosowanie struktury dwupoziomowej. Pierwszy stopień odpowiedzialny jest za wyznaczenie reszt modulo 63 i 65 zgodnie z wzorem

$$\begin{aligned} |X|_{65} &= |X|_{13} + \left| \underbrace{\left|13^{-1}\right|_{5}}_{2} \cdot (|X|_{5} - |X|_{13}) \right|_{5} \cdot 13 \\ |X|_{63} &= |X|_{9} + \left| \underbrace{\left|9^{-1}\right|_{7}}_{4} \cdot (|X|_{7} - |X|_{9}) \right|_{7} \cdot 9 \end{aligned} . \tag{2.84}$$

Następny krok obejmuje obliczenie reszty modulo 4095 równej

$$|X|_{4095} = |X|_{65} + \left| \underbrace{\left|65^{-1}\right|_{63}}_{32} \cdot (|X|_{63} - |X|_{65}) \right|_{63} \cdot 65. \tag{2.85}$$

Wzór (2.85) można zaimplementować za pomocą jednego sumatora modulo 63 obliczającego różnicę $|X|_{63} - |X|_{65}$ i jednego sumatora 12–bitowego. Pozostałe działania sprowadzają się do odpowiednich przesunięć wektorów bitowych. Ze względu na prostotę operacji opisanych wzorami (2.84) i (2.85), wartość $|X|_{4095}$ może być obliczona równocześnie z wartością $|X|_{4097}$ według (2.83).

Po obliczeniu reszt modulo 4095 i 4097 wartość reszty modulo $2^{24}-1$ dana jest wzorem

$$|X|_{2^{24}-1} = |X|_{4097} + \left| \underbrace{\left|4097^{-1}\right|_{4095}}_{2^{11}} \cdot (|X|_{4095} - |X|_{4097}) \right|_{4095} \cdot 4097. \tag{2.86}$$

Również w tym przypadku implementacja wymaga jedynie sumatora modulo 4095 i końcowego sumatora 24–bitowego. Pozostałe operacje sprowadzają się do odpowiednich przesunięć bitowych.

Kompletny konwerter może zatem zostać bardzo efektywnie zaimplementowany z użyciem kilku układów wykonujących mnożenie modulo małe moduły przez niewielkie stałe oraz kilku sumatorów z przeniesieniem okrężnym.

Konwersja dla $2^{24}+1$

Liczbę $2^{24}+1$ można podzielić na trzy czynniki, tak więc konwersja według wzoru (2.34) wymaga dwóch kroków. W pierwszym na podstawie reszt dla pary $97 \cdot 673 = 65281$ obliczana jest reszta modulo 65281 jako

$$|X|_{65281} = |X|_{673} + \left| \underbrace{\left|673^{-1}\right|_{97}}_{16} \cdot (|X|_{97} - |X|_{673}) \right|_{97} \cdot 673. \tag{2.87}$$

Implementacja wzoru (2.87) wymaga generatora reszty modulo 97 dla $|X|_{673}$, subtraktora modulo 97, kolejnego generatora reszty modulo 97 dla przesuniętej różnicy $|X|_{97} - |X|_{673}$ oraz układu mnożenia przez stałą 673. Po wyznaczeniu $|X|_{65281}$ wartość reszty modulo $2^{24}+1$ dana jest wzorem

$$|X|_{2^{24}+1} = |X|_{65281} + \left| \underbrace{\left|65281^{-1}\right|_{257}}_{86} \cdot (|X|_{257} - |X|_{65281}) \right|_{257} \cdot \underbrace{65281}_{1111111100000001_2}. \tag{2.88}$$

W równaniu (2.88) operacje modulo są wykonywane dla modułu równego 257, dla którego wykorzystuje się okresowość potęg 2 modulo. Mnożenie przez stałą 86 modulo 257 może zostać zaimplementowane za pomocą sumatora wielooperandowego z EAC dodającego odpowiednio przesunięte pola wektorów reprezentujących $|X|_{62581}$ i $|X|_{257}$. Mnożenie przez stałą $65281 = 2^{16} - 2^8 + 2^0$ sprowadza się do użycia subtraktora, konieczny jest także końcowy sumator 24–bitowy. Pełny konwerter dla czynników $2^{24}+1$ wymaga kilku operacji modulo 97, układu mnożenia przez 9–bitową stałą 673 i kilku sumatorów RCA.

Konwersja dla $2^{30}-1$

Liczbę $2^{30}-1$ można rozłożyć na 6 czynników 7,9,11,31,151,331. W celu uproszczenia operacji występujących w ostatnim stopniu konwertera operującym na liczbach o największej szerokości, czynniki te pogrupowano w dwie trójki będące czynnikami liczb $2^{15} \pm 1$. Dzięki temu można zastosować szereg uproszczeń, podobnie jak w konwerterze opisanym równaniem (2.86).

Obliczenie wartości reszty modulo $2^{15}-1$ można przeprowadzić w dwóch krokach. Pierwszy z

nich obejmuje wyznaczenie reszty dla pary $7 \cdot 151 = 1057$ według wzoru

$$|X|_{1057} = |X|_{151} + \left| \underbrace{\left|151^{-1}\right|_7}_{2} \cdot (|X|_7 - |X|_{151}) \right|_7 \cdot 151. \tag{2.89}$$

Znając $|X|_{1057}$ można obliczyć resztę modulo $2^{15} - 1$ jako

$$|X|_{2^{15}-1} = |X|_{1057} + \left| \underbrace{\left|1057^{-1}\right|_{31}}_{21} \cdot (|X|_{31} - |X|_{1057}) \right|_{31} \cdot \underbrace{1057}_{10000100001_2}. \tag{2.90}$$

Mnożenie różnicy $|X|_{31} - |X|_{1057}$ przez stałą 21 modulo 31 można zaimplementować z użyciem sumatora z przeniesieniem okrężnym dodającego odpowiednio przesunięte pola wektorów reprezentujących $|X|_{31}$ i $|X|_{1057}$. Mnożenie przez stałą $1057 = 2^{10} + 2^5 + 2^0$ sprowadza się do konkatenacji wektora 5–bitowego reprezentującego resztę modulo 31. Wszystkie operacje modulo konieczne do wykonania w układzie wyznaczenia reszty modulo $2^{15} - 1$ przeprowadzane są wyłącznie dla modułów 7 i 31, dla których możliwa jest efektywna implementacja z wykorzystaniem okresowości potęg 2 modulo.

Podczas obliczania reszty modulo $2^{15}+1$ można znacznie zredukować wartości modułów, dla których należy wykonywać operacje modulo, kosztem wzrostu złożoności stałej występującej w ostatnim etapie konwersji. Ponieważ jednak liczba mnożona przez stałą jest reprezentowana 4–bitowym wektorem, mnożenie przez stałą można zaimplementować za pomocą prostego układu złożonego z pojedynczych 16–bitowych pamięci na każdy bit wyniku. Pamięci takie są efektywnie implementowane w FPGA. Sekwencja operacji wymaganych do obliczenia reszty modulo $2^{15} + 1$ rozpoczyna się znalezieniem reszty dla pary $11 \cdot 331 = 3641$ zgodnie z formułą

$$|X|_{3641} = |X|_{331} + \left| \underbrace{\left|331^{-1}\right|_{11}}_{1} \cdot (|X|_{11} - |X|_{331}) \right|_{11} \cdot 331. \tag{2.91}$$

Implementacja operacji opisanej równaniem (2.91) wymaga wyznaczenia reszty modulo 11 dla $|X|_{331}$ i użycia subtraktora modulo 11 dla obliczenia różnicy $|X|_{11} - |X|_{331}$. Jeżeli ostatnim elementem subtraktora jest pamięć ROM, jej zawartość może zostać zmodyfikowana w celu uwzględnienia operacji mnożenia przez 331. Po znalezieniu wartości $|X|_{3641}$ wartość reszty modulo $2^{15} + 1$ jest opisana wzorem

$$|X|_{2^{15}+1} = |X|_{3641} + \left| \underbrace{\left|3641^{-1}\right|_9}_{2} \cdot (|X|_9 - |X|_{3641}) \right|_9 \cdot 3641. \tag{2.92}$$

W implementacji wzoru (2.92) wynik operacji modulo może zostać obliczony z użyciem sumatora wielooperandowego z przeniesieniem okrężnym dodającego odpowiednie pola wektorów reprezentujących $|X|_9$ i $|X|_{3641}$. Jeżeli ostatnim stopniem sumatora będzie pamięć ROM, może ona zawierać jednocześnie wynik mnożenia przez 3641.

Po obliczeniu reszt modulo $2^{15}-1$ i $2^{15}+1$ wartość reszty modulo $2^{30}-1$ dana jest wzorem

$$|X|_{2^{30}-1} = |X|_{2^{15}+1} + \left| \underbrace{\left|(2^{15}+1)^{-1}\right|_{2^{15}-1}}_{2^{14}} \cdot (|X|_{2^{15}-1} - |X|_{2^{15}+1}) \right|_{2^{15}-1} \cdot (2^{15}+1). \tag{2.93}$$

Implementacja operacji opisanej wzorem (2.93) może zostać przeprowadzona według tych samych zasad, co dla (2.80) i (2.86).

Konwersja dla $2^{30}+1$

Konwersja odwrotna dla $2^{30}+1$ jest najbardziej kosztownym rozwiązaniem z dotychczasowych. Poniższe wzory opisują dwa etapy konwersji odwrotnej. Pierwszy etap sprowadza się do znalezienia reszt modulo $33025 = 2^{15}+2^8+1$ i $31513 = 2^{15}-2^8+1$, drugi etap ma na celu obliczenie na ich podstawie reszty modulo $2^{30}+1$.

Wartość reszty modulo 33025 opisana jest wzorem

$$|X|_{33025} = |X|_{1321} + \left| \underbrace{\left|1321^{-1}\right|_{25}}_{6} \cdot (|X|_{25} - |X|_{1321}) \right|_{25} \cdot 1321. \tag{2.94}$$

We wzorze (2.94) konieczne jest zastosowanie generatora modulo 25 dla $|X|_{1321}$, pełnego subtraktora modulo i układu mnożenia przez stałą 1321. Niewielkim udogodnieniem jest niska wartość odwrotności multiplikatywnej $|1321^{-1}|_{25} = 6$.

Wartość reszty modulo 32513 określona jest przez reszty dla trzech czynników $13, 41, 61$, zatem konwersja odwrotna według (2.34) wymaga układu dwupoziomowego. Możliwe są dwie sekwencje działań pozwalające na uzyskanie niewielkich wartości odwrotności multiplikatywnych występujących we wzorze (2.34). W pierwszym przypadku należy najpierw obliczyć resztę modulo $41 \cdot 61 = 2501$, natomiast w drugim jako pierwsza wyznaczana jest reszta modulo $13 \cdot 61 = 793$. Konwersja odwrotna według pierwszej sekwencji opisana jest wzorem

$$\begin{aligned} |X|_{2501} &= |X|_{41} + \left| \underbrace{\left|41^{-1}\right|_{61}}_{3} \cdot (|X|_{61} - |X|_{41}) \right|_{61} \cdot 41 \\ |X|_{32513} &= |X|_{2501} + \left| \underbrace{\left|2501^{-1}\right|_{13}}_{8} \cdot (|X|_{13} - |X|_{2501}) \right|_{13} \cdot 2501 \end{aligned}, \tag{2.95}$$

druga sekwencja wymaga operacji opisanych jako

$$\begin{aligned} |X|_{793} &= |X|_{61} + \left| \underbrace{\left|61^{-1}\right|_{13}}_{3} \cdot (|X|_{13} - |X|_{61}) \right|_{13} \cdot 61 \\ |X|_{32513} &= |X|_{793} + \left| \underbrace{\left|793^{-1}\right|_{41}}_{3} \cdot (|X|_{41} - |X|_{793}) \right|_{41} \cdot 793 \end{aligned} . \tag{2.96}$$

We wzorze (2.95) konieczne jest mnożenie przez dużą stałą $2501 = 100111000101_2$, natomiast mnożenie przez jej odwrotność multiplikatywną modulo 13 sprowadza się do zwykłego przesunięcia wektora bitowego. Zaletą (2.95) jest niewielka szerokość reszty modulo 13. Dzięki temu mnożenie przez stałą można zaimplementować za pomocą pamięci ROM zawierającej jedną tablicę LUT na bit wyniku.

Stała $793 = 1100011001_2$ we wzorze (2.96) jest o 1 bit mniejsza od 2501, ale mnożenie przez jej odwrotność multiplikatywną modulo 41 wymaga użycia sumatora. Dodatkowo, do zapisania reszty modulo 41 wymagane jest słowo o szerokości sześciu bitów, tak więc implementacja przy pomocy pamięci ROM mnożenia przez stałą 793 wymaga co najmniej dwóch tablic LUT na bit wyniku.

Po obliczeniu wartości reszt modulo 33025 i 32513 wartość reszty modulo $2^{30} + 1$ może być wyznaczona zgodnie z równaniem konwersji odwrotnej określonym przez chińskie twierdzenie o resztach. Odpowiednia formuła przyjmuje postać

$$|X|_{2^{30}+1} = \left| 33025 \cdot \left|33025^{-1}\right|_{32513} \cdot |X|_{32513} + 32513 \cdot \left|32513^{-1}\right|_{33025} \cdot |X|_{33025} \right|_{2^{30}+1}, \tag{2.97}$$

skąd po przekształceniach

$$\begin{aligned} |X|_{2^{30}+1} &= \left| 33025 \cdot 16193 \cdot |X|_{32513} + 32513 \cdot 16577 \cdot |X|_{33025} \right|_{2^{30}+1} = \\ &= \left| (2^{15} + 2^8 + 1) \cdot (2^{14} - 2^8 + 2^6 + 1) \cdot |X|_{32513} + \right. \\ &\quad \left. + (2^{15} - 2^8 + 1) \cdot (2^{14} + 2^8 - 2^6 + 1) \cdot |X|_{33025} \right|_{2^{30}+1} \end{aligned} \tag{2.98}$$

można otrzymać następujące równanie konwersji odwrotnej

$$|X|_{2^{30}+1} = \left| (2^{29} - 2^{21} + 2^6 + 1) \cdot |X|_{32513} + (2^{29} + 2^{21} - 2^6 + 1) \cdot |X|_{33025} \right|_{2^{30}+1} . \tag{2.99}$$

Równanie (2.99) może zostać zaimplementowane z użyciem wielooperandowego sumatora z przeniesieniem okrężnym dodającego odpowiednio przesunięte wektory sumy i różnicy liczb $|X|_{32513}$ i $|X|_{33025}$. W tym przypadku zaprezentowane równanie konwersji odwrotnej pozwala uniknąć konieczności wykonywania operacji modulo 32513 lub 33025 niezbędnych w przypadku korzystania z (2.34).

2.4 Podsumowanie

Analiza podstawowych metod tworzenia obrazów fotorealistycznych pozwala zidentyfikować ich najbardziej istotne własności wykorzystywane w dalszej części pracy. Podstawową cechą obu algorytmów jest podatność na implementację na maszynach równoległych. W przypadku śledzenia promieni możliwe jest przeprowadzanie obliczeń dla każdego promienia niezależnie od pozostałych. Co więcej, sąsiednie promienie najczęściej wędrują przez ten sam fragment przestrzeni, możliwe jest więc wykorzystanie równoległości na poziomie danych (*ang. Single Instruction Multiple Data, SIMD*) [Sch06]. W metodach energetycznych rzucanie promieni może zostać wykorzystane do wyznaczania współczynników sprzężenia. Możliwe jest także zastosowanie maszyn równoległych do rozwiązywania układu równań oświetlenia. Jednostka sprzętowego wspomagania obliczeń w AOG powinna zatem być konstrukcją złożoną z pracujących równolegle elementów obliczeniowych. Łatwość zrównoleglenia obliczeń pozwoli także na zmniejszenie kosztów związanych z przygotowaniem danych (np. konwersją pomiędzy systemem pozycyjnym a resztowym), ponieważ dane te będą mogły być wykorzystane przez wiele układów obliczeniowych.

Drugą istotną właściwością AOG jest intensywne wykorzystanie mnożenia i dodawania. Udział pozostałych działań arytmetycznych, w tym porównywania liczb i dzielenia, jest znacznie mniejszy. Pozwala to mieć uzasadnioną nadzieję na zwiększenie wydajności sprzętowych implementacji powyższych algorytmów za pomocą arytmetyki resztowej. Konieczne będą pewne modyfikacje dotychczasowych rozwiązań, np. wyodrębnienie wszystkich operacji różnych od dodawania i mnożenia i umieszczenie ich poza torem obliczeniowym. Strukturę procesora pozwalającą na sprzętowe wspomaganie AOG oraz wyniki implementacji kluczowych operacji AOG z wykorzystaniem arytmetyki resztowej zaprezentowano w rozdz. 4.4.

Resztowe systemy liczbowe umożliwiają reprezentację dużych liczb za pomocą zbioru reszt o niewielkich wartościach. Reprezentacja ta pozwala na implementację dodawania, odejmowania i mnożenia za pomocą zestawu niewielkich jednostek pracujących równolegle i niezależnie od siebie. Przekłada się to na zmniejszenie rozmiaru układów i poboru mocy oraz zwiększenie szybkości. Zyski te są szczególnie widoczne w układach mnożących, gdzie rozmiar jednostek w klasycznych systemach pozycyjnych rośnie z kwadratem liczby bitów operandów. Korzyści wynikające z zastosowania RNS w układach arytmetycznych rosną wraz ze zmniejszaniem się wartości modułów w bazie.

Pomimo powyższych zalet obszar zastosowań RNS jest ograniczony. Jest to spowodowane przede

wszystkim wysokim kosztem wykonywania niektórych trudnych operacji arytmetycznych i konwersji z/na system pozycyjny. Stosowanie konwerterów pomiędzy RNS a systemami pozycyjnymi wynika z konieczności współpracy torów resztowych z istniejącymi systemami cyfrowymi wykorzystującymi pozycyjne systemy liczbowe. Poza tym, stosowanie pozycyjnych systemów liczbowych jest znacznie bardziej naturalne dla człowieka, który z reguły jest odbiorcą wyników. Dodatkową przyczyną wymuszającą używanie konwerterów jest brak uniwersalnego RNS przydatnego w wielu zastosowaniach, ponieważ wybór bazy RNS jest zawsze dokonywany dla określonego zakresu dynamicznego i wykonywanych operacji. Podstawową wadą konwerterów jest ich wysoki koszt, szczególnie dla RNS o bazach złożonych z dużej liczby niewielkich modułów. Istnieją niewielkie i wydajne układy konwerterów dla specyficznych zbiorów modułów, ale zbiory te zawierają niewielką liczbę modułów o dużych wartościach, co zwiększa koszt układów arytmetycznych w resztowym torze obliczeniowym.

Oprócz stosowania konwerterów drugą wadą RNS są skomplikowane algorytmy wykonywania trudnych operacji arytmetycznych. Do operacji trudnych w RNS zalicza się dzielenie, porównywanie liczb, wykrywanie nadmiaru multiplikatywnego i addytywnego oraz detekcję znaku, które to operacje są istotne w procesorach ogólnego zastosowania. Dla wybranych klas RNS jest możliwe efektywne przeprowadzanie niektórych operacji trudnych, co zostało pokazane na przykładzie nowej metody detekcji znaku dla RNS$(2^k - 1, 2^k, 2^k + 1)$. Metoda ta pozwala na implementację układu detekcji znaku o znacznie mniejszym obszarze i krótszej ścieżce krytycznej od dotychczasowych rozwiązań. Niestety, z powodu dużych wartości modułów implementacja mnożenia i dodawania w RNS$(2^k - 1, 2^k, 2^k + 1)$ jest kosztowna.

Dodatkową wadą RNS jest komplikacja jednostek arytmetycznych implementujących dodawanie, odejmowanie i mnożenie. Operacje te muszą być przeprowadzane modulo poszczególne moduły bazy RNS. Opracowano wiele struktur jednostek arytmetycznych modulo, lecz większość z nich jest nieefektywnie implementowana w układach FPGA. Rozwiązania dedykowane dla FPGA wykorzystują pamięci ROM o pojemności zależnej wykładniczo od szerokości modułu, dlatego też ich stosowanie dla modułów o średnich i dużych wartościach jest nieopłacalne.

Stosowanie RNS jest uzasadnione jedynie wtedy, gdy zyski w układach arytmetycznych przeważą nad dodatkowymi kosztami. Z tego powodu systemy resztowe są przydatne jedynie w sytuacjach, gdzie po przeprowadzeniu pojedynczej konwersji można wykonać dużą liczbę mnożeń i dodawań bez konieczności przeprowadzania pozostałych operacji arytmetycznych. Przykładem jest wiele algorytmów DSP, takich jak filtracja cyfrowa czy transformaty numeryczne.

Hierarchiczne resztowe systemy liczbowe pozwalają ominąć ograniczenia RNS spowodowane wzrostem złożoności konwerterów w miarę zmniejszania wartości modułów stanowiących bazę RNS. W rozdz. 2.3 zaproponowano nową klasę HRNS zbudowaną z czynników modułów $2^k \pm 1$ i modułu 2^k. Pozwala to na wykorzystanie opracowanych dla RNS$(2^k - 1, 2^k, 2^k + 1)$ efektywnych konwerterów i układów realizujących operacje trudne w RNS. Podano zbiory modułów umożliwiających implementację proponowanych HRNS o zakresie dynamicznym do 90 bitów. Zaproponowano efektywne konwertery wykorzystujące pamięci ROM o niewielkiej pojemności i sumatory dodające wektory o szerokościach równych okresom/półokresom potęg 2 modulo moduły HRNS. Podano równania konwersji odwrotnej możliwe do zaimplementowania za pomocą niewielkich i szybkich układów, w których operacje modulo są przeprowadzane modulo moduły znacznie mniejsze od zakresu pełnego systemu.

Podstawową zaletą proponowanych HRNS jest znaczne uproszczenie konwersji z i do systemu pozycyjnego. Dzięki temu można mieć nadzieję na ich efektywne zastosowanie nawet dla krótkich torów obliczeniowych. Niestety, wadą prezentowanych HRNS jest dość duża wartość niektórych modułów ich bazy. Ponieważ obszar dotychczas opracowanych dedykowanych dla FPGA układów mnożących dla dowolnego modułu rośnie wykładniczo z szerokością modułu, konieczne jest poszukiwanie nowych struktur pozwalających na implementację resztowych układów arytmetycznych w FPGA.

Rozdział 3

Układy arytmetyki resztowej w FPGA

Celem rozdziału jest zaprezentowanie nowych metod zwiększenia wydajności układów arytmetycznych wykonujących mnożenie i dodawanie implementowanych w układach FPGA [Tom06b], [Tom06a]. Proponowane techniki mogą być stosowane dla wszystkich matryc FPGA zawierających pamięci o małej pojemności używane jako generatory funkcji oraz dodatkową logikę pozwalającą na efektywną implementację sumatorów ze skrośną propagacją przeniesień i układów mnożących $2 \times k$ bitów. Wszystkie eksperymenty były przeprowadzane z użyciem układów rodziny Spartan 2 firmy Xilinx.

Rozdział ma dwie części. Najpierw zaprezentowano struktury jednostek arytmetyki resztowej o złożoności mniejszej od dotychczasowych rozwiązań [Tom06b]. Zasadniczą zaletą proponowanych układów jest wyeliminowanie pamięci o dużych pojemnościach na rzecz efektywnego wykorzystania podstawowych elementów dostępnych w matrycach FPGA. Umożliwia to znaczne zmniejszenie zajmowanego obszaru przy zachowaniu wysokiej wydajności. Dodatkowo, prezentowane struktury pozwalają na skonstruowanie układów o różnym obszarze i opóźnieniu dla tego samego modułu i wykonywanego działania. Dla wielu RNS wpływ na opóźnienie całego układu ma pewien podzbiór modułów, ponieważ długość ścieżki krytycznej jednostek arytmetycznych dla pozostałych modułów jest zdecydowanie mniejsza. Zaproponowana ramowa struktura umożliwia zbudowanie jednostek o zbliżonej długości ścieżki krytycznej dla wszystkich modułów. W przypadku części modułów skutkuje to zmniejszeniem zajmowanego obszaru, ponieważ nie jest konieczne użycie najszybszych rozwiązań. Konsekwencją jest zmniejszenie obszaru zajmowanego przez resztowy tor obliczeniowy bez wpływu na opóźnienie całego układu. Dotychczasowe metody konstruowania układów arytmetyki resztowej definiują tylko jedną konfigurację jednostki w ramach danej metody uniemożliwiając dopasowanie parametrów jednostki do pozostałej części układu.

W drugiej części zaproponowano algorytm konstruowania proponowanych resztowych jednostek arytmetycznych pozwalający na znalezienie struktur bliskich optymalnym [Tom06a]. Algorytm automatycznej generacji prezentowanych jednostek sprowadza się do wygenerowania pewnego podzbioru wszystkich możliwych struktur układu, a następnie wyboru układu o wymaganych parametrach AT. Złożoność algorytmu jest wykładnicza, jednak przyjęte ograniczenia na zbiór dopuszczalnych rozwiązań umożliwiają uzyskanie wyniku dla modułów $M_* < 10^5$ w krótkim czasie.

3.1 Jednostki arytmetyczne w strukturach FPGA

Proponowana struktura jednostek arytmetycznych pozwala na konstrukcję układu obliczającego iloczyn korygowany sumą, czyli wartość

$$W_* = \left| X_* \cdot Y_* + \sum_{i=1}^{l} Z_*^i \right|_{M_*}, \tag{3.1}$$

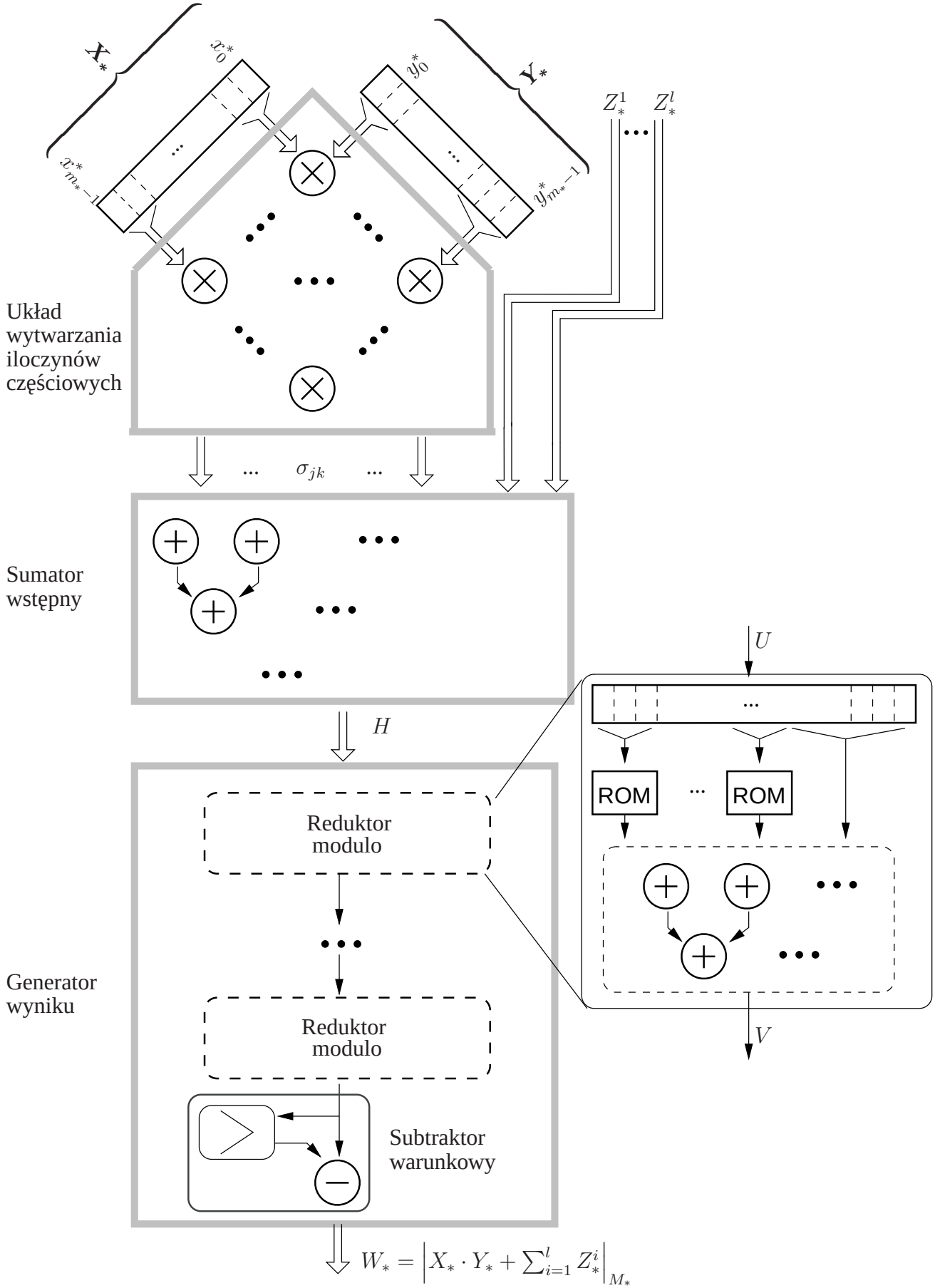
gdzie liczby naturalne $X_*, Y_*, Z_*^i, W_* \in [0, M_*)$, a $M_* > 1$ jest wartością modułu. Wszystkie operandy oraz wynik reprezentowane są w systemie naturalnym binarnym (NB) za pomocą wektorów bitowych. Dowolną liczbę naturalną $X_* \in [0, M_*)$ można przedstawić w systemie NB jako wektor binarny $\boldsymbol{X_*} = x^*_{m_*-1}, \ldots, x^*_0$ o m_* pozycjach, przy tym

$$X_* = \sum_{i=0}^{m_*-1} 2^i \cdot x_i^*, \tag{3.2}$$

gdzie $x_i^* \in \{0, 1\}$, zaś $m_* = \lfloor \log_2(M_*) \rfloor + 1$ jest *szerokością* wektora.

Schemat struktury prezentowanych jednostek przedstawiono na rys. 3.1. Zawiera on trzy bloki: układ wytwarzania iloczynów częściowych, sumator wstępny i generator wyniku. Blok wytwarzania iloczynów częściowych realizuje dwa zadania. Pierwszym z nich jest podział wektorów $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ na pola o szerokości co najmniej dwóch bitów. Drugą operacją jest obliczenie wartości iloczynów częściowych dla wszystkich kombinacji dwuelementowych pól wektora $\boldsymbol{X_*}$ z polami wektora $\boldsymbol{Y_*}$. Iloczyn j–tego pola wektora $\boldsymbol{X_*}$ z k–tym polem wektora $\boldsymbol{Y_*}$ oznaczono symbolem σ_{jk}. Iloczyny częściowe σ_{jk} mogą być także obliczone jako reszty modulo M_*, co pozwala uprościć strukturę pozostałych bloków jednostki arytmetycznej. Niestety, redukcja modulo pojedynczych iloczynów częściowych powoduje wzrost złożoności układu wytwarzania iloczynów częściowych.

Wszystkie obliczone iloczyny częściowe i dodatkowe składniki Z_*^i są akumulowane w sumatorze



Rysunek 3.1. Struktura resztowych jednostek arytmetycznych.

wstępnym. Wyjściem sumatora wstępnego jest liczba

$$H = \sum_{j} \sum_{k} \sigma_{jk} + \sum_{i=1}^{l} Z_*^i. \tag{3.3}$$

Suma wszystkich iloczynów częściowych oraz dodatkowych składników Z_*^i jest następnie przekazywana do generatora wyniku, gdzie obliczana jest wartość wyniku W_* jako

$$W_* = |H|_{M_*}. \tag{3.4}$$

Obliczanie reszty $|H|_{M_*}$ w reduktorze modulo jest przeprowadzane dwuetapowo. Pierwszym krokiem jest redukcja wartości H za pomocą kaskadowego połączenia reduktorów modulo obliczających taką liczbę V na podstawie U, aby $V \ll U$ i $V \equiv U \bmod M_*$. Następnie od liczby wyjściowej V ostatniego reduktora odejmowana jest taka krotność M_*, aby wynik był liczbą z zakresu $[0, M_*)$

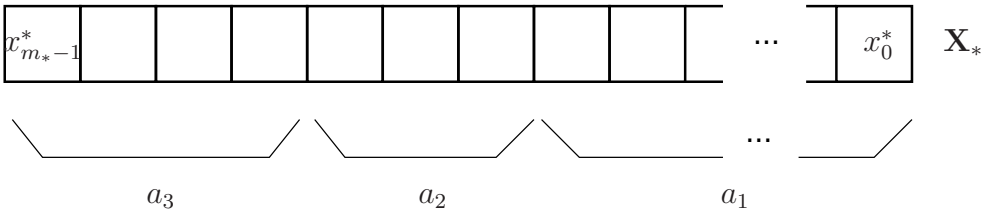
$$W_* = \begin{cases} V & \text{dla} \quad V \in [0, M_*) \\ V - M_* & \text{dla} \quad V \in [M_*, 2 \cdot M_*) \\ V - 2 \cdot M_* & \text{dla} \quad V \in [2 \cdot M_*, 3 \cdot M_*) \\ & \dots \end{cases}. \tag{3.5}$$

Przedstawiona ogólna idea działania proponowanej jednostki arytmetycznej modulo może zostać zaimplementowana dowolnie. Została ona jednak opracowana w sposób umożliwiający szczególnie efektywną implementację w matrycach FPGA. Poniżej zostaną przedstawione metody implementacji poszczególnych podukładów za pomocą elementów dostępnych w układach FPGA rodzin Spartan 2/Virtex firmy Xilinx.

3.1.1 Układ wytwarzania iloczynów częściowych

Układ wytwarzania iloczynów częściowych (UWIC) jest pierwszym blokiem ramowej struktury z rys. 3.1. Zadaniem bloku wytwarzania iloczynów częściowych jest podział wektorów $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ na pola bitowe i obliczenie iloczynów σ_{jk} dla kombinacji tych pól. Wszystkie iloczyny częściowe są obliczane jednocześnie.

Sposób podziału wektorów $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ jest wybierany na etapie projektowania jednostki. W dalszej części pracy podział $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ opisują wektory $\boldsymbol{a}$ i $\boldsymbol{b}$ wybrane spośród kompozycji m_* nie zawierających elementów równych 1 (rozdz. 2.2.1, str. 45). Dla przykładu, zbiór dopuszczalnych kompozycji liczby 7 to $\Gamma(7) = \{(2,2,3),(2,3,2),(3,2,2),(2,5),(5,2),(3,4),(4,3),(7)\}$. Kolejne pozycje a_j, b_k oznaczają liczbę bitów wchodzącą w skład j–tego pola $\boldsymbol{X_*}$ i k–tego pola $\boldsymbol{Y_*}$ (rys. 3.2).



Rysunek 3.2. Znaczenie elementów wektora **a**.

Znając sposób podziału $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ można opisać wartości reprezentowane przez poszczególne pola bitowe. Najstarszym bitem j–tego pola wydzielonego z $\boldsymbol{X_*}$ jest bit na pozycji

$$a(j) = \sum_{i=1}^{j} a_i - 1, \tag{3.6}$$

a więc pole to zawiera bity $x^*_{a(j):a(j)-a_j+1}$. Wartość zapisana na j–tym polu wynosi zatem

$$X_*(j) = \sum_{i=a(j)-a_j+1}^{a(j)} 2^i \cdot x^*_i, \tag{3.7}$$

przy czym

$$X_* = \sum_j X_*(j). \tag{3.8}$$

Analogiczne zależności można zapisać dla wektorów $\boldsymbol{b}$ i $\boldsymbol{Y_*}$.

Po podziale $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ na pola bitowe, wzór (3.1) można zapisać w postaci

$$W_* = \left| \sum_j \sum_k X_*(j) \cdot Y_*(k) + \sum_i Z^i_* \right|_{M_*}. \tag{3.9}$$

Z niezmienniczości kongruencji względem dodawania i mnożenia wynika, że wartości wszystkich, lub tylko niektórych, iloczynów $X_*(j) \cdot Y_*(k)$ można zastąpić wartościami ich reszt modulo M_*. Wartości iloczynów częściowych są zatem dane jako

$$\sigma_{jk} = \begin{cases} X_*(j) \cdot Y_*(k) \\ \text{lub} \\ |X_*(j) \cdot Y_*(k)|_{M_*} \end{cases}. \tag{3.10}$$

Wybór metody obliczania iloczynu σ_{jk} dla konkretnej pary $X_*(j)$, $Y_*(k)$ dokonywany jest podczas projektowania struktury jednostki. W dalszej części pracy sposób generowania iloczynu σ_{jk} opisany jest macierzą $\boldsymbol{c}$. Rozmiar macierzy zależy od szerokości wektorów $\boldsymbol{a}$ i $\boldsymbol{b}$. Symbol c_{jk} oznacza element znajdujący się w j–tej kolumnie i k–tym wierszu. Wartość elementu c_{jk} opisuje metodę generowania

iloczynu częściowego zgodnie ze wzorem

$$\sigma_{jk} = \begin{cases} X_*(j) \cdot Y_*(k) & \text{dla} \quad c_{jk} = 1 \\ |X_*(j) \cdot Y_*(k)|_{M_*} & \text{dla} \quad c_{jk} = 0 \end{cases} . \tag{3.11}$$

Podział wektorów $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ na mniejsze pola pozwala na zastosowanie zbioru układów o niewielkiej złożoności obliczających wartości iloczynów częściowych σ_{jk}. Dla średnich i dużych szerokości wektorów $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$ suma obszarów zajętych przez układy obliczające σ_{jk} oraz sumator dodający wszystkie iloczyny częściowe jest mniejsza od obszaru pojedynczej pamięci ROM wyznaczającej wartość iloczynu X_* i Y_* modulo M_*. Dodatkowo sumator iloczynów częściowych może zostać łatwo poszerzony o możliwość dodawania dowolnej liczby składników Z_*^i.

Iloczyny σ_{jk} mogą być obliczane na dwa sposoby: jako wynik bezpośredniego mnożenia odpowiednich pól lub jako reszta modulo M_* dla wyniku mnożenia pól. Sposób obliczania wartości σ_{jk} jednoznacznie określa, jaki element musi być użyty do obliczenia danego iloczynu częściowego. Wartość iloczynu częściowego σ_{jk} może zostać obliczona za pomocą pamięci ROM lub układu mnożącego. Układy te są efektywnie implementowane w używanych w pracy matrycach FPGA.

Zaletą implementacji za pomocą pamięci ROM jest możliwość bezpośredniego uzyskania wyniku w postaci reszty modulo M_*. Niestety, rozmiar pamięci ROM zależy wykładniczo od sumy szerokości pól, dla których obliczany jest iloczyn częściowy. Stosowanie pamięci ROM jest więc opłacalne jedynie dla niewielkich szerokości tych pól.

Znacznie mniejszy obszar dla dużych szerokości pól zajmują implementacje z użyciem układów mnożących. Układ mnożenia liczb binarnych konstruowany jest z układów mnożenia $2 \cdot k$ bitów, których wyjścia są następnie sumowane. Obszar i opóźnienie wnoszone przez pojedynczy układ mnożący $2 \cdot k$ bitów jest porównywalne z obszarem i opóźnieniem dla sumatora $(k+1)$–bitowego. Wadą układów mnożących jest brak redukcji modulo M_* dla wyniku mnożenia pól wektorów $\boldsymbol{X_*}$ i $\boldsymbol{Y_*}$. Powoduje to zwiększenie wartości sumy H i w konsekwencji wzrost złożoności generatora wyniku.

Struktura podstawowych elementów obliczających σ_{jk} powoduje, że gdyby pola o wartościach $X_*(j)$ i $Y_*(k)$ mogły mieć szerokość 1 bitu, elementy te mogły by zostać częściowo niewykorzystane. Minimalny rozmiar pamięci ROM to 16×1 bitów, tak więc suma szerokości pól o wartościach $X_*(j)$ i $Y_*(k)$ powinna wynosić co najmniej 4 bity. Układ mnożący $1 \cdot k$ bitów wymaga takiej samej ilości zasobów, jak układ mnożenia $2 \cdot k$ bitów. Stosowanie pól o szerokości mniejszej od 2 bitów skutkuje zwiększeniem liczby układów mnożących i sumatorów w sumatorze wstępnym, a więc zwiększeniem złożoności układu. Ograniczenie minimalnej szerokości pola do 2 bitów gwarantuje efektywne

wykorzystanie podstawowych elementów dostępnych w matrycach FPGA.

3.1.2 Sumator wstępny

Sumator wstępny jest drugim blokiem struktury pokazanej na rys. 3.1. Zadaniem sumatora wstępnego jest obliczenie wartości H danej wzorem (3.3). Sumator wielooperandowy można zrealizować na dwa sposoby: za pomocą sumatora przechowującego przeniesienia (ang. *carry-save adder*) CSA uzupełnionego o wyjściowy sumator z propagacją przeniesień (*ang. carry propagate adder, CPA*), lub z użyciem kaskady sumatorów CPA.

Sumatory CSA są powszechnie używane w implementacjach układów ASIC ze względu na regularną strukturę i łatwość przetwarzania potokowego na poziomie pojedynczego ogniwa sumatora pełnego (*ang. full adder*) FA. Uzupełnienie drzewa CSA o szybki sumator CPA pozwala na skonstruowanie sumatora wielooperandowego o niewielkim opóźnieniu przy użyciu układów FA zajmujących niewielki obszar. Wielopoziomowy układ CPA o podobnym opóźnieniu zajmuje znacznie większy obszar.

W układach FPGA rodziny Spartan 2 sumatory CPA są sumatorami szeregowymi RCA (ang. *ripple-carry adder*), w których propagacja przeniesień następuje poprzez szybkie multipleksery. Ponieważ czas propagacji przeniesienia pomiędzy sąsiednimi ogniwami FA jest wielokrotnie mniejszy od czasu propagacji od wejścia do wyjścia LUT realizującego FA, opóźnienie sumatora o szerokości do kilkunastu bitów jest porównywane z opóźnieniem pojedynczego ogniwa FA. Niestety, implementacja ogniwa FA poza dedykowaną strukturą RCA zajmuje dwukrotnie większy obszar. Z tego powodu implementacja sumatorów CSA w strukturach FPGA jest kosztowna.

W sumatorze CSA skonstruowanym z ogniw sumatora pełnego FA pojedyncze ogniwo redukuje 1 bit wejściowy. Dla sumatora wielooperandowego zbudowanego z układów CPA, pojedynczy sumator CPA redukuje dwa k-bitowe wektory na $k + 1$ bitów sumy. Jeżeli sumatory CPA są sumatorami szeregowymi RCA, liczb użytych ogniw FA i opóźnienie dla obu struktur sumatorów wielooperandowych są porównywalne.

Ponieważ liczba układów FA wymaganych przez obie struktury sumatorów wielooperandowych jest porównywalna, sumator z użyciem kaskady CPA jest zdecydowanie lepszy w implementacji FPGA. Z tego powodu sumator wielooperandowy realizujący operację określoną równaniem (3.3) jest zbudowany jako kaskada sumatorów CPA. Liczba poziomów kaskady, przekładająca się na dłu-

gość ścieżki krytycznej, rośnie z logarytmem liczby składników równania (3.3).

Konstrukcja kaskady sumatorów

Wektory wejściowe kaskady sumatorów RCA reprezentują wszystkie iloczyny częściowe σ_{jk} i wszystkie składniki Z_*^i. Wektory reprezentujące iloczyny częściowe mogą znacznie się różnić między sobą szerokością i indeksem najniższego bitu. Z tego powodu wybór pary wektorów sumowanej za pomocą pojedynczego sumatora RCA istotnie wpływa na rozmiar całego sumatora wielooperandowego.

Niech *LSB*(σ_{jk}) oraz *MSB*(σ_{jk}) oznaczają odpowiednio numery pozycji najniższego i najwyższego bitu wektora reprezentującego σ_{jk}. Wartości tych indeksów zależą od metody generowania iloczynu częściowego oraz od zakresu wartości dla $X_*(j)$ i $Y_*(k)$. Jeśli iloczyn σ_{jk} jest obliczany jako reszta modulo M_*, indeks najniższego bitu wynosi 0. Indeks najwyższego bitu σ_{jk} może być równy co najwyżej $m_* - 1$, gdzie $m_* = \lfloor \log_2(M_*) \rfloor + 1$. Dla niektórych kombinacji wartości M_* i przyjętego sposobu podziału na pola bitowe iloczyn ρ_{jk} może wymagać wektora o mniejszej szerokości. Niestety, dokładne określenie maksymalnej wartości σ_{jk} wymaga obliczenia reszt dla iloczynów wszystkich możliwych kombinacji wartości $X_*(j)$ i $Y_*(k)$. Ze względu na wysoki koszt tej operacji założono dalej, że dla iloczynu obliczanego jako reszta modulo M_* wartość *MSB*(σ_{jk}) wynosi $m_* - 1$.

W przypadku obliczania σ_{jk} jako zwykłego iloczynu $X_*(j) \cdot Y_*(k)$ wartości *LSB*(σ_{jk}) oraz *MSB*(σ_{jk}) zależą od indeksów najniższego i najwyższego bitu mnożonych pól. Najniższym bitem pola o wartości $X_*(j)$ jest bit na pozycji $a(j) - a_j + 1$, gdzie $a(j)$ jest opisane wzorem (3.6). Indeks najniższego bitu iloczynu σ_{jk} jest równy sumie *LSB*$(X_*(j))$ + *LSB*$(Y_*(k))$, czyli $a(j) - a_j + b(k) - b_k + 2$. Indeks najwyższego bitu σ_{jk} zależy od maksymalnej wartości σ_{jk}. Górnym ograniczeniem $\max(\sigma_{jk})$ jest iloczyn maksymalnych wartości pól o wartościach $X_*(j)$ i $Y_*(k)$, tak więc w prezentowanym rozwiązaniu *MSB*(σ_{jk}) jest oszacowane za pomocą iloczynu $\max(X_*(j)) \cdot \max(Y_*(k))$.

Sposób generowania iloczynów σ_{jk} opisany jest macierzą $\boldsymbol{c}$ zgodnie ze wzorem (3.11). Wartościami indeksów najniższego i najwyższego bitu σ_{jk} są więc

$$\textit{LSB}(\sigma_{jk}) = \begin{cases} 0 & \text{dla} \quad c_{jk} = 0 \\ a(j) + b(k) - a_j - b_k + 2 & \text{dla} \quad c_{jk} = 1 \end{cases} \tag{3.12}$$

oraz

$$\textit{MSB}(\sigma_{jk}) = \begin{cases} m_* - 1 & \text{dla} \quad c_{jk} = 0 \\ \lfloor \log_2(\max(X_*(j)) \cdot \max(Y_*(k))) \rfloor & \text{dla} \quad c_{jk} = 1 \end{cases}. \tag{3.13}$$

Z wzorów (3.12) i (3.13) wynika, że iloczyny σ_{jk} mogą znacznie różnić się między sobą indeksem najniższego bitu i szerokością. Minimalna wartość *LSB*(σ_{jk}) wynosi 0. Maksymalna wartość *LSB*(σ_{jk}) wystąpi dla mnożenia dwóch pól $x^*_{m_*-1:m_*-2} \cdot y^*_{m_*-1:m_*-2}$ zawierających najwyższe bity wektorów $\boldsymbol{X}_*$ i $\boldsymbol{Y}_*$, przy czym szerokość tych pól musi być minimalna (2 bity). Maksymalną wartością *LSB*(σ_{jk}) jest więc $2 \cdot m_* - 4$. Minimalna wartość *MSB*(σ_{jk}) jest dana przez minimalną szerokość pola i wynosi 3 dla mnożenia dwóch pól $x^*_{1:0} \cdot y^*_{1:0}$. Kres górny *MSB*$(\sigma_{jk})$ jest dany przez wynik mnożenia wektorów $\boldsymbol{X}_* \cdot \boldsymbol{Y}_*$ i wynosi $\lfloor \log_2((M_* - 1)^2) \rfloor$.

Dla wektorów bitowych o tak dużej rozbieżności indeksów wybór par sumowanych za pomocą pojedynczego sumatora CPA ma istotny wpływ na liczbę ogniw FA w całej kaskadzie sumatorów CPA. Liczba ogniw FA zależy od liczby i szerokości użytych sumatorów. Liczba sumatorów zależy od liczby sumowanych składników, ale rozmiar pojedynczego sumatora może zmieniać się w szerokich granicach: od szerokości najwęższego z wektorów do szerokości sumy wszystkich składników.

Lemat 3.1.1. *Niech l oznacza liczbę wektorów o szerokości co najmniej n bitów sumowanych w kaskadzie sumatorów RCA, a h szerokość sumy wszystkich wektorów. Istnieje struktura kaskady sumatorów o liczbie poziomów nie większej, niż $\log_2(l)$ i liczbie ogniw FA z przedziału $[(l-1) \cdot n, (l-1) \cdot h]$.*

Dowód. Pojedynczy sumator RCA może dodać co najmniej dwa wektory. Sumowanie większej liczby wektorów jest możliwe, gdy operandy sumatora będą wynikiem konkatenacji wektorów nie posiadających bitów o tych samych wagach, czyli *MSB*$(\sigma_1) <$ *LSB*(σ_2). Pojedynczy sumator zmniejsza więc liczbę składników sumowanych w kaskadzie o co najmniej 1, należy więc użyć co najwyżej $l-1$ sumatorów do obliczenia sumy l składników, przy czym zawsze można skonstruować kaskadę zawierającą $l-1$ sumatorów. W zależności od indeksów bitów sumowanych wektorów, liczba ogniw pojedynczego sumatora mieści się w przedziale $[n, h]$. Liczba ogniw użytych w kompletnej kaskadzie sumatorów należy więc do przedziału $[(l-1) \cdot n, (l-1) \cdot h]$.

Wszystkie wektory wejściowe kaskady są dostępne jednocześnie, można więc skonstruować kaskadę jako układ, w którym każdy poziom zredukuje liczbę operandów o połowę. Kolejne poziomy będą więc zawierały odpowiednio $\frac{l}{2}$, $\frac{l}{4}$, $\frac{l}{8}$, ... sumatorów. Sumowanie jest przeprowadzane do uzyskania pojedynczego wektora, a na każdym poziomie można dodać liczbę wektorów będącą kolejną potęgą 2. Liczba poziomów kaskady wynosi zatem $\log_2(l)$. □

Liczba ogniw pojedynczego sumatora RCA dodającego dwa wektory zależy od maksymalnej wartości sumy oraz od różnic indeksów najniższych bitów operandów. Indeks najniższego bitu wektora

reprezentującego sumę liczb naturalnych $\sigma_1 + \sigma_2$ wynosi

$$\mathit{LSB}(\sigma_1 + \sigma_2) = \min(\mathit{LSB}(\sigma_1), \mathit{LSB}(\sigma_2)). \tag{3.14}$$

Indeks najwyższego bitu $\sigma_1 + \sigma_2$ zależy od maksymalnych wartości składników, a więc

$$\mathit{MSB}(\sigma_1 + \sigma_2) = \lfloor \log_2(\max(\sigma_1) + \max(\sigma_2)) \rfloor . \tag{3.15}$$

Pojedynczy sumator RCA dodający dwa wektory bitowe reprezentujące σ_1 i σ_2 wymaga $\max(\mathit{MSB}(\sigma_1), \mathit{MSB}(\sigma_2)) - \max(\mathit{LSB}(\sigma_1), \mathit{LSB}(\sigma_2)) + 1$ ogniw FA. Szerokość sumy jest określona jako $\mathit{MSB}(\sigma_1 + \sigma_2) - \mathit{LSB}(\sigma_1 + \sigma_2) + 1$ bitów.

Najmniejsza liczba ogniw sumatora zostanie użyta wtedy, gdy każde z nich zredukuje dwa bity wejściowe. Stanie się tak jedynie wtedy, gdy wagi bitów wejściowych będą tak dobrane, aby zawsze można było znaleźć dwa wektory zawierające bity na tych samych pozycjach. Obszar kaskady sumatorów będzie najmniejszy, jeśli wejścia wszystkich ogniw sumatora będą wykorzystane. Każde ogniwo, którego co najmniej jedno wejście będzie miało stałą wartość "0", zwiększa koszt kaskady nie zmniejszając jednocześnie liczby bitów. Niestety, dla wektorów wyjściowych układu wytwarzania iloczynów częściowych rozrzut szerokości i położenia wektorów utrudnia lub wręcz uniemożliwia skonstruowanie układu, w którym wszystkie wejścia ogniw sumatora będą wykorzystane. Należy więc dążyć do ograniczenia liczby ogniw z niewykorzystanymi wejściami.

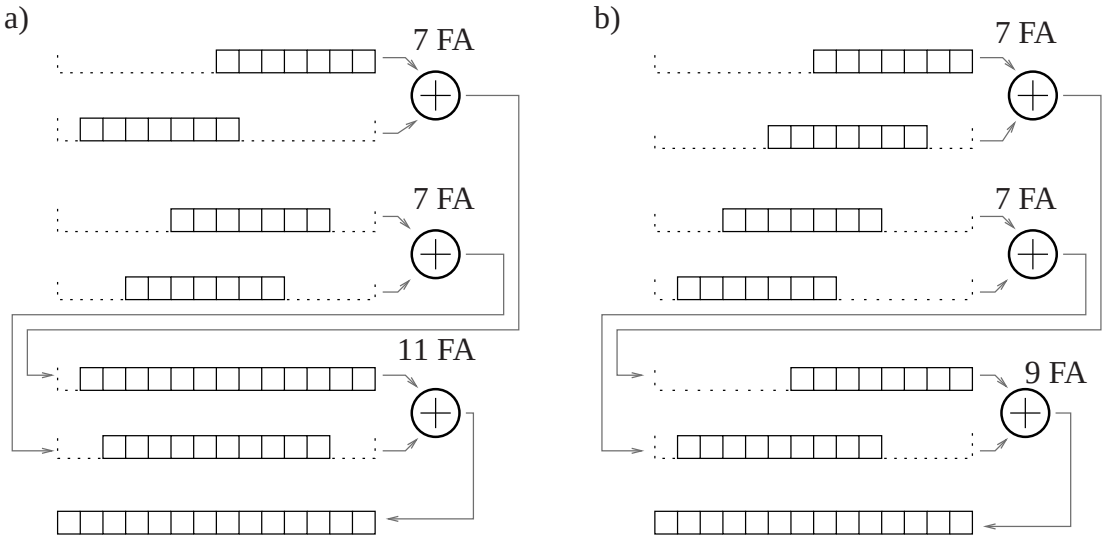
Niech σ_i^j oznacza i–ty składnik sumowany na j–tym poziomie kaskady. Zgodnie z wzorami (3.14) i (3.15), liczba ogniw w całej kaskadzie wynosi

$$\sum_{j>1} \sum_{i} \left(\max(\mathit{MSB}(\sigma_{\kappa(j,i)}^{j-1}), \mathit{MSB}(\sigma_{\kappa'(j,i)}^{j-1})) - \max(\mathit{LSB}(\sigma_{\kappa(j,i)}^{j-1}), \mathit{LSB}(\sigma_{\kappa'(j,i)}^{j-1})) + 1 \right), \tag{3.16}$$

gdzie $\kappa(j,i), \kappa'(j,i)$ oznaczają funkcje definiujące indeksy składników sumowanych w i–tym sumatorze na j–tym poziomie. Znalezienie takich wartości $\kappa(j,i), \kappa'(j,i)$, dla których wartość funkcji (3.16) jest minimalna, jest trudne. Przyczyną jest wpływ decyzji podjętych na j–tym poziomie na parametry układu na wszystkich następnych poziomach. Na podstawie analizy zbiorów wektorów sumowanych w kaskadzie zaproponowano więc heurystyczny algorytm wyboru wektorów sumowanych na poszczególnych poziomach.

Podczas wyboru pary wektorów sumowanych za pomocą pojedynczego sumatora należy minimalizować liczbę ogniw FA użytych wyłącznie do propagacji przeniesień i nie zwiększać szerokości wektorów w kierunku najniższych bitów. Dodatkowo, ponieważ liczba sumatorów na każdym poziomie kaskady maleje o połowę, największy wpływ na obszar całej kaskady mają decyzje podjęte na

początkowych poziomach. Nie ma jednak potrzeby znalezienia rozwiązań optymalnych na pojedynczym poziomie, ponieważ nie gwarantuje to optymalnej struktury całej kaskady.



Rysunek 3.3. Przykład struktur sumatora wstępnego a) niewłaściwy dobór wektorów, b) dobór według zaprezentowanych reguł.

Liczba ogniw użytych wyłącznie do propagacji przeniesień zależy od różnicy indeksów najwyższych bitów sumowanych wektorów. Niewielki wzrost szerokości wektorów na kolejnych poziomach wystąpi wtedy, gdy dodawane będą wektory o najmniejszej różnicy najniższych bitów. Jako wektory dodawane w pojedynczym sumatorze są więc wybierane wektory o najmniejszej różnicy indeksów najwyższych bitów. Jeśli istnieje kilka par wektorów o takiej samej różnicy $\left|MSB(\sigma_1^j) - MSB(\sigma_2^j)\right|$, wybierane są wektory o najmniejszej szerokości. Przykład dwóch struktur sumatora wielooperandowego przedstawiono na rys. 3.3.

Algorytm konstrukcji kaskady sumatorów

Kompletny algorytm konstrukcji kaskady sumatorów (alg. 4) jest algorytmem iteracyjnym. Pojedyncza iteracja tworzy jeden poziom kaskady na podstawie danych o wektorach z poprzedniego poziomu. Danymi wejściowymi algorytmu jest zbiór wektorów reprezentujących wszystkie iloczyny częściowe oraz dodatkowe składniki Z_*^i. Wynikiem algorytmu jest struktura kaskady sumatorów RCA obliczającej sumę wszystkich wektorów wejściowych.

Struktura kompletnej kaskady jest opisana za pomocą wektora $\mathbf{\Theta}$. Kolejnymi składowymi tego wektora są wektory $\mathbf{\Theta}_j$ opisujące kolejne poziomy kaskady. Poszczególne sumatory j–tego poziomu

są opisane elementami θ_i^j tworzącymi wektory $\mathbf{\Theta}_j$. Każdy sumator jest opisany przez indeks najniższego bitu, maksymalną wartość reprezentowanej liczby oraz indeksy sumowanych składników z wyższego poziomu. Indeks najniższego bitu jest oznaczony jako $LSB(\theta_i^j)$, natomiast maksymalna wartość sumy jako $\max(\theta_i^j)$.

Algorytm 4 Algorytm tworzenia sumatora wstępnego

Dane wejściowe: $\mathbf{\Theta}_0$: Opis wszystkich wektorów reprezentujących iloczyny częściowe σ_{jk} i wszystkie składniki Z_*^i

1: sortuj rosnąco elementy θ_i^0 wektora $\mathbf{\Theta}_0$ według kryterium:
$\theta_i < \theta_j \iff \max(\theta_i) < \max(\theta_j)$
if $\max(\theta_i) = \max(\theta_j)$ **then** $\theta_i < \theta_j \iff LSB(\theta_i) < LSB(\theta_j)$
2: $j = 0$ {indeks poziomu kaskady sumatorów}
3: **while** $|\mathbf{\Theta}_j| > 1$ **do**
4: **for** $i = 1$ **to** $\left\lfloor \frac{|\mathbf{\Theta}_j|}{2} \right\rfloor$ **do**
5: Pobierz dwa kolejne sumatory θ_{2i}^j i θ_{2i+1}^j
6: Dla pobranych elementów konstruuj sumator θ_i^{j+1} o parametrach opisanych wzorami (3.14) i (3.15)
7: **end for**
8: **if** $|\mathbf{\Theta}_j| \equiv 1 \bmod 2$ **then**
9: dopisz ostatni element wektora $\mathbf{\Theta}_j$ na na odpowiednią pozycję w wektorze $\mathbf{\Theta}_{j+1}$
10: **end if**
11: $j = j + 1$
12: **end while**
13: **return** $\mathbf{\Theta} = (\mathbf{\Theta}_0, \mathbf{\Theta}_1, \ldots)$

Argumentem wejściowym algorytmu jest wektor $\mathbf{\Theta_0}$ zawierający opis wszystkich iloczynów częściowych obliczonych w układzie wytwarzania il. część. (rys. 3.1) i dodatkowych składników Z_*^i. Postać kaskady sumatorów jest jednoznacznie określona przez argumenty wejściowe.

Pierwszym i najważniejszym krokiem algorytmu 4 jest uporządkowanie wektorów wejściowych względem szerokości i indeksu najstarszego bitu. Posortowanie elementów gwarantuje, że dwa kolejne elementy wektora $\mathbf{\Theta}_0$ spełniają kryteria pozwalające ograniczyć liczbę ogniw kaskady sumatorów.

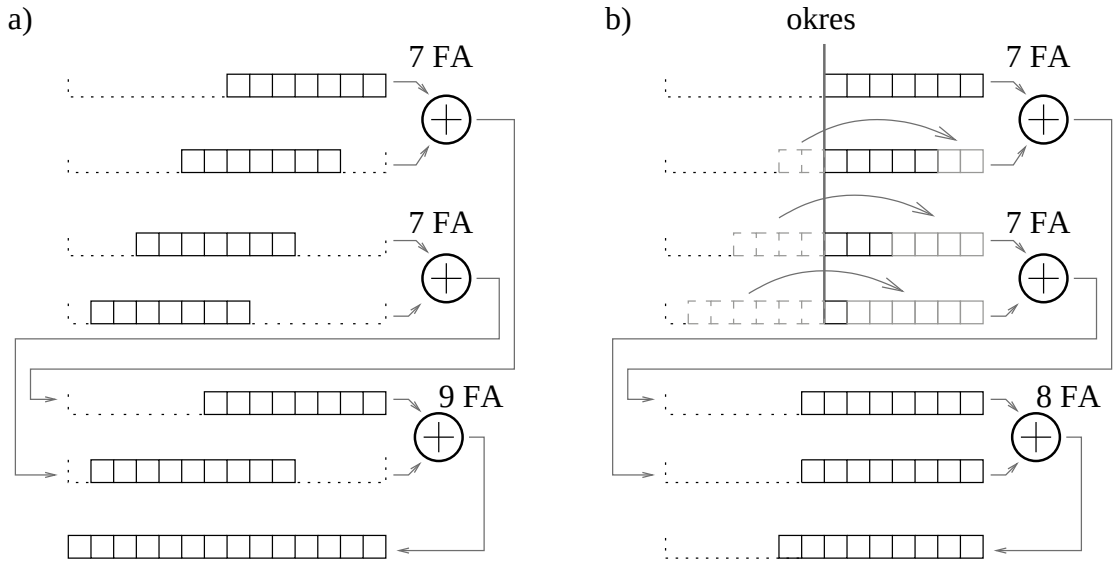
Po posortowaniu elementów wektora $\mathbf{\Theta}_0$ rozpoczyna się część iteracyjna algorytmu. Wybiera-

ne są kolejne pary elementów, po czym dla wybranej pary konstruowany jest sumator następnego poziomu. Parametry sumatora i wektora reprezentującego sumę opisane są wzorami (3.14) i (3.15). Skonstruowany sumator jest dopisywany na ostatnią pozycję wektora zawierającego opis następnego poziomu. Gwarantuje to zachowanie właściwej kolejności wektorów na każdym poziomie. Po wyczerpaniu wszystkich elementów na danym poziomie algorytm rozpoczyna kolejną iterację operując na elementach kolejnego poziomu. Warunkiem zakończenia algorytmu jest zredukowanie liczby sumatorów na danym poziomie do jednego.

Kaskada sumatorów jest jedynym elementem bloku sumatora wstępnego w jednostce pokazanej na rys. 3.1. Otrzymana suma jest następnie redukowana modulo M_*, można zatem już na etapie sumowania poszczególnych składników zmniejszyć wartość sumy przy zachowaniu informacji o reszcie modulo M_*. W opisywanej strukturze wykorzystano do tego celu okresowość potęg modulo M_* wynikającą z tw. Eulera i małego tw. Fermata (str. 43). Właściwość okresowości jest użyteczna przy konstrukcji generatorów modulo dla szerokich wektorów bitowych. Wektory te są dzielone na pola o szerokości równej okresowi bądź półokresowi potęg 2. Reszty modulo dla kolejnych bitów w wydzielonych polach są sobie równe lub stanowią swoje odwrotności addytywne, można więc te pola dodać do siebie otrzymując wynik o szerokości porównywalnej z okresem bądź półokresem. W przypadku dodawania pól o szerokości równej półokresowi należy odejmować od siebie pola zawierające bity, których reszty są odwrotnościami addytywnymi. Reszta modulo M_* dla wyniku sumowania pól jest równa reszcie modulo M_* dla liczby liczby reprezentowanej wektorem wejściowym. Jeżeli szerokość wektora jest znacznie większa od wartości okresu lub półokresu, wykorzystanie okresowości pozwala uprościć strukturę układu obliczającego resztę modulo M_*.

Wykorzystując okresowość w kaskadzie sumatorów można zmniejszyć szerokość sumy do okresu lub półokresu potęg 2 modulo M_*. Dzięki temu generator wyniku operuje na wektorze o zmniejszonej szerokości. Przekłada się to bezpośrednio na ograniczenie zajmowanego obszaru i wprowadzanego opóźnienia. W zależności od wartości okresu liczba ogniw w sumatorze wstępnym może się nieznacznie zmienić. W prototypowym rozwiązaniu obsługiwane są wyłącznie przypadki, w których okres potęg dwójki modulo M_* jest o dwa bity większy od m_*. Jest to podyktowane łatwością modyfikacji kaskady sumatorów, ponieważ dla założonych ograniczeń na minimalną szerokość pola wydzielonego z wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$ łatwo znaleźć wektory iloczynów częściowych, których indeksy najniższych bitów są wielokrotnościami 2.

Okresowość potęg 2 mod M_* jest wykorzystywana do redukcji szerokości obliczonych iloczynów



Rysunek 3.4. Przykład struktur sumatora wstępnego a) bez wykorzystania okresowości potęg 2 modulo M, b) z wykorzystaniem okresowości.

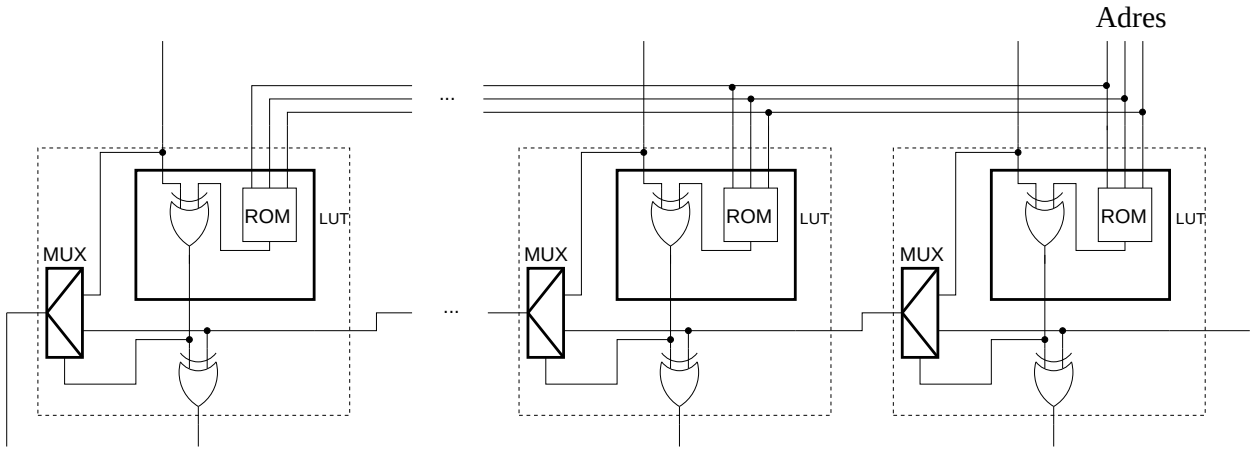
częściowych oraz dla każdej wytworzonej sumy pośredniej (rys. 3.4). Idea polega na przesunięciu bitów o najstarszych wagach na najmłodsze, nieużywane pozycje wektora. Jeśli liczba bitów o wagach większych niż okres przekracza liczbę dostępnych, wolnych pozycji, wektor pozostaje bez zmian. Pomimo prostoty rozwiązania pozwala ono na znaczną redukcję parametrów AT całej jednostki arytmetycznej dla odpowiednich wartości okresu potęg 2 mod M_*.

3.1.3 Generator wyniku

Generator wyniku jest ostatnim blokiem struktury przedstawionej na rys. 3.1. Jego zadaniem jest obliczenie reszty dla wektora wyjściowego sumatora wstępnego. Generator wyniku składa się z połączonych kaskadowo reduktorów modulo uzupełnionych o końcowy układ odejmujący niewielką krotność M_* od wyjścia ostatniego reduktora modulo. Każdy z reduktorów modulo redukuje wartość liczby wejściowej zachowując informację o jej reszcie modulo M_*. Redukcja wartości jest wynikiem sumowania reszt obliczonych dla niewielkich pól bitowych wektora wejściowego. Reszty obliczane są za pomocą pamięci ROM o małej pojemności. Układ odejmujący składa się z zestawu komparatorów ze stałymi, pamięci ROM i subtraktora. W dalszej części pracy układ odejmujący jest nazywany *subtraktorem warunkowym*.

Sposób implementacji pamięci w matrycach FPGA rodziny Spartan 2 ogranicza od dołu szerokość

adresu do 4 bitów. Stosowanie pól o mniejszej szerokości powoduje nieefektywną implementację. Możliwe jest jednak wkomponowanie pamięci w inne podstawowe bloki konstrukcyjne obecne w FPGA. Przykładem może być wykorzystywana w reduktorach modulo modyfikacja sumatora RCA nazywana w dalszej części pracy *sumatorem z wbudowaną pamięcią*. Modyfikacja ta pozwala na stosowanie pamięci ROM adresowanych słowem o szerokości 3 bitów.



Rysunek 3.5. Sumator z wbudowaną pamięcią.

Sumator z wbudowaną pamięcią (rys. 3.5) pozwala obliczyć sumę wektora binarnego i argumentu z wyjścia pamięci ROM. Przy jego konstrukcji wykorzystano fakt, że w sumatorze RCA w układach FPGA rodziny Spartan/Virtex bramka XOR wyznaczająca sumę modulo 2 operandów jest implementowana za pomocą 4–wejściowej tablicy LUT. Możliwe jest więc wbudowanie w tablicę LUT dodatkowej pamięci ROM, której wyjście jest podane na jedno z wejść bramki XOR. Dodatkowa pamięć może być adresowana słowem o szerokości 3 bitów. Obszar zajmowany przez tak zmodyfikowany sumator jest identyczny z obszarem prostego sumatora RCA.

W prezentowanych w pracy jednostkach arytmetycznych sumator z wbudowaną pamięcią jest wykorzystywany w dwóch przypadkach. W reduktorach modulo umożliwia on obliczenie sumy wektora binarnego i reszty modulo M_* dla pola o szerokości 3 bitów będącego adresem pamięci. Drugim zastosowaniem jest subtraktor warunkowy, w którym w pamięci ROM znajdują się odpowiednie krotności M_*, a pamięć jest adresowana sygnałami z komparatorów.

Subtraktor warunkowy

Wykorzystanie sumatora z pamięcią w subtraktorze warunkowym narzuca ograniczenia na zakres liczby wejściowej subtraktora. Ponieważ pamięć ROM wbudowana w sumator może mieć co najwy-

żej 3 linie adresowe, możliwe jest użycie co najwyżej 3 komparatorów. Większa liczba komparatorów wymusza stosowanie dodatkowych poziomów LUT. Trzy komparatory mogą zbadać jedynie, czy argument jest większy lub równy M_*, $2 \cdot M_*$ i $3 \cdot M_*$. Zakres argumentu subtraktora warunkowego jest zatem ograniczony do $[0, 4 \cdot M_*)$. Możliwe jest rozbudowanie układu w kierunku obsługi większego zakresu, spowoduje to jednak wzrost wprowadzanego opóźnienia i zajmowanego obszaru. Z tego powodu w prezentowanym rozwiązaniu ograniczono wartości argumentów komparatorów warunkowych do $4 \cdot M_* - 1$. Z warunku tego wynika długość kaskadady reduktorów modulo. Redukcja musi być przeprowadzana do momentu, w którym wartość wyjścia ostatniego reduktora w kaskadzie jest mniejsza od $4 \cdot M_*$.

Reduktor modulo

Zadaniem reduktora modulo jest zredukowanie wartości bezwzględnej liczby wejściowej U do liczby wyjściowej V, przy czym reszty modulo M_* dla słowa wejściowego i wyjściowego muszą być sobie równe. Idea działania reduktora modulo polega na podziale wektora reprezentującego daną wejściową na niewielkie pola bitowe, następnie obliczeniu wartości reszt modulo dla pól reprezentujących liczby większe od M_* i zsumowaniu otrzymanych reszt i pozostałych pól. Wejściem pojedynczego reduktora modulo jest liczba naturalna U. Na jej podstawie obliczana jest wartość wyjściowa V, przy czym $U > V$ oraz $U \equiv V \bmod M_*$. Liczby wejściowa i wyjściowa reprezentowane są przez wektory binarne $\boldsymbol{U}$ i $\boldsymbol{V}$. Przekształcenie wewnątrz reduktora modulo obejmuje trzy kroki: podział wektora wejściowego na pola bitowe, wyznaczanie reszt modulo dla części pól bitowych oraz obliczenie sumy reszt i pozostałych pól bitowych.

Struktura reduktora modulo jest jednoznacznie określona przez sposób podziału wektora wejściowego $\boldsymbol{U}$. Sposób podziału wektora $\boldsymbol{U}$ na pola jest wybierany podczas projektowania układu i opisany wektorem $\boldsymbol{d}$. Kolejne elementy d_i określają szerokość i–tego pola, analogicznie jak w przypadku wektorów $\boldsymbol{a}, \boldsymbol{b}$ (wzory (3.6)–(3.8), str. 89). Wartość liczby wyjściowej reduktora modulo wynosi

$$V = U(1) + \sum_{i>1} |U(i)|_{M_*} . \tag{3.17}$$

Ponieważ na wszystkich polach oprócz $u_{d_1-1:0}$ można zapisać wartości znacznie większe od M_*, wartość liczby wyjściowej V jako sumy reszt dla wydzielonych pól jest znacznie ograniczona w stosunku do U. Warunek $V < U$ jest spełniony, jeśli:

- wartości wszystkich pól bitowych, dla których wyznaczane są reszty, są większe od M_*, czyli

$$U(i) > |U(i)|_{M_*} \tag{3.18}$$

  dla $i > 1$,

- istnieje co najmniej jedno pole bitowe, dla którego obliczana jest reszta modulo M_*.

Użycie reduktora umożliwia zatem zredukowanie wartości sumy wytworzonej w bloku sumatora wstępnego do wartości umożliwiającej efektywną implementację subtraktora warunkowego. Jeśli zakres liczby wyjściowej pojedynczego reduktora jest nadal zbyt duży, stosuje się kaskadowe połączenie reduktorów.

Szerokości pól wydzielonych z $\boldsymbol{U}$ podlegają ograniczeniom wynikającym ze struktury elementów użytych do implementacji reduktora oraz z konieczności zapewnienia warunku $V < U$. Dla pól, dla których wyznaczane są reszty modulo M_* za pomocą pamięci ROM, minimalna szerokość wynosi 3 bity z powodu struktury elementów użytych do implementacji. Maksymalna szerokość pola zależy jedynie od dopuszczalnej złożoności układu – wymagana pojemność pamięci rośnie wykładniczo z szerokością pola. Zupełnie inne warunki ograniczają rozmiar pola $u_{d_1-1:0}$, dla którego nie jest obliczana reszta modulo M_*. Liczba bitów pola $u_{d_1-1:0}$ musi być tak dobrana, aby zapewnić zbieżność liczb wyjściowych kolejnych reduktorów modulo w kaskadzie do liczby mniejszej od $4 \cdot M_*$.

Minimalna szerokość pola $u_{d_1-1:0}$ wynika z konieczności spełnienia warunku (3.18). Minimalna szerokość pola, dla którego wyznaczana jest reszta modulo M_*, wynosi 3 bity. Spełnienie warunku (3.18) wymaga, aby reszta dla pola o minimalnej szerokości następującego bezpośrednio po polu $u_{d_1-1:0}$ była mniejsza od wartości zapisanej na tym polu. W skład pola, dla którego jest obliczana reszta, musi zatem wchodzić bit o wadze większej od M_*, czyli o indeksie $\geq m_* + 1$. Wynika stąd, że w polu $u_{d_1-1:0}$ musi znajdować się bit o indeksie $m_* - 2$, a więc minimalna szerokość pola $u_{d_1-1:0}$ wynosi $m_* - 1$ bitów.

Maksymalna szerokość pola $u_{d_1-1:0}$ jest określona przez maksymalną wartość liczby wyjściowej V. Wartość V jest określona jako suma pola $u_{d_1-1:0}$ i reszt dla pozostałych pól. Kaskadowe połączenie reduktorów modulo ma za zadanie zredukowanie wartości liczby wejściowej do poziomu co najwyżej $4 \cdot M_* - 1$. Liczby wyjściowe kolejnych reduktorów w kaskadzie tworzą ciąg malejący, można więc założyć, że wejściem ostatniego reduktora jest wektor o szerokości umożliwiającej podział na pole $u_{d_1-1:0}$ i jedno pole dodatkowe. Wartość liczby wyjściowej ostatniego reduktora jest zatem ograni-

czona od góry sumą $u_{d_1-1:0}$ i $M_* - 1$. Wynika stąd, że wartością maksymalną pola $u_{d_1-1:0}$ musi być liczba mniejsza od $3 \cdot M_*$. Maksymalna szerokość pola $u_{d_1-1:0}$ wynosi więc $m_* + 1$ bitów.

Mając dane ograniczenia szerokości pól wydzielonych z wektora wejściowego reduktora modulo, można dokładnie określić możliwości redukcji pojedynczego reduktora i liczbę reduktorów w kaskadzie niezbędną do zredukowania liczby o zadanej szerokości.

Lemat 3.1.2. *Niech K oznacza minimalną szerokość pól wydzielanych z wyższej części k–bitowego wektora wejściowego reduktora modulo, a $m_{max} = m_* + 1$ i $m_{min} = m_* - 1$ maksymalną i minimalną szerokość pola zawierającego bit o indeksie 0. Maksymalna szerokość wektora wyjściowego reduktora modulo jest nie większa niż $m_* + \left\lceil \log_2 \left(\left\lfloor \frac{k-m_{min}}{K} \right\rfloor + 4 \right) \right\rceil$.*

Dowód. Zgodnie z (3.17) wartość liczby wyjściowej reduktora modulo jest równa sumie reszt dla pól wydzielonych z bardziej znaczącej części i liczby zapisanej na mniej znaczącej części wektora wejściowego reduktora modulo. Maksymalna wartość sumy reszt wystąpi dla maksymalnej liczby tych reszt, czyli gdy bardziej znacząca część wektora będzie podzielona na pola o minimalnej szerokości. Maksymalna szerokość sumy jest ograniczona od góry przez sumę reszt dla pól wydzielonych z $(k - m_{min})$ bardziej znaczących bitów i wektora m_{max} bitowego. Największa możliwa liczba reszt modulo M_* wynosi więc $\left\lfloor \frac{k-m_{min}}{K} \right\rfloor$. Maksymalna wartość liczby wyjściowej reduktora jest więc mniejsza niż

$$\left\lfloor \frac{k - m_{min}}{K} \right\rfloor \cdot (M_* - 1) + 2^{m_{max}} - 1. \tag{3.19}$$

Ponieważ $2^{m_{max}} - 1 = 2^{m_*+1} - 1 < 4 \cdot (M_* - 1)$ dla $M_* > 2^{m_*-1}$, a więc

$$\left\lfloor \frac{k - m_{min}}{K} \right\rfloor \cdot (M_* - 1) + 2^{m_{max}} - 1 < \left(\left\lfloor \frac{k - m_{min}}{K} \right\rfloor + 4 \right) \cdot (M_* - 1) \tag{3.20}$$

Szerokość słowa wyjściowego reduktora modulo jest więc ograniczona od góry przez

$$\left\lfloor \log_2 \left(\left(\left\lfloor \frac{k - m_{min}}{K} \right\rfloor + 4 \right) \cdot (M_* - 1) \right) \right\rfloor + 1 = \left\lfloor \log_2 (M_* - 1) + \log_2 \left(\left\lfloor \frac{k - m_{min}}{K} \right\rfloor + 4 \right) \right\rfloor + 1.$$

Ponieważ $\lfloor \log_2 (M_* - 1) \rfloor + 1 \leq m_*$, to maksymalna szerokość słowa wyjściowego jest ograniczona z góry przez

$$m_* + \left\lceil \log_2 \left(\left\lfloor \frac{k - m_{min}}{K} \right\rfloor + 4 \right) \right\rceil. \tag{3.21}$$

□

Z lematu 3.1.2 wynika, że słowo wyjściowe reduktora modulo jest szersze od m_* o co najwyżej $\left\lceil \log_2 \left(\left\lfloor \frac{k-m_{min}}{K} \right\rfloor + 4 \right) \right\rceil$ bitów. Niestety, nie gwarantuje to uzyskania wektora szerszego o co najwyżej

dwa bity od m_*. Należy jednak zauważyć, że ograniczenia przyjęte w lemacie 3.1.2 są dość niedokładne. Jedynym pewnym wnioskiem wynikającym z lem. 3.1.2 jest to, że szerokości wektorów wyjściowych reduktorów modulo zawsze zależą od logarytmu liczby nadmiarowych bitów. Kaskada reduktorów modulo umożliwia więc redukcję szerokich wektorów przy niewielkiej liczbie poziomów. Dokładniejszą informację o liczbie poziomów kaskady w zależności od liczby bitów nadmiarowych przy przyjętych ograniczeniach na strukturę reduktora modulo pozwala określić lemat 3.1.3.

Lemat 3.1.3. *Niech $k > m_* + 4$ oznacza szerokość wektora wejściowego reduktora modulo, $m_{max} = m_* + 1$ i $m_{min} = m_* - 1$ oznaczają maksymalną i minimalną szerokość pola zawierającego bit o indeksie 0, a $K_{min} = 3$ i $K_{max} \in [3, k - m_* + 1]$ będą minimalną i maksymalną szerokością pól wydzielonych z pozostałej części wektora wejściowego. Istnieje taka konfiguracja reduktora modulo, dla której szerokość wektora wyjściowego jest mniejsza lub równa $m_* + \lceil \log_2(k - m_* + 1 + K_{max}) - \log_2(K_{max}) \rceil$.*

Dowód. Dla zadanych ograniczeń na k, K_{max} i K_{min}, można zawsze podzielić $k - m_{min}$ bitów na $\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil$ pól zawierających co najmniej K_{min} i co najwyżej K_{max} bitów każde. Wynikająca stąd suma reszt i mniej znaczącej części wektora wejściowego jest więc mniejsza niż

$$\left\lceil \frac{k - m_* + 1}{K_{max}} \right\rceil \cdot (M_* - 1) + 2^{m_*-1} - 1.$$

Ponieważ $2^{m_*-1} - 1 \leq M_* - 1$, ograniczenie to można zapisać w postaci

$$\left(\left\lceil \frac{k - m_* + 1}{K_{max}} \right\rceil + 1 \right) \cdot (M_* - 1).$$

Szerokość słowa wyjściowego reduktora jest więc mniejsza lub równa

$$\left\lfloor \log_2 \left(\left(\left\lceil \frac{k - m_* + 1}{K_{max}} \right\rceil + 1 \right) \cdot (M_* - 1) \right) \right\rfloor + 1 =$$

$$= \left\lfloor \log_2 \left(\left\lceil \frac{k - m_* + 1 + K_{max}}{K_{max}} \right\rceil \right) + \log_2(M_* - 1) \right\rfloor + 1.$$

Następnie, ponieważ $\lceil \log_2 (\lceil \alpha \rceil) \rceil = \lceil \log_2 (\alpha) \rceil$ dla $\alpha > 1$ (dowód w [GKP01], str. 92), górnym ograniczeniem szerokości jest $\lceil \log_2(k - m_* + 1 + K_{max}) - \log_2(K_{max}) \rceil + \lfloor \log_2(M_* - 1) \rfloor + 1$. Ponieważ $\lfloor \log_2(M_* - 1) \rfloor + 1 \leq m_*$, to liczba bitów wyjściowych reduktora modulo jest mniejsza lub równa $m_* + \lceil \log_2(k - m_* + 1 + K_{max}) - \log_2(K_{max}) \rceil$. □

Z lematu 3.1.3 wynikają dwa istotne wnioski:

- szybkość zbieżności liczb wyjściowych kolejnych reduktorów w kaskadzie zależy wyłącznie od maksymalnej szerokości pól, dla których są obliczane reszty modulo,

- istnieje taka szerokość k wektora wejściowego, dla której szerokość wektora wyjściowego reduktora modulo jest co najwyżej $m_* + 2$.

Możliwe jest więc zbudowanie kaskady reduktorów modulo redukującej dowolną liczbę do liczby mniejszej od $4 \cdot M_*$. Przykładowe wzory opisujące szerokości wektorów wyjściowych kolejnych trzech poziomów reduktorów są następujące:

$$\text{Poziom } 1: \quad m_* + \lceil \log_2(k - m_* + C_1) + C_2 \rceil ,$$
$$\text{Poziom } 2: \quad m_* + \lceil \log_2 (\log_2(k - m_* + C_1) + C_2 + C_1) + C_2 \rceil ,$$
$$\text{Poziom } 3: \quad m_* + \lceil \log_2 (\log_2 (\log_2(k - m_* + C_1) + C_2 + C_1) + C_2 + C_1) + C_2 \rceil ,$$

gdzie $C_1 = 1 + K_{max}$, $C_2 = 1 - \log_2(K_{max})$. W tabeli 3.1.3 umieszczono szerokości wektorów, które można zredukować do słów $(m_* + 2)$–bitowych za pomocą kaskady reduktorów o zadanym poziomie i maksymalnej szerokości pola równej 3 lub 4 bity. Wynika z niej, że dla praktycznie stosowanych układów w zupełności wystarcza kaskadowe połączenie trzech reduktorów modulo, a w wielu przypadkach nawet dwóch.

Tabela 3.1. Maksymalne szerokości wektorów wejściowych redukowalnych do wektora $(m_* + 2)$–bitowego w zależności od liczby poziomów kaskady reduktorów i szerokości pól bitowych.

K_{max}	Liczba poziomów kaskady		
	1	2	3
3	$m_* + 8$	$m_* + 3 \cdot 2^7 - 4$	$m_* + 3 \cdot 2^{3 \cdot 2^7 - 5} - 4$
4	$m_* + 11$	$m_* + 2^{13} - 5$	$m_* + 2^{2^{13} - 3} - 5$

Algorytm konstrukcji reduktora modulo

Struktura pojedynczego reduktora modulo jest określona jednoznacznie przez sposób podziału wektora wejściowego $\mathbf{U}$. Poniżej przedstawiono algorytm generujący strukturę reduktora na podstawie wektora $\mathbf{U}$ reprezentującego liczbę wejściową i wektora $\mathbf{d}$ opisującego sposób podziału $\mathbf{U}$ na pola. Algorytm konstrukcji reduktora modulo składa się z dwóch etapów.

Etap pierwszy obejmuje podział wektora wejściowego $\boldsymbol{U}$ na pola określone przez wektor $\boldsymbol{d}$. Dla pól, które nie zawierają bitu o indeksie 0, a ich szerokość jest większa niż 3 bity, są budowane pamięci ROM wyznaczające reszty modulo M_*. Jeśli istnieje co najmniej jedno pole o szerokości 3 bitów, na etapie pierwszym konstruowany jest także sumator z wbudowaną pamięcią dodający resztę dla tego pola do wartości zapisanej na polu $u_{d_i - 1:0}$.

Algorytm 5 Algorytm tworzenia wektora $\mathbf{\Psi}_0$

Dane wejściowe: $\boldsymbol{U}, \boldsymbol{d}$

1: Stwórz wektor $\mathbf{\Psi}_0$ zawierający pole $u_{d_1-1:0}$
2: **for** $j = 1$ **to** $|\boldsymbol{d}|$ **do**
3: **if** $d_j > 3$ **then**
4: Dopisz na ostatnią pozycję $\mathbf{\Psi}_0$ wektor reszty dla pola opisanego przez d_j
5: Ustaw typ dopisanego elementu na CPA
6: **end if**
7: **end for**
8: **for** $j = 1$ **to** $|\boldsymbol{d}|$ **do**
9: **if** $d_j = 3$ **then**
10: Dopisz na ostatnią pozycję $\mathbf{\Psi}_0$ pole 3–bitowe opisane przez d_j
11: Ustaw typ dopisanego elementu na CPY_3
12: **end if**
13: **end for**
14: **if** ostatni element $\mathbf{\Psi}_0$ jest typu CPY_3 **then**
15: Zastąp pierwszy element $\mathbf{\Psi}_0$ sumą pola $u_{d_1-1:0}$ i reszty dla ostatniego elementu $\mathbf{\Psi}_0$
16: Usuń ostatni element $\mathbf{\Psi}_0$
17: **end if**
18: **return** $\mathbf{\Psi}_0$

Etap drugi algorytmu obejmuje konstrukcję wielooperandowego sumatora. W pierwszej kolejności dodawane są reszty dla pól 3–bitowych do wektorów zawierających bity o indeksie 0 (np. reszt obliczonych dla pól o szerokości większej niż 3 bity). Dodawanie reszt dla pól 3–bitowych jest realizowane przez sumatory z wbudowaną pamięcią. Jeśli wszystkie pola 3–bitowe zostały wykorzystane, sumowane są wszystkie powstałe wektory.

Sumator wielooperandowy jest układem wielopoziomowym. Poziom i stanowią wektory sum częściowych opisane elementami ψ_j^i tworzącymi wektor $\mathbf{\Psi}_i$. Każdy wektor sumy częściowej jest zdefiniowany przez indeks najmłodszego bitu, maksymalną wartość reprezentowanej liczby, indeksy operandów i rodzaj wykonywanej operacji. Struktura sumatora wielooperandowego jest jednoznacznie określona przez przekształcenia wektorów poziomu i na wektory poziomu $i + 1$.

Wektory na każdym poziomie można podzielić na dwie kategorie: pola 3–bitowe i wektory zawierające bit o indeksie 0. Wektor zawierający bit o indeksie 0 jest resztą modulo M_*, polem $u_{d_1-1:0}$ lub sumą, jego szerokość wynosi więc co najmniej $m_* - 1$ bitów. Wektory zawierające bit o indeksie 0 mogą być dodawane do siebie lub do reszt dla pól 3–bitowych. W algorytmie wektorom zawierającym bit o indeksie 0 przyporządkowany jest typ oznaczony jako CPA.

Dla pól 3–bitowych można jedynie obliczyć reszty modulo i dodać do wektora typu CPA za pomocą sumatora z pamięcią. Polom 3–bitowym przyporządkowany jest typ CPY_3. Aby umożliwić szybkie wyeliminowanie wszystkich pól typu CPY_3, należy pola te wykorzystywać w pierwszej kolejności.

Kompletny algorytm tworzenia reduktora modulo konstruuje kolejne poziomy sumatorów na podstawie opisu wektorów wyjściowych poprzedniego poziomu. Opis struktury reduktora jest określony przez przekształcenia wektorów $\mathbf{\Psi}_i$ w wektory $\mathbf{\Psi}_{i+1}$. Pierwszym krokiem algorytmu jest skonstruowanie wektora $\mathbf{\Psi}_0$ zawierającego pole $u_{d_1-1:0}$, reszty modulo M_* dla pól o szerokości większej od 3 bitów i pola 3–bitowe. Elementy wektora $\mathbf{\Psi}_0$ są posortowane względem typu: na początku znajdują się elementy typu CPA, potem elementy CPY_3. Wektor $\mathbf{\Psi}_0$ tworzony jest według alg. 5.

Algorytm tworzenia wektora $\mathbf{\Psi}_0$ (alg. 5) składa się z czterech kroków. W pierwszym z nich (linia 1) na początek wektora $\mathbf{\Psi}_0$ dopisywane jest pole $u_{d_1-1:0}$. Następny krok (wiersze 2-7) dopisują do $\mathbf{\Psi}_0$ reszty modulo M_* dla pól o szerokości większej od 3 bitów. Trzeci etap (linie 8-13) dopisują na ostatnie pozycje $\mathbf{\Psi}_0$ wszystkie pola o szerokości 3 bitów. W ostatnim kroku algorytmu (linie 14-17) pierwszy element $\mathbf{\Psi}_0$ jest zastępowany sumą pola $u_{d_1-1:0}$ i reszty modulo M_* dla 3–bitowego pola.

Dzięki oddzieleniu elementów typu CPA od elementów typu CPY_3 możliwe jest znaczne uproszczenie i przyspieszenie algorytmu tworzenia reduktora modulo poprzez uniknięcie konieczności wielokrotnego przeszukiwania $\mathbf{\Psi}_0$ oraz wszystkich kolejnych wektorów $\mathbf{\Psi}_i$. Jedynymi elementami składowymi reduktora modulo są pamięci ROM obliczające reszty modulo M_* dla pól szerszych niż 3 bity, sumatory CPA obliczające sumy pośrednie dla elementów typu CPA oraz sumatory z wbudowaną pamięcią obliczające sumy pośrednie dla elementów CPA i reszt modulo M_* dla pól 3–bitowych. Ponieważ wszystkie reszty dla pól szerszych niż trzy bity są wyznaczane w procesie generowania wektora $\mathbf{\Psi}_0$, operacje na kolejnych poziomach ograniczają się do dodawania.

Uporządkowanie elementów wektora $\mathbf{\Psi}_0$ gwarantuje, że pary zawierające skrajne elementy $\mathbf{\Psi}_0$ (tj. pierwszy i ostatni, drugi i przedostatni itd.) będą zawierały element typu CPA i element typu CPY_3 lub dwa elementy tego samego typu. Dodatkowo jeśli wektor $\mathbf{\Psi}_0$ zawiera elementy obu typów, naj-

pierw zostaną utworzone pary zawierające elementy różnych typów, a następnie pary elementów typu stanowiącego większość. Ponieważ wynikiem sumowania elementu CPA z dowolnym jest element CPA, a elementy CPY_3 są przekazywane do następnego poziomu, najpierw zostaną obliczone wszystkie możliwe sumy, a następnie niewykorzystane elementy typu CPY_3 będą przekazane do następnego stopnia. Dzięki temu wektory kolejnych poziomów także będą uporządkowane.

Algorytm 6 Algorytm tworzenia reduktora modulo

Dane wejściowe: $\boldsymbol{U}, \boldsymbol{d}$

1: Konstruuj wektor $\mathbf{\Psi}_0$ według algorytmu 5
2: $i = 0$ {indeks poziomu oraz kolejnych zbiorów}
3: **while** $|\mathbf{\Psi}_i| > 1$ **do**
4: **for** $j = 0$ **to** $\left\lfloor \frac{|\mathbf{\Psi}_i|}{2} \right\rfloor$ **do**
5: **if** $\mathtt{typ}(\psi^i_j) = CPA$ **and** $\mathtt{typ}(\psi^i_{|\mathbf{\Psi}_i|-j}) = CPY\_3$ **then**
6: Dopisz do wektora $\mathbf{\Psi}_{i+1}$ wektor sumy $\psi^i_j + \psi^i_{|\mathbf{\Psi}_i|-j}$ obliczonej z użyciem sumatora z redukcją
7: Ustaw typ dopisanego elementu na CPA
8: **else if** $\mathtt{typ}(\psi^i_j) = CPA$ **and** $\mathtt{typ}(\psi^i_{|\mathbf{\Psi}_i|-j}) = CPA$ **then**
9: Dopisz do wektora $\mathbf{\Psi}_{i+1}$ wektor sumy $\psi^i_j + \psi^i_{|\mathbf{\Psi}_i|-j}$ obliczonej z użyciem sumatora RCA
10: Ustaw typ dopisanego elementu na CPA
11: **else if** $\mathtt{typ}(\psi^i_j) = CPY\_3$ **then**
12: Dopisz do zbioru $\mathbf{\Psi}_{i+1}$ wektory ψ^i_j i $\psi^i_{|\mathbf{\Psi}_i|-j}$
13: **end if**
14: **end for**
15: $i = i + 1$
16: **end while**
17: **return** $\mathbf{\Psi} = (\mathbf{\Psi}_0, \mathbf{\Psi}_1, \ldots, \mathbf{\Psi}_i)$

Po utworzeniu $\mathbf{\Psi}_0$ rozpoczyna się iteracyjna część algorytmu. W każdej iteracji tworzony jest jeden poziom kaskady sumatorów. W pojedynczej iteracji wybierane są pary wektorów i dla każdej z nich konstruowany jest odpowiedni sumator dodający wytypowane wektory. Wektory, dla których nie można zbudować sumatora, są przekazywane do następnego poziomu. Algorytm jest kontynuowany do momentu wytworzenia sumy wszystkich elementów $\mathbf{\Psi}_0$.

Właściwy algorytm tworzenia reduktora modulo (alg. 6) sprowadza się do iteracyjnego generowania kolejnych wektorów $\mathbf{\Psi}_{i+1}$ na podstawie $\mathbf{\Psi}_i$ (wiersze 3-16). Generowanie kolejnego wektora $\mathbf{\Psi}_{i+1}$ polega na wyborze par skrajnych elementów $\mathbf{\Psi}_i$ (linia 4), po czym w zależności od typu elementów wybranej pary (linie 5, 8 i 11) tworzony jest następny element $\mathbf{\Psi}_{i+1}$. Jeżeli parę stanowią elementy różnych typów lub dwa elementy CPA, z ich sumy tworzony jest jeden element CPA i dopisywany na ostatnią pozycję $\mathbf{\Psi}_{i+1}$. Para złożona z dwóch elementów CPY_3 jest w całości przekazywana do następnego poziomu. Warunkiem zakończenia iteracji jest utworzenie zbioru zawierającego jeden element (linia 3). Warunek ten zostanie spełniony po co najwyżej $|\mathbf{\Psi}_0| - 1$ iteracjach, ponieważ w wektorze $\mathbf{\Psi}_0$ znajduje się co najmniej jeden element typu CPA (pole $u_{d_1-1:0}$).

3.1.4 Obszar i opóźnienie jednostek arytmetycznych

Do najistotniejszych parametrów układów cyfrowych zalicza się koszt wyprodukowania układu i jego wydajność, charakteryzowane za pomocą zajmowanego obszaru A i wprowadzanego opóźnienia T. Często stosowanymi miarami jakości układów cyfrowych są także iloczyny AT i AT^2. W typowych układach FPGA zajmowany obszar opisany jest liczbą wykorzystanych tablic LUT, ponieważ wraz z dodatkowymi elementami (przerzutnik, multipleksery itp.) są one podstawowym blokiem składowym większych układów. Wprowadzane opóźnienie jest oszacowane jako liczba poziomów tablic LUT w ścieżce krytycznej.

W oszacowaniu opóźnienia prezentowanych układów nie uwzględniono czasów wprowadzanych przez układy propagacji przeniesień, dodatkowe elementy (multipleksery, wbudowane bramki XOR itp.) oraz połączenia pomiędzy poszczególnymi elementami. Głównym powodem takiej decyzji jest trudność dokładnego oszacowania długości połączeń, która jest zależna od struktury układu oraz stosowanego narzędzia syntezy. Pominięcie opóźnień związanych z układami propagacji przeniesień wynika z ich niewielkiego wpływu na szybkość układu dla modułów o szerokości nie przekraczającej kilkunastu bitów. Poza tym głównym celem oszacowań długości ścieżki krytycznej jest porównanie układów w ramach jednego modułu, dla których szerokości sumatorów i układów mnożących są podobne. Dokładne parametry jednostki są określane dopiero po przeprowadzeniu pełnej implementacji w wybranym układzie FPGA z użyciem konkretnego narzędzia projektowego. Pomimo orientacyjnego charakteru przyjętych oszacowań pozwalają one na porównanie układów i wytypowanie spośród nich niewielkiego podzbioru, w którym znajduje się jednostka o akceptowanych parametrach.

Podstawowymi elementami składowymi przedstawionych jednostek arytmetycznych są sumatory CPA o strukturze RCA, pamięci ROM oraz układy mnożące. Koszt k–bitowego sumatora CPA jest opisany jako

$$\begin{aligned} A_{CPA}(k) &= k \cdot A_L \\ T_{CPA}(k) &= T_L \end{aligned}, \tag{3.22}$$

gdzie obszar zajęty przez jedną tablicę LUT wraz z otoczeniem oznaczono jako A_L, a opóźnienie jako T_L.

Pamięć ROM jest konstruowana z pojedynczych, 4–wejściowych tablic LUT o pojemności 16 bitów każda. Dekoder adresu pamięci jest także konstruowany z użyciem tablic LUT. Parametry pamięci ROM zawierającej k słów m–bitowych mogą być więc oszacowane jako

$$\begin{aligned} A_{ROM}(k,m) &= \left\lceil \tfrac{m \cdot k}{16} \right\rceil \cdot A_L \\ T_{ROM}(k,m) &= \lceil \log_4(\lceil \log_2(k) \rceil) \rceil \cdot T_L \end{aligned}. \tag{3.23}$$

W rzeczywistości wzór (3.23) podaje tylko górne oszacowanie parametrów AT pamięci ROM, gdyż w zależności od zawartości ROM narzędzia syntezy mogą dokonać minimalizacji układu. Dla przykładu dokumentacja kompilatora *xst* podaje, że pamięć ROM jest implementowana jako optymalny układ złożony z tablic LUT, multiplekserów i bramek XOR.

Dla układów mnożących należy rozróżnić kilka przypadków. Najprostszym jest mnożenie $2 \cdot k$ bity o koszcie

$$\begin{aligned} A_{2 \cdot k}(k) &= (k+2) \cdot A_L \\ T_{2 \cdot k}(k) &= T_L \end{aligned}. \tag{3.24}$$

Dla mnożenia $3 \cdot k$ bity

$$\begin{aligned} A_{3 \cdot k}(k) &= 2 \cdot (k+2) \cdot A_L \\ T_{3 \cdot k}(k) &= 2 \cdot T_L \end{aligned}. \tag{3.25}$$

Układy mnożące $m \cdot k$ bitów dla dowolnego m są budowane jako odpowiednie kombinacje powyższych, uzupełnione o dodatkowe sumatory.

Prezentowana na rys. 3.1 struktura jednostek arytmetycznych modulo jest kaskadowym połączeniem poszczególnych bloków. Obszar i opóźnienie kompletnej jednostki jest więc równe sumie obszarów i opóźnień układów składowych. Podanie dokładnych charakterystyk AT jest możliwe wyłącznie dla konkretnej struktury jednostki arytmetycznej po jej pełnej implementacji. Można jednak określić granice parametrów AT dla jednostek konstruowanych według struktury ramowej z rys. 3.1. Podstawowym celem oszacowań przedstawionych poniżej jest analiza parametrów AT w funkcji szerokości

operandów m_*. Pozostałe wielkości występujące we wzorach opisujących obszar i opóźnienie są traktowane jak stałe wybierane przed projektowaniem struktury samej jednostki. Dobór wartości tych stałych może być jednak przeprowadzony w oparciu o analizę ich wpływu na parametry jednostek określone poniższymi zależnościami.

Charakterystyki AT układu wytwarzania iloczynów częściowych

Układ wytwarzania iloczynów częściowych (UWIC) jest pierwszym blokiem w strukturze zaprezentowanej na rys. 3.1. Może on zawierać zarówno pamięci ROM, jak i układy mnożące $2 \cdot k$ bity. W zależności od przyjętych ograniczeń na rodzaj i parametry zastosowanych elementów, charakterystyki AT układu wytwarzania iloczynów częściowych mogą zmieniać się w szerokich granicach.

Lemat 3.1.4. *Niech $m_* > 3$ oznacza liczbę bitów wektora wejściowego. Istnieje układ wytwarzania iloczynów częściowych o strukturze opisanej w rozdz. 3.1.1 o obszarze nie większym niż $\left(\left\lceil \frac{m_*^2}{2} \right\rceil + m_*\right) \cdot A_L$ i wprowadzanym opóźnieniu nie większym niż T_L.*

Dowód. Załóżmy, że iloczyny częściowe są wytwarzane wyłącznie za pomocą układów mnożących $2 \times m_*$ bitów. W tym przypadku tylko jeden z wektorów wejściowych musi być podzielony na pola o szerokości dwóch bitów. Jeśli m_* jest nieparzyste, można wektor dzielony na pola rozszerzyć o 1 bit. Należy zatem użyć $\left\lceil \frac{m_*}{2} \right\rceil$ układów mnożących, z których każdy zajmuje obszar $(m_* + 1) \cdot A_L$, zgodnie z wzorem (3.24). Obszar całego układu wynosi więc $\left\lceil \frac{m_*}{2} \right\rceil \cdot (m_* + 1) \cdot A_L$. Wszystkie układy mnożące pracują równolegle, opóźnienie jest więc równe opóźnieniu układu najwolniejszego, czyli T_L zgodnie z (3.24). □

Z lematu 3.1.4 wynika, że jeśli do wytwarzania iloczynów częściowych użyte zostaną wyłącznie układy mnożące $2 \cdot m_*$ bitów, obszar UWIC zależy od kwadratu szerokości wektorów wejściowych. Ograniczenie elementów UWIC do układów mnożących zwiększa jednak szerokość sumy iloczynów częściowych, co powoduje pogorszenie parametrów AT pozostałych bloków jednostki z rys. 3.1. Z tego powodu układ wytwarzania iloczynów częściowych może także zawierać pamięci ROM obliczające reszty dla wszystkich, bądź tylko wybranych, iloczynów częściowych. W zależności od pojemności pamięci, obszar i opóźnienie UWIC mogą zmieniać się w szerokich granicach. Skrajnym przypadkiem jest rozwiązanie zawierające pojedynczą pamięć ROM adresowaną złożeniem wektorów reprezentujących operandy. Zgodnie z (3.23), obszar takiej pamięci jest ograniczony od góry przez $2^{2 \cdot m_* - 4} \cdot m_* \cdot A_L$, a opóźnienie można oszacować jako $\lceil \log_4 (2 \cdot m_*) \rceil \cdot T_L$.

Jeśli maksymalny rozmiar słowa adresowego pamięci ROM zostanie ograniczony, zmniejszy się także obszar układu wytwarzania iloczynów częściowych. Załóżmy, że iloczyny częściowe są obliczane za pomocą wyłącznie pamięci ROM. Dla maksymalnej szerokości adresu ROM równej 4 bity, należy podzielić oba wektory wejściowe na pola 2–bitowe i obliczyć iloczyny częściowe dla $\left(\frac{m_*}{2}\right)^2$ par tych pól. Obszar pojedynczej pamięci wynosi $m_* \cdot A_L$, a więc obszar całego UWIC jest równy $\frac{m_*^3}{4} \cdot A_L$. Wprowadzane opóźnienie jest w tym przypadku równe T_L.

Dla innych ograniczeń na rozmiar pojedynczej pamięci ROM oszacowanie obszaru całego układu może być trudne. Jeśli wszystkie iloczyny częściowe mają być wytwarzane za pomocą pamięci ROM, należy tak wybrać podziały na pola wektorów wejściowych, aby suma bitów dla każdej kombinacji tych pól nie przekroczyła maksymalnej szerokości adresu pamięci. Dla maksymalnej szerokości słowa adresowego ROM równej 6 bitów, jeden z wektorów może być podzielony na pola 2–bitowe, a drugi 4-bitowe, lub oba wektory należy podzielić na pola 2– lub 3–bitowe. Inne podziały spowodują powstanie niedopuszczalnych szerokości słów adresowych pamięci ROM. Liczba wytworzonych iloczynów częściowych należy więc do przedziału $\left[\frac{m_*^2}{9}, \frac{m_*^2}{4}\right]$. Obszar pamięci wytwarzającej pojedynczy iloczyn częściowy jest ograniczony z góry przez $\left\lceil\frac{2^6 \cdot m_*}{16}\right\rceil$. Problem dodatkowo się komplikuje, jeśli w tym samym układzie można stosować pamięci o różnym rozmiarze i układy mnożące. Możliwe jest jednak podanie górnego ograniczenia obszaru UWIC w zależności od szerokości adresu pamięci ROM. Poniżej zostanie wykazane, że wzrost pojemności pamięci stosowanych w UWIC powoduje wzrost obszar obszaru tego bloku.

Lemat 3.1.5. *Niech układ wytwarzania iloczynów częściowych o strukturze opisanej w rozdz. 3.1.1 zawiera wyłącznie pamięci ROM. Jeśli szerokość $a_{j'}$ jednego z pól wydzielonych z wektora wejściowego UWIC zostanie zwiększona kosztem szerokości a_j pola szerszego, to górne ograniczenie obszaru UWIC zmniejszy się, gdy $a_{j'} < a_j - 1$, i pozostanie bez zmian dla $a_{j'} = a_j - 1$.*

Dowód. Adres pamięci wytwarzającej iloczyn częściowy σ_{jk} jest złożeniem pola o szerokości a_j i pola o szerokości b_k. Górne ograniczenie obszaru tej pamięci jest określone formułą (3.23) jako $\frac{m_* \cdot A_L}{16} \cdot 2^{a_j + b_k}$. Załóżmy, że szerokość $a_{j'}$ jednego z pól została zwiększona o 1 bit kosztem szerokości a_j, przy czym $a_j > a_{j'}$. Górne ograniczenie obszaru pamięci ROM, których adres zawiera pole o zmniejszonej szerokości, zmaleje o połowę. Górne ograniczenie obszaru pamięci ROM, których adres zawiera pole o zwiększonej szerokości, wzrośnie dwukrotnie. Adresy pamięci o zmienionej pojemności są złożeniem jednego z pól o szerokościach b_k i pola o szerokości a_j lub $a_{j'}$. Dla pamięci,

których adres nie zawiera pola o zmienionej szerokości, obszar pozostanie taki sam. Niech A_P oznacza sumę obszaru pamięci o nie zmienionych parametrach. Sumaryczny obszar wszystkich pamięci przez zmianą szerokości pól wynosi

$$A = A_P + \frac{m_* \cdot A_L}{16} \left(\sum_k 2^{a_j + b_k} + \sum_k 2^{a_{j'} + b_k} \right), \tag{3.26}$$

a po przesunięciu bitów

$$A' = A_P + \frac{m_* \cdot A_L}{16} \left(\sum_k 2^{a_j - 1 + b_k} + \sum_k 2^{a_{j'} + 1 + b_k} \right). \tag{3.27}$$

Zmiana obszaru wynikająca z przesunięcia bitów zależy więc od różnicy

$$A - A' = \frac{m_* \cdot A_L}{16} \cdot \left\{ \left(\sum_k 2^{a_j + b_k} + \sum_k 2^{a_{j'} + b_k} \right) - \left(\sum_k 2^{a_j - 1 + b_k} + \sum_k 2^{a_{j'} + 1 + b_k} \right) \right\}, \tag{3.28}$$

którą można uprościć do postaci

$$A - A' = C_A \cdot \left((2^{a_j} + 2^{a_{j'}}) - \left(2^{a_j - 1} + 2^{a_{j'} + 1}\right) \right) = C_A \cdot 2^{a_{j'}} \cdot \left(2^{a_j - a_{j'}} - 2^{a_j - a_{j'} - 1} - 1\right)$$

dla $C_A = \frac{m_* \cdot A_L}{16} \cdot \sum_k 2^{b_k}$. Wartość tej różnicy jest dodatnia dla $a_j - a_{j'} > 1$. Wynika stąd, że maksymalny sumaryczny obszar pamięci po przesunięciu bitu do pola o mniejszej szerokości ulegnie zmniejszeniu, gdy $a_j - a_{j'} > 1$, i pozostanie bez zmian dla $a_j - a_{j'} = 1$. □

Z lematu 3.1.5 wynika, że zwiększanie szerokości szerokich pól kosztem pól węższych powoduje wzrost maksymalnego obszaru UWIC. Należy jeszcze przeanalizować sytuację, w której zmiana rozmiaru pól jest wynikiem zmiany ich liczby, tj. dodania lub usunięcia pola.

Lemat 3.1.6. *Niech układ wytwarzania iloczynów częściowych o strukturze opisanej w rozdz. 3.1.1 zawiera wyłącznie pamięci ROM. Wprowadzenie do jednego z wektorów wejściowych UWIC 2–bitowego pola kosztem szerokości pozostałych pól tego wektora powoduje ograniczenie obszaru UWIC.*

Dowód. Dwubitowe pole można utworzyć zmniejszając szerokość jednego z pól o dwa bity lub dwóch pól o jeden bit. Niech $a_j \geq 4$ oznacza szerokość pola, którego szerokość jest zmniejszana o 2 bity, $a_{j'} \geq 3$ i $a_{j''} \geq 3$ szerokości pól zmniejszanych o 1 bit, przy czym $a_{j'} \geq a_{j''}$, a A_P sumę obszaru pamięci, których adresy nie zmieniają się po dodaniu nowego pola. Jeśli nowe pole powstaje wskutek zmniejszenia a_j o 2 bity, to obszary przed i po dodaniu nowego pola wynoszą

$$A = A_P + \frac{m_* \cdot A_L}{16} \cdot \left(\sum_k 2^{a_j + b_k} \right) \tag{3.29}$$

i

$$A' = A_P + \frac{m_* \cdot A_L}{16} \cdot \left(\sum_k 2^{a_j - 2 + b_k} + \sum_k 2^{2 + b_k} \right), \tag{3.30}$$

a ich różnica jest równa

$$A - A' = C_A \cdot \left(\frac{3}{4} \cdot 2^{a_j} - 4 \right), \tag{3.31}$$

gdzie $C_A = \frac{m_* \cdot A_L}{16} \cdot \sum_k 2^{b_k}$. Dla $a_j \geq 4$ wartość dana przez (3.31) jest zawsze dodatnia, co oznacza, że wprowadzając dodatkowe, 2–bitowe pole poprzez zmniejszenie szerokości pola o szerokości $a_j \geq 4$ zawsze uzyskuje się redukcję maksymalnego obszaru UWIC.

Jeśli nowe pole powstaje wskutek zmniejszenia $a_{j'} \geq 3$ i $a_{j''} \geq 3$ o jeden bit, to maksymalny sumaryczny obszar pamięci przed zmniejszeniem $a_{j'}$ i $a_{j''}$ wynosi

$$A = A_P + \frac{m_* \cdot A_L}{16} \left(\sum_k 2^{a_{j'} + b_k} + \sum_k 2^{a_{j''} + b_k} \right), \tag{3.32}$$

a po wprowadzeniu dodatkowego pola

$$A' = A_P + \frac{m_* \cdot A_L}{16} \left(\sum_k 2^{a_{j'} - 1 + b_k} + \sum_k 2^{a_{j''} - 1 + b_k} + \sum_k 2^{2 + b_k} \right). \tag{3.33}$$

Różnica maksymalnych obszarów przed i po wprowadzeniu dodatkowego pola jest równa

$$A - A' = C_A \cdot \left(\frac{1}{2} \cdot (2^{a_{j'}} + 2^{a_{j''}}) - 4 \right) \tag{3.34}$$

dla $C_A = \frac{m_* \cdot A_L}{16} \cdot \sum_k 2^{b_k}$. Dla $a_{j'} \geq 3$ i $a_{j''} \geq 3$ wartość dana wzorem (3.34) jest dodatnia, skąd wynika, że wprowadzając dodatkowe, 2–bitowe pole poprzez zmniejszenie szerokości pól o szerokościach $a_{j'} \geq 3$ i $a_{j''} \geq 3$ uzyskuje się redukcję maksymalnego sumarycznego obszaru pamięci. Maksymalny obszar UWIC maleje więc po wprowadzeniu dodatkowego, 2–bitowego pola kosztem zmniejszenia szerokości istniejących pól. □

Wniosek. *Z lematów 3.1.5 i 3.1.6 wynika, że zmniejszenie obszaru UWIC można osiągnąć przez ograniczenie maksymalnej szerokości adresu pamięci ROM i przez unikanie stosowania pamięci adresowanych słowem o maksymalnej szerokości.*

W dowodzie lematu 3.1.6 wykazano, że redukcja obszaru UWIC jest możliwa na skutek dodania dodatkowego pola do jednego z wektorów wejściowych UWIC. Co więcej, badając wartości różnic (3.31) i (3.34) można określić, w jaki sposób należy zmniejszać szerokości istniejących pól, aby uzyskać najmniejszy obszar układu. Jeśli

$$\frac{3}{4} \cdot 2^{a_j} > \frac{1}{2} \cdot (2^{a_{j'}} + 2^{a_{j''}}), \tag{3.35}$$

to mniejszy układ powstaje po dodaniu 2–bitowego pola kosztem szerokości $a_j \geq 4$. Nierówność (3.35) można przekształcić do postaci

$$2^{a_{j''}-a_j} + 2^{a_{j'}-a_j} < \frac{3}{2}, \tag{3.36}$$

która jest prawdziwa, jeśli $a_{j'}$ i $a_{j''}$ są mniejsze od a_j. Wynika stąd, że dla $a_{j'} < a_j$ i $a_{j''} < a_j$, dodanie 2–bitowego pola kosztem a_j pozwala na uzyskanie mniejszego obszaru UWIC, niż gdyby dodane pole powstało kosztem szerokości pól $a_{j'}$ i $a_{j''}$. Jeśli $a_{j'} > a_j$ lub $a_{j''} > a_j$, to mniejszy układ powstaje po zmniejszeniu szerokości pól $a_{j'}$ i $a_{j''}$.

Niestety, zmniejszanie szerokości adresów pamięci na rzecz ich większej ilości powoduje zwiększenie liczby iloczynów częściowych i w konsekwencji szerokości sumy tych iloczynów oraz obszaru sumatora wstępnego. Ponieważ jednak szerokość sumy iloczynów częściowych rośnie z logarytmem liczby dodawanych składników, liczba ogniw FA w sumatorze wstępnym zależy liniowo od liczby dodawanych składników (lemat 3.1.1), a obszar pamięci w UWIC wykładniczo w funkcji szerokości adresu (wzór (3.23)), stosowanie pamięci o dużej pojemności jest nieopłacalne. Na podstawie lematów 3.1.5 i 3.1.6 można oszacować górne ograniczenie obszaru UWIC, w którym iloczyny częściowe są obliczane wyłącznie za pomocą pamięci ROM.

Lemat 3.1.7. *Niech $K \geq 4$ oznacza maksymalną szerokość adresu pamięci ROM. Jeśli w układzie wytwarzania iloczynów częściowych o strukturze opisanej w rozdz. 3.1.1 zostaną użyte wyłącznie pamięci ROM adresowane słowem co najwyżej K–bitowym, to obszar układu wynosi co najwyżej $\frac{2^{K-5}}{K-2} \cdot m_*^3 \cdot A_L$, a wprowadzane opóźnienie $\lceil \log_4(K) \rceil \cdot T_L$.*

Dowód. Iloczyny częściowe są obliczane dla *wszystkich* kombinacji dwuelementowych zawierających po jednym polu z obu wektorów wejściowych. Stąd, maksymalna szerokość adresu pamięci ROM wystąpi, gdy kombinacja zawiera najszersze pola wydzielone z obu operandów. Jeśli więc maksymalna szerokość pól jednego z operandów wynosi a_{max}, to maksymalna szerokość pól dla drugiego z operandów może być co najwyżej $b_{max} = K - a_{max}$.

Zgodnie z lematami 3.1.5 i 3.1.6, układ wytwarzania iloczynów częściowych zajmie największy obszar, gdy szerokości adresów pamięci będą największe. Aby mieć pewność, że maksymalna liczba pamięci ROM będzie pamięciami adresowanymi słowem o maksymalnej szerokości, należy każdy z wektorów wejściowych UWIC podzielić na pola o takiej samej szerokości. Jeśli wektory wejściowe $\mathbf{X}_*, \mathbf{Y}_*$ będą dzielone na pola o różnej szerokości, kombinacje pól wektora $\mathbf{X}_*$ z polami wektora $\mathbf{Y}_*$ także będą miały różne szerokości.

Niech szerokość pól jednego z wektorów będzie równa $a_{max} \in [2, K-2]$, a szerokość pól drugiego wektora $K - a_{max}$. Liczba pól jednego z wektorów wyniesie więc $\frac{m_*}{a_{max}}$, a drugiego $\frac{m_*}{K-a_{max}}$. Liczba iloczynów częściowych jest więc równa $\frac{m_*^2}{a_{max}(K-a_{max})}$. Dla ustalonego m_* maksymalne wartości tej funkcji wystąpią dla $a_{max} = 2$ lub $a_{max} = K - 2$, a więc największa liczba iloczynów częściowych powstanie, gdy jeden z wektorów zostanie podzielony na pola o szerokości 2 bitów, a drugi $K - 2$ bitów. Liczba pamięci ROM wyniesie zatem $\frac{m_*}{2} \cdot \frac{m_*}{K-2}$, a parametry każdej z nich są opisane wzorem (3.23). Obszar UWIC jest więc ograniczony z góry przez $\frac{m_*^2}{2(K-2)} \cdot \left\lceil \frac{2^K \cdot m_*}{16} \right\rceil \cdot A_L$, a opóźnienie przez $\lceil \log_4(K) \rceil \cdot T_L$. Ponieważ minimalna wartość K jest ograniczona do 4, z wzoru opisującego obszar można usunąć operację sufitu, skąd bezpośrednio wynika wartość podana w lemacie 3.1.7. □

Z lematów 3.1.4 i 3.1.7 wynika, że maksymalny obszar układu wytwarzania iloczynów częściowych zależy od trzeciej potęgi szerokości słów wejściowych. Jeśli zrezygnuje się ze stosowania pamięci ROM, można zależność maksymalnego obszaru od m_* opisać funkcją kwadratową. Pozwala to przypuszczać, że dla odpowiednio dużego m_* stosowanie pamięci ROM w UWIC powoduje pogorszenie charakterystyk AT jednostki arytmetycznej. Aby jednak wykazać poprawność tego wniosku, należy zbadać parametry pozostałych bloków struktury z rys. 3.1.

Charakterystyki AT sumatora wstępnego

Struktura sumatora wstępnego jest opisana w rozdz. 3.1.2. Składa się on z kaskady sumatorów RCA. Złożoność tej kaskady zależy zarówno od liczby dodawanych składników, jak też od wag bitów w wektorach reprezentujących składniki. Charakterystyki AT sumatora wstępnego zależą więc od struktury układu wytwarzania iloczynów częściowych i liczby dodatkowych składników sumowanych w sumatorze wstępnym. Dokładne określenie parametrów AT sumatora wstępnego jest niezmiernie trudne z powodu skomplikowanej struktury wynikającej z rozbieżności wag bitów w sumowanych wektorach. Można jednak wykazać, że w zależności od struktury UWIC, obszar sumatora wstępnego zależy od m_*^2 lub co najwyżej m_*^3, a opóźnienie od $\log_2(m_*)$.

Lemat 3.1.8. *Niech $l \geq 0$ oznacza liczbę dodatkowych składników Z_*^i. Jeśli w UWIC do wytwarzania iloczynów częściowych używane są wyłącznie układy mnożące $2 \cdot m_*$ bitów, to istnieje struktura sumatora iloczynów częściowych o obszarze nie większym niż $\left(m_*^2 + m_* \cdot \left(2 \cdot l + \frac{\log_2(l)}{2}\right) + l \cdot \log_2(l)\right) \cdot A_L$ i opóźnieniu nie większym niż $\log_2\left(\frac{m_*}{2} + l\right) \cdot T_L$.*

Dowód. Jeśli w UWIC iloczyny częściowe są wytwarzane za pomocą układów mnożących $2 \cdot m_*$ bitów, to sumator iloczynów częściowych sumuje $\frac{m_*}{2}$ wektorów (m_*+2)–bitowych i l wektorów m_*–bitowych. Szerokość sumy jest nie większa niż $2m_* + \log_2(l)$ bitów. Charakterystyki AT sumatora wstępnego są określone na podstawie lematu 3.1.1 na str. 93 przy założeniu, że obszar pojedynczego ogniwa FA wynosi A_L, a opóźnienie sumatora m_* bitowego jest równe T_L. Przyjmując, że wszystkie sumatory mają szerokość równą szerokości sumy wyjściowej, górne ograniczenie rozmiaru sumatora iloczynów częściowych wynosi $\left(\frac{m_*}{2} + l\right) \cdot (2 \cdot m_* + \log_2(l))$. Długość ścieżki krytycznej sumatora iloczynów częściowych jest równa opóźnieniu wprowadzanemu przez kaskadę sumatorów dodającą $\frac{m_*}{2} + l$ składników, czyli $\log_2\left(\frac{m_*}{2} + l\right) \cdot T_L$. □

Lemat 3.1.9. *Niech $l \geq 0$ oznacza liczbę dodatkowych składników Z_*^i, a minimalna szerokość pól wydzielonych z wektorów wejściowych UWIC będzie równa 2. Jeśli w UWIC używane są pamięci ROM do wytwarzania iloczynów częściowych, to istnieje struktura sumatora iloczynów częściowych o obszarze nie większym niż $\left(\frac{m_*^3}{2} + \frac{m_*^2 \cdot \log_2(l)}{4} + 2 \cdot l \cdot m_* + l \cdot \log_2(l)\right) \cdot A_L$ i opóźnieniu nie większym niż $\log_2\left(\frac{m_*^2}{2^2} + l\right) \cdot T_L$.*

Dowód. Zgodnie z lematem 3.1.1 obszar zajęty przez kaskadę sumatorów zależy liniowo od liczby wektorów wejściowych i szerokości sumatorów. Liczba i szerokość wektorów wejściowych sumatora zależy od sposobu podziału wektorów wejściowych i struktury UWIC. Jeśli co najmniej jeden z iloczynów będzie obliczany za pomocą układu mnożącego, szerokość sumy iloczynów częściowych może osiągnąć wartość maksymalną określoną przez sumę iloczynu argumentów UWIC i dodatkowych składników Z_*^i (wzór (3.3)).

Poszukiwane jest górne ograniczenie obszaru sumatora iloczynów częściowych, można więc przyjąć, że wszystkie sumatory mają maksymalną szerokość $2 \cdot m_* + \log_2(l)$ równą maksymalnej szerokości sumy iloczynów argumentów UWIC i dodatkowych składników Z_*^i. Maksymalna liczba sumatorów zostanie użyta wtedy, gdy w UWIC powstanie maksymalna liczba iloczynów częściowych, czyli gdy oba wektory wejściowe UWIC zostaną podzielone na pola o szerokości 2 bitów. Liczba iloczynów częściowych wyniesie wtedy $\frac{m_*^2}{2^2}$. Przyjmując szerokość maksymalną $2 \cdot m_* + \log_2(l)$ dla wszystkich sumatorów, górne ograniczenie obszaru kaskady sumatorów wynosi $\left(\frac{m_*^2}{2^2} + l\right) \cdot (2 \cdot m + \log_2(l)) \cdot A_L$ na podstawie lematu 3.1.1. Maksymalne opóźnienie wprowadzane przez kaskadę sumatorów jest określone na podstawie lem. 3.1.1 jako opóźnienie układu dodającego $\frac{m_*^2}{2^2} + l$ składników. □

Z lematu 3.1.9 wynika, że jeśli w UWIC stosowane są pamięci ROM, to obszar sumatora wstępnego zależy od m_*^3. Jeśli w UWIC używane są wyłącznie układy mnożące $2 \cdot m_*$ bitów, obszar sumatora wstępnego zależy od m_*^2 (lemat 3.1.8). Dla niewielkich m_* używanie pamięci ROM może być korzystne, ponieważ w tym wypadku potęgi m_*^2 i m_*^3 są dzielone przez wielkości zależne od szerokości adresu zastosowanych pamięci ROM. Natomiast dla szerokich operandów zajmowany obszar może być zbyt duży. Jednak, aby określić wpływ struktury UWIC na parametry całej jednostki, należy jeszcze zbadać ostatni stopień ramowej struktury z rys. 3.1.

Charakterystyki AT generatora wyniku

Struktura generatora wyniku jest opisana w rozdz. 3.1.3. Składa się ona z kaskadowego połączenia reduktorów modulo i końcowego subtraktora warunkowego. Zgodnie z lematem 3.1.3, liczba reduktorów w kaskadzie nie przekracza dwóch lub trzech. Dokładna struktura reduktorów zależy od szerokości sumy iloczynów częściowych, na którą wpływają struktury UWIC oraz sumatora wstępnego. Dla uproszczenia rozważań można jednak generator wyniku rozpatrywać jako oddzielny układ wyznaczający resztę dla słowa k–bitowego.

Lemat 3.1.10. *Niech $K_{max} \in [4, k - m_* + 1]$ oznacza maksymalną szerokość pól wydzielonych z bardziej znaczącej części wektora wejściowego reduktora modulo. Obszar reduktora modulo zdefiniowanego w lemacie 3.1.3 jest nie większy niż*

$$\left\lceil \frac{k - m_* + 1}{K_{max}} \right\rceil \cdot \left\{ \left(2^{K_{max}-4} + 1\right) \cdot m_* + \log_2 \left(\left\lceil \frac{k - m_* + 1}{K_{max}} \right\rceil + 1 \right) \right\} \cdot A_L.$$

Dowód. Bardziej znacząca część wektora wejściowego jest podzielna na $\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil$ pól zawierających co najmniej 3 i co najwyżej K_{max} bitów każde. Jeśli szerokość pola przekracza 3 bity, zostanie dla niego zastosowana pamięć ROM o obszarze mniejszym lub równym pamięci o pojemności $2^{K_{max}} \cdot m_*$ bitów. Łączny obszar zajęty przez pamięci ROM jest więc ograniczony z góry przez $\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil \cdot 2^{K_{max}-4} \cdot m_*$.

Drugim elementem reduktora jest kaskada sumatorów obliczająca sumę $\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil + 1$ wektorów reprezentujących liczby mniejsze od M_*. Z lematu 3.1.1 wynika, że potrzeba $\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil$ sumatorów, których szerokość będzie rosła od m_* do $m_* + \lceil \log_2(k - m_* + 1 + K_{max}) - \log_2(K_{max}) \rceil$ bitów. Niech $l_s = \left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil$ oznacza liczbę sumatorów w kaskadzie. Górnym ograniczeniem obszaru kaskady sumatorów jest $l_s \cdot (m_* + \log_2(l_s + 1)) \cdot A_L$. Obszar całego reduktora jest więc ograniczony z góry sumą $\left(\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil \cdot 2^{K_{max}-4} \cdot m_* + \left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil \cdot \left(m_* + \log_2 \left(\left\lceil \frac{k-m_*+1}{K_{max}} \right\rceil + 1 \right) \right) \right) \cdot A_L$. □

Niestety, w niektórych przypadkach reduktor zdefiniowany w lemacie 3.1.3 jest układem o opóźnieniu liniowo zależnym od liczby bitów bardziej znaczącej części wektora wyjściowego. Sytuacja taka pojawi się wtedy, gdy $K_{min} = 3$ i $k - m_* + 1$ będzie taką wielokrotnością 3, której nie da się podzielić na pola o innej niż 3 szerokości. Przykładem jest wektor zawierający 9 bitów. Jeśli minimalny i maksymalny rozmiar pola wynosi 3 i 4 bity, to jedyną możliwością podziału jest podział na trzy pola 3–bitowe.

Lemat 3.1.11. *Dla reduktora modulo, w którym wszystkie pola bitowe mają szerokość 3 bitów, zajęty obszar mieści się w przedziale* $\left[\left\lceil\frac{k-m_*+1}{3}\right\rceil \cdot m_* \cdot A_L, \left\lceil\frac{k-m_*+1}{3}\right\rceil \cdot \left(m_* + \log_2\left(\left\lceil\frac{k-m_*+1}{3}\right\rceil\right)\right) \cdot A_L\right]$*, a wprowadzane opóźnienie jest nie większe niż* $\left\lceil\frac{k-m_*+1}{3}\right\rceil \cdot T_L$.

Dowód. Jeśli wszystkie pola bitowe wydzielone z bardziej znaczącej części wektora wejściowego mają szerokość 3 bitów, reszty dla tych pól są wyznaczane wyłącznie za pomocą pamięci ROM wbudowanych w sumatory RCA. Konieczne jest więc użycie sumatora dla każdego 3–bitowego pola, przy czym kolejne sumatory muszą być połączone kaskadowo. Pierwszy sumator w kaskadzie dodaje resztę do mniej znaczącej części wektora wejściowego, konieczne jest więc użycie $\left\lceil\frac{k-m_*+1}{3}\right\rceil$ sumatorów. Minimalną szerokością sumatora jest m_* bitów, a maksymalną $m_* + \log_2\left(\left\lceil\frac{k-m_*+1}{3}\right\rceil\right)$, skąd wynikają ograniczenia obszaru podane w lemacie 3.1.11. □

Aby uniknąć sytuacji, w której struktura opisana w lemacie 3.1.11 będzie jedyną możliwością konstrukcji reduktora modulo, w rozprawie zdecydowano, że liczba bardziej znaczących bitów wektora wejściowego reduktora dzielona na pola będzie mogła zmieniać się w przedziale $m_* \pm 1$. Zawsze będzie więc istniał podział, dla którego część lub wszystkie pola wydzielone z wyższej części wektora wejściowego będą miały szerokość większą niż 3 bity. Pozwoli to zmniejszyć długość ścieżki krytycznej do wartości zależnej od logarytmu liczby bitów dzielonych na pola.

Lemat 3.1.12. *Niech* $K_{min} > 3$ *oznacza minimalną szerokość pola wydzielonego z wyższej części* k*–bitowego wektora wejściowego reduktora modulo, a* $m_* - 1$ *minimalną szerokość niższej części tego wektora. Istnieje struktura reduktora modulo o długości ścieżki krytycznej nie większej niż*

$$\left(\lceil\log_4(K_{min})\rceil + \left\lceil\log_2\left(\left\lfloor\frac{k-m_*+1}{K_{min}}\right\rfloor + 1\right)\right\rceil\right) \cdot T_L.$$

Dowód. W reduktorze o minimalnej szerokości pola równej 4 dla każdego pola używana jest pamięć ROM wyznaczająca resztę modulo M_*. Otrzymane reszty i niższa część wektora wejściowego

są następnie sumowane w kaskadzie sumatorów. Opóźnienie jest więc równe sumie opóźnień najwolniejszej pamięci ROM (wzór (3.23)) i kaskady sumatorów. Dla $K_{min} > 3$ wszystkie wektory wejściowe kaskady są obliczane równocześnie, a więc zgodnie z lematem 3.1.1 liczba poziomów kaskady nie przekroczy $\log_2\left(\left\lfloor\frac{k-m_*+1}{K_{min}}\right\rfloor+1\right)$. Liczba poziomów kompletnego reduktora jest więc równa co najwyżej $\lceil\log_4(K_{min})\rceil+\left\lceil\log_2\left(\left\lfloor\frac{k-m_*+1}{K_{min}}\right\rfloor+1\right)\right\rceil$. □

Charakterystyki AT kompletnej jednostki

Na podstawie złożoności poszczególnych bloków jednostki skonstruowanej według struktury ramowej z rys. 3.1 można oszacować granice parametrów kompletnego układu. Najprostszym oszacowaniem maksymalnego obszaru dla układu zawierającego pamięci ROM w UWIC jest suma maksymalnych wartości danych przez lematy 3.1.7, 3.1.9 i 3.1.10 jako

$$\left(\left(\frac{2^{K-5}}{K-2}+\frac{1}{2}\right)\cdot m_*^3+\frac{2^{K_{max}-4}}{K_{max}}\cdot m_*^2+2\cdot l\cdot m_*\right)\cdot A_L. \tag{3.37}$$

Podobne oszacowanie można znaleźć dla ścieżki krytycznej jako

$$\left(\log_2(m_*^2)+\log_2(m_*)\right)\cdot T_L. \tag{3.38}$$

Są to jednak parametry dla jednostki, w której stosowana jest duża liczba pamięci ROM o największej dopuszczalnej pojemności. Znacznie bardziej interesujące są charakterystyki AT układu, w którym pamięci ROM są stosowane jedynie w nielicznych przypadkach (np. dla iloczynów częściowych pól zawierających najwyższe bity).

Twierdzenie 3.1.1. *Niech m_* oznacza szerokość wektorów reprezentujących operandy wejściowe X_*, Y_*, Z_*^i, a $l \geq 0$ oznacza liczbę składników Z_*^i. Istnieje jednostka arytmetyczna o strukturze opisanej w rozdz. 3.1 o obszarze nie większym niż $\left(2\cdot m_*^2+\left(8+2\cdot l++\frac{\log_2(m_*)}{4}\right)\cdot m_*\right)\cdot A_L$ i opóźnieniu nie większym niż $(\log_2(m_*+2\cdot l)+\log_2(m_*+\log_2(l)+5)+5)\cdot T_L$, przy czym z uwagi na ograniczenia wynikające ze struktury reduktora modulo musi być spełniony warunek $2\cdot m_*+\log_2(l)<2^{12}-17$.*

Dowód. Załóżmy, że w układzie wytwarzania iloczynów częściowych używane są wyłącznie układy mnożące $2\cdot m_*$ bitów. Zgodnie z lematem 3.1.4 UWIC jest układem jednopoziomowym o obszarze nie większym niż $\frac{m_*^2+m_*}{2}\cdot A_L$. Górne ograniczenie obszaru sumatora iloczynów częściowych jest określone przez lemat 3.1.8 jako $\left(m_*^2+2\cdot l\cdot m_*+\frac{m_*\cdot\log_2(l)}{2}+l\cdot\log_2(l)\right)\cdot A_L$. Liczba poziomów sumatora jest zdefiniowana przez lemat 3.1.1 jako $\log_2(\frac{m_*}{2}+l)$.

Wektor wyjściowy sumatora iloczynów częściowych jest $(2m_*+\log_2(l))$–bitowy. Niech generator wyniku zawiera kaskadowe połączenie dwóch reduktorów modulo, w których maksymalna szerokość adresu pamięci ROM wynosi 4 bity. Z tabeli 3.1.3 wynika, że maksymalna szerokość wektora wejściowego reduktora modulo o wektorze wyjściowym szerszym o dwa bity od m_* wynosi $m_* + 11$. Mniejszy obszar i opóźnienie zajmuje reduktor o $(m_* + 9)$–bitowym słowie wejściowym, można więc przyjąć, że będzie on drugim reduktorem w kaskadzie. Wyższa część $(m_* + 9)$–bitowego wektora wejściowego podzielona jest na dwa pola 3–bitowe i jedno 4–bitowe. Reduktor składa się więc z pamięci ROM o obszarze m_* i trzech sumatorów, z których dwa są m_*–bitowe z wbudowaną pamięcią, a jeden $(m_* + 1)$–bitowy. Długość ścieżki krytycznej wynosi $3 \cdot T_L$ (ROM i dwa sumatory). W pierwszym reduktorze w kaskadzie bardziej znacząca część wektora wejściowego jest podzielona na pola o szerokości 4 bitów, a szerokość jego wektora wyjściowego jest co najwyżej $m_* + 9$.

Z ograniczenia maksymalnej szerokości wektora wyjściowego pierwszego reduktora do $m_* + 9$ bitów można na podstawie lematu 3.1.2 wyznaczyć maksymalną szerokość k sumy iloczynów częściowych przez rozwiązanie nierówności

$$\left\lceil \log_2 \left(\left\lfloor \frac{k - m_* + 1}{4} \right\rfloor + 4 \right) \right\rceil \leq 9.$$

Po usunięciu funkcji podłogi i sufitu otrzymujemy

$$\log_2 \left(\frac{k - m_* + 1 + 16}{4} \right) < 10,$$

skąd

$$k - m_* < 2^{12} - 17. \tag{3.39}$$

Obszar pierwszego reduktora modulo jest określony według lematu 3.1.10 jako

$$\left(\left\lceil \frac{m_* + \log_2(l) + 1}{4} \right\rceil \cdot \left(2 \cdot m_* + \log \left(\left\lceil \frac{m_* + \log_2(l) + 1 + 4}{4} \right\rceil \right) \right) \right) \cdot A_L.$$

Liczba poziomów pierwszego reduktora modulo jest określona według lematu 3.1.12 jako

$$\log_2 \left(m_* + \log_2(l) + 5 \right) + 2.$$

W generatorze wyniku należy jeszcze zastosować wyjściowy subtraktor warunkowy, którego obszar jest co najwyżej $4 \cdot m_*$, a opóźnienie równe 2. Całkowity obszar i długość ścieżki krytycznej jednostki jest sumą obszarów i długości ścieżek krytycznych poszczególnych podukładów, czyli

$$\left(\frac{m_*^2 + m_*}{2} + 4 \cdot m_* + \frac{m_*^2}{2} + \frac{m_* \log_2(l)}{2} + \frac{m_*}{2} + \frac{m_* + \log_2(l)}{4} \cdot \log_2 \left(\frac{m_* + \log_2(l)}{4} \right) + 4 \cdot m_* \right) \cdot A_L$$

oraz

$$\left(1+3+\log_2(\frac{m_*}{2}+l)+\log_2\left(m_*+\log_2(l)+5\right)+2+2\right)\cdot T_L,$$

skąd wynikają przybliżenia podane w tw. 3.1.1. □

Z twierdzenia 3.1.1 wynika, że jest możliwe znaczne zmniejszenie obszaru i opóźnienia kompletnej jednostki arytmetycznej w stosunku do oszacowań (3.37) i (3.38). Twierdzenie 3.1.1 dotyczy wyłącznie układów, w których w UWIC nie są używane pamięci ROM. Z dotychczasowych rozważań wynika jednak, że dla niewielkich szerokości modułu stosowanie pamięci ROM o małej pojemności może być uzasadnione. Niestety, skomplikowana natura zależności definiujących strukturę i charakterystyki AT jednostki arytmetycznej znacznie utrudnia rozwiązanie analityczne problemu znalezienia struktury optymalnej. Należy zatem stosować inne metody pozwalające znaleźć strukturę jednostki o poszukiwanych parametrach.

3.2 Algorytm automatycznej generacji jednostek arytmetycznych

Dla danego modułu i działania możliwe są różne realizacje układu w strukturze podanej na rys. 3.1. Dokładne oszacowanie parametrów podczas projektowania układu jest jednak niezmiernie trudne. W trakcie projektowania układu konieczne jest więc wielokrotne wprowadzanie zmian i sprawdzanie ich skutków. Korzystne jest więc zautomatyzowanie tego procesu pozwalające na odciążenie projektanta, przyspieszenie procesu projektowania i uniknięcie błędów.

Idea proponowanego algorytmu generowania jednostek mnożenia akumulacyjnego modulo polega na utworzeniu podzbioru możliwych konfiguracji układu, oszacowaniu parametrów powstałych jednostek i wybraniu tej o poszukiwanych parametrach. W algorytmie badane są jedynie te jednostki, których struktura podlega ograniczeniom wybranym na podstawie poszukiwanych parametrów. Pozwala to wyeliminować na samym początku struktury o charakterystykach AT znacznie różniących się od poszukiwanych. Dla pozostałych jednostek przeprowadzany jest następnie pełny proces implementacji, po czym wybierane jest rozwiązanie o parametrach najbliższych poszukiwanym.

Generowanie zbioru możliwych konfiguracji składa się z dwóch etapów. W pierwszym kroku tworzone są konfiguracje układu wytwarzania iloczynów częściowych i odpowiadające im struktury sumatora wstępnego. Drugi etap obejmuje konstrukcję wszystkich możliwych generatorów wyniku dla uzyskanych szerokości wektorów wyjściowych sumatora wstępnego. Następnie z każdym zesta-

wem generatora iloczynów częściowych i sumatora wstępnego łączone są wszystkie struktury generatora wyniku operujące na wektorze o szerokości równej szerokości sumy iloczynów częściowych. Po utworzeniu dopuszczalnych konfiguracji jednostek arytmetycznych są one sortowane względem wybranego kryterium, którym może być zajmowany obszar A, wprowadzane opóźnienie T lub jeden z iloczynów AT czy AT^2. Posortowanie wyników umożliwia szybkie wytypowanie podzbioru jednostek o zbliżonych parametrach.

3.2.1 Układ wytwarzania iloczynów częściowych i sumator wstępny

Skonstruowanie układu wytwarzania iloczynów częściowych i sumatora wstępnego wymaga podjęcia dwóch decyzji określających strukturę tych bloków. Pierwszą z nich jest sposób podziału na pola bitowe wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$ reprezentujących czynniki. Druga decyzja dotyczy wyboru metody wytwarzania iloczynu częściowego σ_{jk} dla każdej pary pól wydzielonych z wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$. Obie te decyzje określają także strukturę sumatora wstępnego, która zależy od postaci iloczynów σ_{jk}.

Dla zadanego modułu M_* i liczby l dodatkowych składników Z_*^i, liczba możliwych szerokości wektora sumy $\mathbf{H}$ może osiągnąć najwyżej $m_* = \lfloor \log_2(M_*) \rfloor + 1$ wartości: od m_* bitów do $2m_*$ bitów. Ponieważ struktura generatora wyniku zależy od szerokości wektora sumy $\mathbf{H}$, więc liczba możliwych konfiguracji generatora jest funkcją zakresu możliwych szerokości tego wektora. Liczba możliwych konfiguracji układu wytwarzania iloczynów częściowych i sumatora wstępnego może znacznie przekroczyć m_*. Celowe jest zatem oszacowanie szerokości $\mathbf{H}$ dla wygenerowanych struktur układu wytwarzania iloczynów częściowych i sumatora wstępnego.

Wynikiem algorytmu generowania układu wytwarzania iloczynów częściowych i sumatora wstępnego są więc dwa zbiory. Pierwszy z nich zawiera wybrane konfiguracje wymienionych bloków, w drugim znajdują się szerokości wektora $\mathbf{H}$ będącego wyjściem sumatora wstępnego. Konfiguracje układu wytwarzania iloczynów częściowych i sumatora wstępnego są jednoznacznie opisane za pomocą zdefiniowanych w rozdz. 3.1 wektorów $\mathbf{a}$ i $\mathbf{b}$ oraz macierzy $\mathbf{c}$. Wektory $\mathbf{a}$ i $\mathbf{b}$ opisują sposób podziału wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$ na pola bitowe. Macierz $\mathbf{c}$ opisuje sposób obliczania iloczynów σ_{jk} dla wszystkich kombinacji pól bitowych wydzielonych z $\mathbf{X}_*$ i $\mathbf{Y}_*$. Zbiór konfiguracji układu wytwarzania iloczynów częściowych i sumatora wstępnego oznaczony jest jako zbiór $\Upsilon = \{(\mathbf{a}, \mathbf{b}, \mathbf{c})\}$. Zbiór możliwych szerokości wektora $\mathbf{H}$ jest oznaczony przez $\hat{H}$.

Pierwszym krokiem algorytmu jest znalezienie wszystkich możliwych podziałów wektora m_*–

bitowego na pola o szerokości minimum 2 bity. W drugim kroku, dla znalezionych podziałów tworzone są kombinacje dwuelementowe z powtórzeniami opisujące podziały wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$. Trzeci etap algorytmu obejmuje wybranie metody obliczania iloczynów częściowych σ_{jk} dla wszystkich kombinacji pól bitowych określonych przez sposób podziału wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$. Ostatnim krokiem jest zbudowanie możliwych struktur sumatora wstępnego i znalezienie zbioru szerokości wektora $\mathbf{H}$. Formalny zapis kompletnego algorytmu generowania układu wytwarzania iloczynów częściowych i sumatora wstępnego przedstawia alg. 7.

Algorytm 7 Algorytm generowania wybranych konfiguracji układu wytwarzania iloczynów częściowych i sumatora wstępnego

Dane wejściowe: M_*, l

1: generuj zbiór $\Gamma(m_*) = \{\mathbf{\Gamma}^i\}$ kompozycji m_* nie zawierających składników 1
2: **for** $j = 0$ **to** $|\Gamma(m_*)|$ **do**
3: $\quad \mathbf{a} = \mathbf{\Gamma}^j$
4: $\quad$ **for** $k = 0$ **to** $|\Gamma(m_*)|$ **do**
5: $\quad\quad \mathbf{b} = \mathbf{\Gamma}^k$
6: $\quad\quad$ Znajdź elementy $\mathbf{c}$ według równania (3.40)
7: $\quad\quad$ Dopisz $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ do zbioru Υ
8: $\quad\quad$ Znajdź szerokość sumy $\mathbf{H}$ dla utworzonego układu
9: $\quad\quad$ Jeśli obliczona szerokość nie należy do zbioru $\hat{H}$, dopisz ją do zbioru $\hat{H}$
10: $\quad$ **end for**
11: **end for**
12: **return** $\Upsilon, \hat{H}$

Układ wytwarzania iloczynów częściowych

Podziały wektora m_*–bitowego są opisane za pomocą kompozycji liczby m_* nie zawierających składników 1 (rozdz. 2.2.1, str. 45). Po znalezieniu zbioru $\Gamma(m_*)$ zawierającego kompozycje m_*, kolejnym krokiem algorytmu jest utworzenie par wektorów $\mathbf{a}$, $\mathbf{b}$ jako dwuelementowych kombinacji z powtórzeniami elementów zbioru $\Gamma(m_*)$. Następnie dla każdej pary $\mathbf{a}$, $\mathbf{b}$ wybierane są metody obliczania poszczególnych iloczynów częściowych σ_{jk}.

W celu zmniejszenia liczby konfiguracji układu, metoda obliczania σ_{jk} jest uzależniona od szero-

kości pól, dla których dany iloczyn jest obliczany. Pozwala to na wyeliminowanie najbardziej kosztownych struktur na początku algorytmu. Kryterium wyboru metody wytwarzania iloczynów częściowych jest szerokość pól, dla których dany iloczyn jest obliczany. Dla małych szerokości stosowane są pamięci ROM, dla dużych szerokości tańszym rozwiązaniem jest układ mnożenia $2 \cdot k$ bitów. W proponowanym rozwiązaniu układ mnożenia $2 \cdot k$ lub $3 \cdot k$ bity jest wybierany tylko wtedy, gdy jedno z pól ma szerokość 2 lub 3 bity, a drugie co najmniej K bitów. Dla uproszczenia procedury w opisywanym rozwiązaniu przyjęto $K = m_*$. W pozostałych przypadkach stosowana jest pamięć ROM. Funkcja określająca wartości elementów macierzy $\boldsymbol{z}$ wynosi więc

$$\chi(a_j, b_k) = \begin{cases} 1 & \text{dla} \quad (a_j \in [2,3] \cap b_k \geq K) \cup (b_k \in [2,3] \cap a_j \geq K) \\ 0 & \text{w pozostałych przypadkach} \end{cases} . \tag{3.40}$$

Uzasadnieniem tej decyzji są trzy argumenty. Po pierwsze, matryce mnożące operandy dowolnej szerokości składają się z układów mnożenia $2 \cdot k$ bitów oraz sumatorów CPA, tak więc po dodaniu sumatora wstępnego przypadki te powstają automatycznie. Po drugie, pominięcie pamięci ROM dla szerokich pól nie wpływa na zmniejszenie liczby użytecznych układów ze względu na wyeliminowanie jedynie rozwiązań o dużym obszarze. Po trzecie, stosowanie układów mnożenia $2 \cdot k$ bitów dla wąskich pól jest w dużej części przypadków nieefektywne ze względu na możliwość znacznego zwiększenia szerokości wektora $\mathbf{H}$ i w konsekwencji złożoności generatora wyniku. Przykładem może być obliczenie iloczynu częściowego dla pól zawierających dwa najstarsze bity $\mathbf{X}_*$ i $\mathbf{Y}_*$. W tym przypadku lepszym rozwiązaniem jest zastosowanie pamięci ROM, w której przy porównywalnym obszarze można zredukować wartość iloczynu częściowego modulo M_*.

Sumator wstępny i oszacowanie szerokości sumy iloczynów częściowych

Struktura sumatora wstępnego jest określona przez postać iloczynów σ_{jk}, która zależy od wyznaczonych dotychczas elementów zbioru $\Upsilon = \{(\mathbf{a}, \mathbf{b}, \mathbf{c})\}$. Sumator wstępny jest konstruowany według alg. 4 opisanego w rozdz. 3.1.2 na str. 96. Następnie, dla każdego sumatora wstępnego obliczana jest szerokość wektora sumy $\mathbf{H}$. Obliczone szerokości są dopisywane do zbioru $\hat{H}$.

Szerokość wektora zależy od maksymalnej wartości sumy H. Maksymalna wartość H jest ograniczona od góry przez sumę maksymalnych wartości wszystkich iloczynów σ_{jk} i składników Z_*^i. Maksymalne wartości składników Z_*^i wynoszą $M_* - 1$. W ten sam sposób określone są maksymalne wartości tych iloczynów częściowych, które są wytwarzane za pomocą pamięci ROM. Dla iloczynów

częściowych obliczanych za pomocą układów mnożących wartością maksymalną jest liczba reprezentowana przez wektor zawierający same "1".

Obliczony według powyższej reguły zakres wartości szerokości wektora H wynosi $[m_*, 2 \cdot m_* + \log_2(l+1)]$. Minimum równe m_* występuje dla $l = 0$ i pojedynczego iloczynu częściowego obliczonego za pomocą pamięci ROM adresowanej złożeniem całych wektorów $\mathbf{X}_*$ i $\mathbf{Y}_*$. Maksymalna szerokość $\mathbf{H}$ oznacza liczbę bitów wymaganą w przypadku, gdy wszystkie iloczyny częściowe są wytwarzane za pomocą układów mnożących $2 \cdot k$ lub $3 \cdot k$ bitów.

Rzeczywista maksymalna wartość H może być mniejsza od oszacowania przyjętego jako suma wartości maksymalnych wszystkich składników. Wynika to z nieliniowości funkcji realizowanych przez pamięci ROM oraz ograniczonego zbioru wartości argumentów X_* i Y_*. Ponieważ X_* i Y_* są ograniczone do przedziału $[0, M_*)$, może się zdarzyć, że kombinacje bitów wymagane do wymuszenia wartości maksymalnych dla wszystkich iloczynów częściowych są poza zakresem wartości X_* i Y_*. Szczególnie widoczne jest to w przypadku, gdy część iloczynów częściowych jest wytwarzana za pomocą pamięci ROM, a część za pomocą układów mnożących. Wymuszenie maksymalnych wartości na wyjściach układów mnożących wymaga podania na ich wejścia słów zawierających same "1". Dla pamięci ROM wymagane są zupełnie inne kombinacje, zależne od wag poszczególnych bitów i modułu M_*. Co więcej, jest możliwa sytuacja, w której dla zadanych wag bitów adresowych niemożliwe jest otrzymanie liczby, której reszta modulo M_* wynosi $M_* - 1$. Niestety, dokładne oszacowanie maksymalnej wartości H wymaga sprawdzenia wszystkich kombinacji wartości operandów wejściowych, co jest niezwykle kosztowne.

3.2.2 Generator wyniku

Po wyznaczeniu elementów zbioru $\hat{H}$ można wygenerować wszystkie wymagane struktury generatora wyniku. Struktura generatora wyniku zależy od sposobu podziału wektora $\mathbf{H}$ na pola, który jest ściśle zależny od szerokości tego wektora. Dopasowanie poszczególnych struktur generatora wyniku do odpowiedniego sumatora wstępnego dokonywane jest na podstawie szerokości wektora $\mathbf{H}$ łączącego te bloki.

Konfiguracja pojedynczego generatora wyniku jest zdefiniowana przez sposoby podziałów na pola bitowe wektorów wejściowych kolejnych reduktorów w kaskadzie. Znając sposób podziału wektora wejściowego danego reduktora można jednoznacznie określić jego strukturę na podstawie alg. 6 ze

str. 107. Sposób podziału wektora wejściowego i–tego reduktora jest opisany wektorem $\mathbf{d}^i$. Kolejne wektory $\mathbf{d}^i$ tworzą macierz $\mathbf{D}$ opisującą strukturę kompletnej kaskady reduktorów modulo. Wektory $\mathbf{d}^i$ są tak dobierane, aby liczby wyjściowe kolejnych reduktorów modulo w kaskadzie tworzyły ciąg malejący zbieżny do liczby mniejszej od $4 \cdot M_*$.

Wszystkie wygenerowane konfiguracje generatora wyniku są opisane za pomocą zbioru $\Lambda\colon \{\lambda = (\mathbf{D})\}$ macierzy opisujących kaskady reduktorów modulo. Dla każdej szerokości ze zbioru $\hat{H}$ może powstać wiele macierzy $\mathbf{D}$ opisujących różne struktury kaskady.

Algorytm konstruowania struktur generatora wyniku

Zadaniem algorytmu konstruowania generatora wyniku (alg. 8) jest zbudowanie kaskad reduktorów modulo zakończonych subtraktorem warunkowym. Wejściem kaskad jest wektor $\mathbf{H}$ o dopuszczalnych szerokościach zapisanych w zbiorze $\hat{H}$. Konstrukcja kaskad dla zadanej szerokości (alg. 9) jest zrealizowana za pomocą kolejnych iteracji odpowiedzialnych za utworzenie możliwych konfiguracji reduktorów modulo na tym samym poziomie kaskady. Iteracja jest kontynuowana do momentu, w którym maksymalne wartości liczb wyjściowych wszystkich wygenerowanych reduktorów modulo są mniejsze od $4 \cdot M_*$. Po zakończeniu części iteracyjnej algorytmu każda kaskada reduktorów jest uzupełniana o subtraktor warunkowy. Subtraktor zawiera zestaw komparatorów, których liczba zależy od maksymalnej wartości liczby wyjściowej ostatniego reduktora kaskady.

Algorytm 8 Algorytm automatycznej generacji wszystkich konfiguracji generatora wyniku

Dane wejściowe: Zbiór $\hat{H}$ zawierający szerokości wektorów wyjściowych sumatorów wstępnych

1: **for** $i = 1$ **to** $\left|\hat{H}\right|$ **do**
2: **if** $\hat{h}_i \geq m_* + 3$ **then**
3: Generuj dla zadanej szerokości $\hat{h}_i$ zestaw wszystkich możliwych konfiguracji kaskad reduktorów modulo zgodnie z alg. 9
4: Wszystkie otrzymane konfiguracje dopisz do zbioru Λ
5: **else**
6: Dopisz do zbioru Λ macierz $\mathbf{D}$ zawierającą wektor $\boldsymbol{d}^0 = (\hat{h}_i)$
7: **end if**
8: **end for**
9: **return** Λ

Istotną operacją w alg. 9 jest znalezienie wszystkich struktur pojedynczego reduktora modulo dla zadanej szerokości wektora wejściowego (linie 1 i 13). Struktura reduktora jest jednoznacznie określona przez wektor $\mathbf{d}^i$ opisujący sposób podziału wektora wejściowego na pola bitowe. Poszczególne elementy wektora $\mathbf{d}^i$ muszą być tak dobrane, aby lemat 3.1.3 ze str. 103 był spełniony, czyli $d_0^i \in [m_* - 1, m_* + 1]$ i $d_j^i \geq 3$ dla $j > 0$. Górne ograniczenie wartości d_j^i nie jest wymagane dla zapewnienia poprawności działania układu. Ma ono jednak istotny wpływ na złożoność całego układu.

Wszystkie układy prezentowane w niniejszej pracy są projektowane pod kątem FPGA. Pamięci ROM są tam z reguły budowane z wykorzystaniem elementarnych pamięci o małej pojemności (*ang. Look–Up Tables, LUT*). Używanie pamięci adresowanej słowem szerszym niż adres pojedynczej pamięci elementarnej wymaga konstruowania układu zawierającego dodatkowe elementy. Liczba wykorzystanych LUT zależy wykładniczo od szerokości adresu pamięci, którą należy zaimplementować. W układach rodziny Spartan 2/Virtex tablice LUT mają pojemność 16×1 bitów. Wynika stąd, że zbudowanie pamięci adresowanej słowem 5–bitowym może wymagać dwupoziomowego układu, którego opóźnienie będzie znacznie większe od opóźnienia pojedynczej LUT. Liczba wykorzystanych LUT może być dwukrotnie większa, niż dla pamięci adresowanej słowem 4–bitowym. Zatem, używanie pamięci adresowanych słowem 5–bitowym wiąże się ze znacznym wzrostem kosztów w stosunku do pamięci adresowanych słowem 4–bitowym. Dla szerszych słów adresowych koszty wzrastają wykładniczo. Z tego powodu w algorytmach konstrukcji reduktora modulo przyjęto ograniczenie maksymalnej szerokości pól d_j^i dla $j > 0$ równe szerokości adresu LUT, czyli 4 bity.

Ograniczenie maksymalnej szerokości pola do 4 bitów powoduje, że jedynymi dopuszczalnymi szerokościami pól d_j^i są 3 lub 4 bity. Pola te są adresami pamięci ROM, których obszar i wprowadzane opóźnienie są praktycznie niezależne od ich zawartości. Kolejność pól wydzielonych z wektora wejściowego reduktora modulo nie ma więc większego znaczenia. Można więc przyjąć, że jedynymi interesującymi strukturami reduktora modulo są układy opisane przez wektory $\mathbf{d}$, dla których

$$d_0 + 3 \cdot k_3 + 4 \cdot k_4 = \hat{h}, \tag{3.41}$$

gdzie k_3 oznacza liczbę elementów d_j równych 3, a k_4 oznacza liczbę elementów $d_j = 4$. Kolejność elementów d_j dla $j > 0$ można przyjąć dowolną, czyli np. elementy o wyższych indeksach przyjmują wartość 4, a pozostałe 3. Podejście to pozwala na szybkie wyznaczenie możliwych sposobów podziału wektora o szerokości $\hat{h}$, ponieważ rozwiązania (3.41) można znaleźć za pomocą algorytmu Euklidesa.

Algorytm 9 Algorytm generacji wszystkich kaskad reduktorów dla zadanej szerokości $\hat{h}$

Dane wejściowe: Szerokość $\hat{h}$

1: Za pomocą alg. Euklidesa znajdź wszystkie struktury pojedynczego reduktora modulo dla zadanego $\hat{h}$ i zapisz je do zbioru Λ_{tmp1}
2: **repeat**
3: $\quad\Lambda_{tmp2} = \emptyset$
4: $\quad$**for** $\mathbf{D}$ równego kolejnym kaskadom z Λ_{tmp1} **do**
5: $\quad\quad$**if** wartość maksymalna liczby wyjściowej kaskady $\mathbf{D} < 4 \cdot M_*$ **then**
6: $\quad\quad\quad$Dopisz $\mathbf{D}$ do zbioru wynikowego Λ_w
7: $\quad\quad$**else**
8: $\quad\quad\quad$Dopisz $\mathbf{D}$ do zbioru Λ_{tmp2}
9: $\quad\quad$**end if**
10: $\quad$**end for**
11: $\quad\Lambda_{tmp1} = \emptyset$
12: $\quad$**for** $\mathbf{D}$ równego kolejnym kaskadom z Λ_{tmp2} **do**
13: $\quad\quad$Za pomocą alg. Euklidesa znajdź wszystkie struktury pojedynczego reduktora modulo dla wektora wyjściowego kaskady opisanej przez $\mathbf{D}$
14: $\quad\quad$**for** wszystkich wygenerowanych reduktorów **do**
15: $\quad\quad\quad$Dodaj bieżący reduktor na koniec kaskady $\mathbf{D}$ i wynik dopisz do Λ_{tmp1}
16: $\quad\quad$**end for**
17: $\quad$**end for**
18: **until** $\Lambda_{tmp2} \neq \emptyset$
19: **return** Λ_w

Procedura wyznaczania wszystkich struktur reduktora modulo dla zadanej szerokości $\hat{h}$ może więc zostać zaimplementowana bardzo prosto. Wystarczy dla wszystkich wartości $d_0 \in [m_* - 1, m_* + 1]$ znaleźć rozwiązania spełniające (3.41).

Po utworzeniu wszystkich kaskad reduktorów modulo należy każdą z nich uzupełnić subtraktorem warunkowym odejmującym odpowiednią krotność M_*. Subtraktor warunkowy składa się ze zbioru komparatorów sterujących sumatorem z pamięcią przedstawionym na rys. 3.5 na str. 99. Ograniczenie wartości liczby wyjściowej kaskady modulo powoduje, że wymaganą krotność można znaleźć za

pomocą maksymalnie 3 komparatorów. Wyjścia komparatorów można zatem podać bezpośrednio na trzy wejścia adresowe pamięci ROM w sumatorze z pamięcią.

Konstrukcja subtraktora warunkowego jest określona wyłącznie przez maksymalną wartość liczby wyjściowej kaskady sumatorów. Z powodu jej prostoty można więc zrezygnować z jakichkolwiek dodatkowych metod reprezentacji subtraktora warunkowego. Wystarczy posiadać informację o maksymalnej wartości liczby wyjściowej kaskady reduktorów i na tej podstawie użyć jednego z kilku predefiniowanych subtraktorów.

3.2.3 Kompletna jednostka arytmetyczna

Struktura jednostki arytmetycznej powstaje w wyniku połączenia struktur układu wytwarzania iloczynów częściowych i sumatora wstępnego z odpowiednimi generatorami wyniku. Struktury układu wytwarzania iloczynów częściowych i sumatora wstępnego są opisane elementami zbioru $\Upsilon\colon \{(\mathbf{a}, \mathbf{b}, \mathbf{c})\}$. Struktury generatora wyniku są opisane elementami zbioru $\Lambda\colon \{(\mathbf{D})\}$. Do pełnego opisu kompletnej jednostki wymagana jest znajomość struktur opisanych zbiorami Υ i Λ. Zbiór wszystkich struktur jednostek arytmetycznych jest oznaczony przez $\Omega = \{(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{D})\}$. Elementy zbiorów Υ i Λ są łączone na podstawie szerokości wektora łączącego opisywane przez nie podukłady.

Algorytm 10 Algorytm automatycznej generacji kompletnych jednostek arytmetycznych

Dane wejściowe: M_*, l

1: Generuj według alg. 7 zbiory $\Upsilon, \hat{H}$ opisujące konfiguracje układu wytwarzania iloczynów częściowych i sumatora wstępnego
2: Na podstawie $\hat{H}$ generuj według alg. 8 zbiór Λ opisujący konfiguracje generatora wyniku
3: **for** υ reprezentującego kolejne elementy Υ **do**
4: **for** λ reprezentującego kolejne elementy Λ **do**
5: **if** szerokość wektora wyjściowego υ jest równa szerokości wektora wejściowego λ **then**
6: Dopisz do zbioru Ω połączenie układów υ i λ
7: **end if**
8: **end for**
9: **end for**
10: Sortuj zbiór Ω według wybranego kryterium
11: **return** Ω

Algorytm tworzenia struktur kompletnej jednostki arytmetycznej (alg. 10) sprowadza się do znalezienia wszystkich odpowiadających sobie par elementów zbioru Υ i Λ. Po wygenerowaniu wszystkich struktur obliczane są ich parametry AT przy pomocy wzorów (3.22)-(3.25). Następnie zbiór rozwiązań jest sortowany według wybranego kryterium, którym może być zajmowany obszar A, wprowadzane opóźnienie T lub jeden z iloczynów AT lub AT^2. Posortowanie wyników umożliwia łatwy wybór podzbioru jednostek o poszukiwanych parametrach.

3.2.4 Złożoność obliczeniowa

Zadaniem alg. 10 jest wygenerowanie zestawu jednostek arytmetycznych i określenie ich charakterystyk AT. Złożoność obliczeniowa algorytmu może być więc opisana liczbą tworzonych konfiguracji jednostek arytmetycznych. Głównymi czynnikami wpływającymi na złożoność algorytmu są zatem liczba możliwych konfiguracji bloku złożonego z układu wytwarzania iloczynów częściowych i sumatora wstępnego oraz liczba możliwych konfiguracji generatora wyniku.

Liczba konfiguracji układu wytwarzania iloczynów częściowych

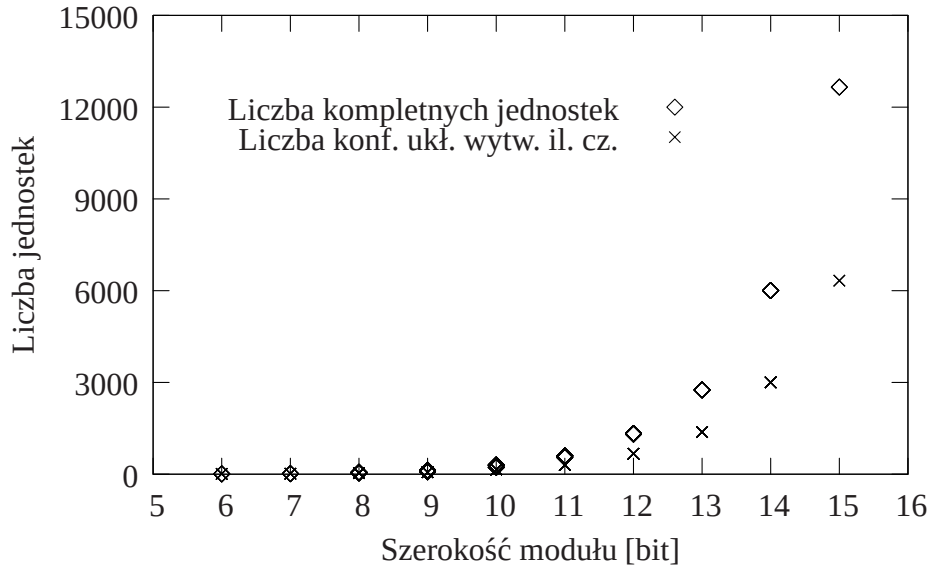
Wszystkie konfiguracje układu wytwarzania iloczynów częściowych i sumatora wstępnego zawarte są w zbiorze Υ. Każdy element zbioru Υ powstaje w wyniku wyboru sposobu podziału operandów wejściowych i metody generowania iloczynów częściowych. Ponieważ metoda wytwarzania iloczynów częściowych zależy od szerokości pól, dla każdej pary podziałów operandów wejściowych istnieje tylko jedna metoda generowania iloczynów częściowych. Z tego powodu moc zbioru Υ zależy wyłącznie od liczby możliwych podziałów wektora o szerokości $m_* = \lfloor \log_2(M_*) \rfloor + 1$ bitów. Sposoby podziału są opisane przez zawarte w zbiorze $\Gamma(m_*)$ kompozycje liczby m_* nie zawierające składników 1. Elementy zbioru Υ powstają w wyniku kombinacji dwuelementowych z powtórzeniami elementów zbioru $\Gamma(m_*)$, tak więc ich liczba jest określona jako

$$|\Upsilon| = \frac{|\Gamma(m_*)| \cdot (|\Gamma(m_*)| + 1)}{2}. \tag{3.42}$$

Zgodnie z (2.27), moc zbioru $\Gamma(m_*)$ zależy wykładniczo od m_*, stąd liczba możliwych konfiguracji bloku złożonego z układu wytwarzania iloczynów częściowych i sumatora wstępnego jest rzędu $O(\phi^{2m_*})$ dla $\phi = \frac{1+\sqrt{5}}{2}$.

Liczba konfiguracji kompletnych jednostek

Określenie liczby możliwych konfiguracji generatora wyniku jest dość trudne. Liczba ta zależy zarówno od liczby możliwych szerokości $\hat{h} \in \hat{H}$ wektora wyjściowego sumatora wstępnego, jak i od wartości tych szerokości. Wartość szerokości $\hat{h}$ jest zależna od metody wytwarzania iloczynów częściowych oraz od konstrukcji kaskady sumatorów w sumatorze wstępnym, która ulega modyfikacji dla modułów o małym okresie potęg 2 modulo M_*. Dodatkowo, dla danej szerokości $\hat{h}$ możliwości konstrukcji generatora wyniku są ściśle związane z ograniczeniem na rozmiar pól d_j^i, dla których obliczane są reszty modulo M_*. Z powyższych względów dokładne zależności opisujące liczbę możliwych konfiguracji kompletnej jednostki arytmetycznej nie są znane.



Rysunek 3.6. Liczba możliwych konfiguracji jednostki modulo w funkcji szerokości modułu $m_* = \lfloor \log_2(M_*) \rfloor + 1$ dla $l = 1$.

Złożoność obliczeniowa kompletnego algorytmu może jednak być oszacowana ze stosunkowo niewielkim błędem. Należy zauważyć, że dla założonych ograniczeń na d_j^i dla $j > 0$ liczba możliwych struktur generatora wyniku dla szerokości $\hat{h} < 30$ nie przekracza kilkunastu, dodatkowo dla wielu wartości $\hat{h}$ jest ona dużo mniejsza. W związku z tym złożoność całego algorytmu może być przybliżona z dokładnością do rzędu wielkości liczbą możliwych konfiguracji układu wytwarzania iloczynów częściowych. Na rys. 3.6 przedstawiono zależność liczby konfiguracji układu wytwarzania iloczynów częściowych oraz liczby wszystkich wygenerowanych konfiguracji kompletnej jednostki arytmetycznej od szerokości modułu. Dane dla wykresu uzyskano poprzez wygenerowanie jednostek

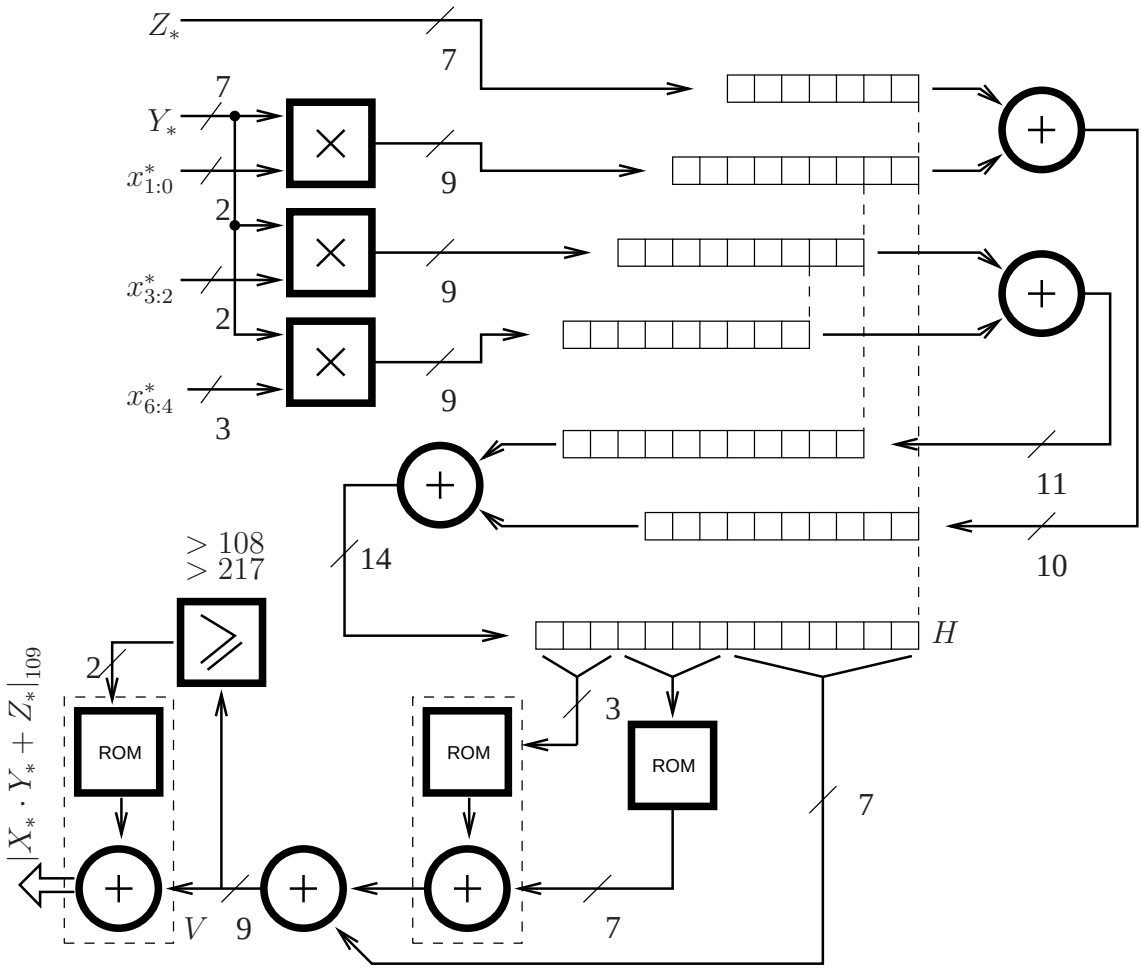
dla 10 wartości modułu M_* dla każdej szerokości m_*. Wartości M_* rozmieszczono równomiernie po całym przedziale $(2^{m_*-1}, 2^{m_*})$. Można zauważyć, że dla pokazanego zakresu modułów liczba konfiguracji kompletnego układu jest co najwyżej 2-3 krotnie większa od liczby konfiguracji układu wytwarzania iloczynów częściowych. Ponieważ liczba układów wytwarzania iloczynów częściowych zależy od $|\Gamma(m_*)|^2$, wobec tego złożoność całego algorytmu jest rzędu $O(\phi^{2 \cdot m_*})$.

3.3 Podsumowanie

Zaproponowana ramowa struktura jednostek arytmetycznych pozwala na implementację dowolnej kombinacji mnożenia i dodawania modulo. Zasadniczą zaletą rozwiązania w porównaniu z dotychczasowymi układami jest wyeliminowanie pamięci ROM o dużej pojemności na rzecz efektywnego wykorzystania podstawowych elementów konstrukcyjnych dostępnych w nowoczesnych FPGA. Dzięki temu zaprojektowana struktura umożliwia skonstruowanie jednostki arytmetycznej o obszarze zależnym od kwadratu szerokości modułu i opóźnieniu zależnym od logarytmu z szerokości modułu.

Jako przykład jednostki o nowej strukturze można przeanalizować pokazany na rys. 3.7 jeden z możliwych układów dla $M_* = 109$. Pierwszym stopniem jednostki jest zbiór trzech układów mnożących obliczających 9–bitowe iloczyny częściowe powstałe poprzez wymnożenie wektora $\boldsymbol{Y_*}$ oraz pól wektora $\boldsymbol{X_*}$. Iloczyny te oraz dodatkowy składnik Z_* są sumowane za pomocą kaskady sumatorów RCA. Ponieważ na każdym poziomie drzewa liczba operandów jest redukowana o połowę, wymagana jest kaskada dwupoziomowa. Sygnałem wyjściowym drzewa sumatorów jest 14–bitowy wektor reprezentujący sumę H.

Z powodu ograniczeń sprzętowych maksymalna wartość argumentu subtraktora warunkowego nie powinna przekraczać $4 \cdot M_*$. Ponieważ zakres wartości sumy H jest znacznie większy, kolejny stopień układu dokonuje redukcji H do 9–bitowej sumy V. Wyższa część wektora $\boldsymbol{H}$ jest dzielona na dwa pola, 3– i 4–bitowe, dla których wyznaczane są reszty modulo 109 za pomocą pamięci ROM. Pamięć zawierająca reszty dla pola 3–bitowego jest wbudowana w sumator RCA. Obliczony sygnał V jest podawany na wejście subtraktora warunkowego złożonego z komparatorów porównujących ze stałymi 108 i 217 i sumatora. Zadaniem komparatorów jest sygnalizacja zakresu wartości V. Na rys. 3.7 komparatory wytwarzają dwa sygnały: $V > 108$ i $V > 217$. Sygnały te są podawane na wejście pamięci ROM zawierającej stałe 0, -109 i -218 w kodzie $U2$ dodawane do V. Po odjęciu takiej krotności 109 od V, aby otrzymany wynik należał do zakresu $[0, 108]$, jest on równy poszukiwanej



Rysunek 3.7. Przykład jednostki arytmetycznej mnożenia akumulacyjnego modulo 109.

wartości $|X_* \cdot Y_* + Z_*|_{109}$.

Proponowana ramowa struktura jednostek arytmetycznych pozwala na implementację wielu różniących się charakterystykami AT układów realizujących to samo działanie dla tego samego modułu. Pozwala to na wybór jednostki o parametrach dostosowanych do pozostałej części systemu, dzięki czemu w wielu przypadkach można uniknąć stosowania rozwiązań o największym obszarze bądź opóźnieniu. Z drugiej strony, duża liczba rozwiązań komplikuje algorytmy wyszukiwania jednostek o wymaganych parametrach.

Zaprojektowana struktura jednostek arytmetycznych składa się z trzech bloków połączonych kaskadowo. Struktura i parametry poszczególnych bloków zależą od decyzji podjętych przy konstrukcji bloków poprzedzających. Utrudnia to znacznie poszukiwania układów o założonych parametrach AT, tym bardziej, że formuły opisujące obszar i opóźnienie pojedynczego bloku są dość skomplikowane. Dodatkowo, struktura jednostki jest określona przez kombinacje kompozycji liczby bitów wektorów

wejściowych. Ponieważ liczba kompozycji zależy wykładniczo od liczby bitów argumentów, wraz ze wzrostem szerokości argumentów liczba możliwych konfiguracji jednostki arytmetycznej rośnie bardzo szybko. Konieczne jest więc znalezienie metody pozwalającej na szybkie i wygodne wytypowanie struktury jednostki o założonych parametrach. Ze względu na dużą liczbę możliwych rozwiązań, znalezienie struktury optymalnej może być niezmiernie trudne. Z drugiej strony, duża liczba struktur powoduje, że istnieje liczna grupa układów o parametrach zbliżonych do poszukiwanych.

Aby skrócić czas poszukiwania wybranej jednostki i odciążyć projektanta zaproponowano algorytm automatycznej generacji układu o parametrach AT mieszczących się w zadanych granicach. Zaprezentowany algorytm umożliwia wygenerowanie i oszacowanie charakterystyk AT jednostek arytmetycznych modulo o strukturze przedstawionej na rys. 3.1. Poprzez ograniczenie maksymalnej pojemności pamięci ROM w układzie wytwarzania iloczynów częściowych i generatorze wyniku nie są konstruowane jednostki o dużym obszarze. Dla wszystkich utworzonych jednostek są następnie szacowane ich charakterystyki AT, co umożliwia wstępny wybór grupy układów o poszukiwanych parametrach. Wybrana grupa układów jest następnie implementowana w docelowym układzie FPGA i na podstawie raportów po syntezie typowany jest układ o najlepszych charakterystykach AT.

Ponieważ w algorytmie konieczne jest sprawdzenie liczby struktur zależnej od liczby kompozycji szerokości operandów, złożoność algorytmu jest wykładnicza. Dla modułów o szerokościach do kilkunastu bitów liczba kombinacji nie przekracza jednak 10^5, co pozwala na szybkie uzyskanie wyników. Dla szerokich operandów wejściowych należy zastosować inne ograniczenia na strukturę UWIC ze względu na wzrost złożoność jednostek arytmetycznych, w których iloczyny częściowe są obliczane za pomocą pamięci ROM.

Rozdział 4

Implementacja

W rozdziale zaprezentowano rezultaty eksperymentów dotyczących implementacji układów arytmetyki resztowej w FPGA i wspomagania obliczeń w algorytmach oświetlenia globalnego. Wszystkie testy zostały przeprowadzone na komputerze wyposażonym w 1 GB pamięci RAM i procesor AMD Athlon^{TM}64 taktowany zegarem 2,5 GHz z zainstalowanym systemem Windows XP. W pierwszej części przedstawiono sposób implementacji algorytmu automatycznego generowania resztowych jednostek arytmetycznych. Druga część zawiera analizę parametrów resztowych jednostek arytmetycznych i ich porównanie z dotychczasowymi rozwiązaniami. W części trzeciej zaprezentowano parametry resztowych kanałów obliczeniowych skonstruowanych z użyciem HRNS zaproponowanych w rozdz. 2.3. W ostatniej części zawarto wyniki implementacji układów sprzętowego wspomagania AOG z wykorzystaniem arytmetyki resztowej.

4.1 Algorytm automatycznej generacji jednostek arytmetycznych

Zaprezentowany w rozdz. 3.2 algorytm automatycznej generacji resztowych jednostek arytmetycznych (alg. 10, str. 129) został zaimplementowany w języku C++. Powstały kod można skompilować zarówno w środowisku Visual Studio 2005, jak i kompilatorem g++.

Program generujący jednostki arytmetyczne tworzy ich opis w języku VHDL. Opis ten jest następnie jest implementowany w układzie FPGA XC2S200-6FG456 z użyciem środowiska Xilinx ISE Foundation 7.1i. Dodatkowo dla każdej wygenerowanej jednostki tworzony jest model zawierający pobudzenia testujące (ang. *testbench*). Model testujący pozwala automatycznie zweryfikować poprawność działania za pomocą symulatora ModelSim XE III 6.0a. Cały proces generowania i im-

plementacji jednostek jest realizowany za pomocą zestawu skryptów dokonujących automatycznej generacji, weryfikacji i implementacji układów dla zadanych parametrów M_* i l. Zadaniem użytkownika jest jedynie wybór najbardziej odpowiedniego rozwiązania.

Zaproponowany algorytm generacji jednostek arytmetycznych modulo miał ułatwić eksperymenty dotyczące wyboru modułów dla resztowego systemu liczbowego używanego do zwiększenia przepustowości układów mnożenia akumulacyjnego używanych w algorytmach oświetlenia globalnego. Ponieważ zakres dynamiczny poszukiwanego RNS nie przekracza kilkudziesięciu bitów, rozmiar pojedynczego modułu został ograniczony do 16 bitów. Ograniczenie to pozwala na użycie 32-bitowych liczb do reprezentacji wyników obliczeń pośrednich w prototypowym programie, co bezpośrednio przekłada się na wysoką szybkość działania oraz łatwość implementacji. Niestety, uniemożliwia to wygenerowanie poprawnie działających jednostek dla modułów $M_* \geq 2^{16}$.

Algorytm zaprezentowany w rozdz. 3.2 (alg. 10, str. 129) zwraca posortowany zbiór wszystkich wygenerowanych jednostek, z których każda charakteryzuje się zajmowanym obszarem A oraz opóźnieniem T. Charakterystyki używane w algorytmie nie obejmują narzutu na prowadzenie połączeń, który jest związany z konkretnym narzędziem używanym do syntezy i implementacji w strukturze FPGA. Wybór najlepszego układu dokonywany jest więc po pełnej implementacji kilku rozwiązań stanowiących najbliższe sąsiedztwo układu zwróconego przez algorytm jako najlepszy. Liczba dodatkowych układów jest określana przez użytkownika – w prototypowym rozwiązaniu wielkością wystarczającą było 10 dodatkowych jednostek.

4.1.1 Implementacja algorytmu

Kompletny proces generowania jednostek składa się z trzech etapów. Pierwszym z nich jest utworzenie opisów w języku VHDL jednostek z otoczenia najlepszego rozwiązania określonego przez alg. 10. W tym kroku generowane są także pliki zawierające modele testowe używane w kolejnym etapie do weryfikacji poprawności działania utworzonych jednostek. Błędy podczas weryfikacji mogą powstać jedynie w razie niewłaściwej implementacji algorytmu 10. Po weryfikacji modele testowe są usuwane, po czym przeprowadzany jest proces syntezy i implementacji wszystkich wygenerowanych układów. Następnie z powstałych raportów są wyodrębniane informacje dotyczące wykorzystania zasobów, z których tworzony jest raport końcowy. Na jego podstawie użytkownik dokonuje wyboru najbardziej odpowiedniej jednostki.

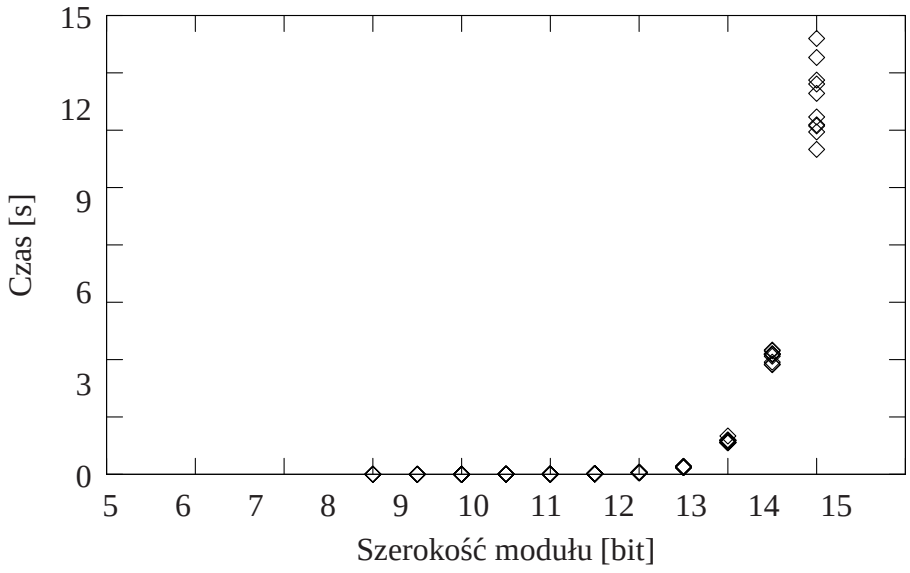
Efektywny w implementacji opis jednostek w języku VHDL wymaga stosowania konstrukcji języka jednoznacznie tłumaczonych na odpowiednie struktury sprzętowe. Dla każdego kompilatora VHDL istnieje zbiór wzorców umożliwiających jednoznaczny opis wielu podstawowych elementów układów cyfrowych: sumatorów, multiplekserów, przerzutników, pamięci, układów mnożących itp. Większość elementów użytych do konstrukcji jednostek resztowych zaprezentowanych w rozprawie może być opisana za pomocą gotowych konstrukcji dla używanego w rozprawie kompilatora firmy Xilinx. Wzorzec taki nie został jednak zdefiniowany przez producenta dla sumatora z wbudowaną pamięcią (rys. 3.5, str. 99).

Sumator z wbudowaną pamięcią wymaga wkomponowania pamięci w tablicę LUT będącą ogniwem sumatora RCA. Konieczne jest więc użycie opisu pozwalającego na zmianę funkcji realizowanej przez tablicę LUT w pojedynczym ogniwie sumatora. Może to zostać zrealizowane różnymi metodami. Sposób używany w pracy jest ściśle związany z używanym narzędziem, ale pozwala na prosty, wygodny i czytelny opis. Dzięki temu można uniknąć definiowania sumatora na poziomie pojedynczych ogniw budowanych z prymitywów dostępnych w stosowanej rodzinie FPGA. Na listingu 4.1 przedstawiono przykład kodu opisującego sumator dodający 7–bitowy wektor $\mathbf{X}$ i resztę $R = |y_9 \cdot 2^9 + y_8 \cdot 2^8 + y_7 \cdot 2^7|_{109}$ dla 3–bitowego pola. Kod z listingu 4.1 jest implementowany w sposób pokazany na rys. 3.5 zarówno przez używane środowisko, jak i jego wcześniejszą wersję Xilinx ISE Foundation 6.3i.

Listing 4.1. Opis sumatora z wbudowaną pamięcią w języku VHDL

```
with Y(9 downto 7) select
        R <=    "0000000" when "000",   -- 0 mod 109 = 0
                "0010011" when "001",   -- 128 mod 109 = 19
                "0100110" when "010",   -- 256 mod 109 = 38
                "0111001" when "011",   -- 384 mod 109 = 57
                "1001100" when "100",   -- 512 mod 109 = 76
                "1011111" when "101",   -- 640 mod 109 = 95
                "0000101" when "110",   -- 768 mod 109 = 5
                "0000010" when "111",   -- 896 mod 109 = 2
                "0000000" when others ;
        W <= ("0" & X) + R ;
```

Złożoność zaproponowanego algorytmu generacji resztowych jednostek arytmetycznych jest wykładnicza. Nie stanowi to jednak problemu dla modułów stosowanych w systemach DSP, gdyż dla modułów o szerokości kilkunastu bitów czas oczekiwania na wynik pozwala na pracę interakcyjną. Na rys. 4.1 przedstawiono zależność czasu wykonania prototypowej implementacji algorytmu od szerokości modułu dla danych jak na rys. 3.6. Czas wykonania był mierzony jako liczba cykli procesora AMD odczytywana instrukcją *rdtsc* [AMD06]. Przy pomiarach czasu wykonania fragment kodu odpowiedzialny za końcowe sortowanie wyników został usunięty. Sortowanie wyników przeprowadzane jest z użyciem algorytmu uogólnionego ze standardowej biblioteki wzorców (*ang. Standard Template Library, STL*) języka C++. Kompilacja została przeprowadzona z wyłączonymi wszystkimi opcjami kompilatora dotyczącymi optymalizacji. Należy tutaj zaznaczyć, że opisywana implementacja była tworzona z myślą o sprawdzeniu poprawności koncepcji dla małych modułów, w związku z czym w wielu punktach jest wysoce nieoptymalna, głównie z powodu nieefektywnego zarządzania pamięcią i powielania danych. Pomimo tych ograniczeń prototypowy program stanowi wydajne i wygodne narzędzie do badań związanych z doborem modułów do resztowych systemów liczbowych w zastosowaniach DSP.



Rysunek 4.1. Czas wykonania prototypowej implementacji algorytmu dla różnych modułów M_* w funkcji szerokości modułu $m_* = \lfloor \log_2(M_*) \rfloor + 1$ dla $l = 1$.

4.1.2 Parametry generowanych jednostek

Na rys. 4.2 przedstawiono parametry wygenerowanych przez opisywany algorytm jednostek mnożenia akumulacyjnego dla $l = 1$ oraz kilku wybranych wartości M_*. Wykresy nie obejmują jednostek o obszarze większym od 5000 tablic LUT z powodu ograniczeń stosowanych układów FPGA.

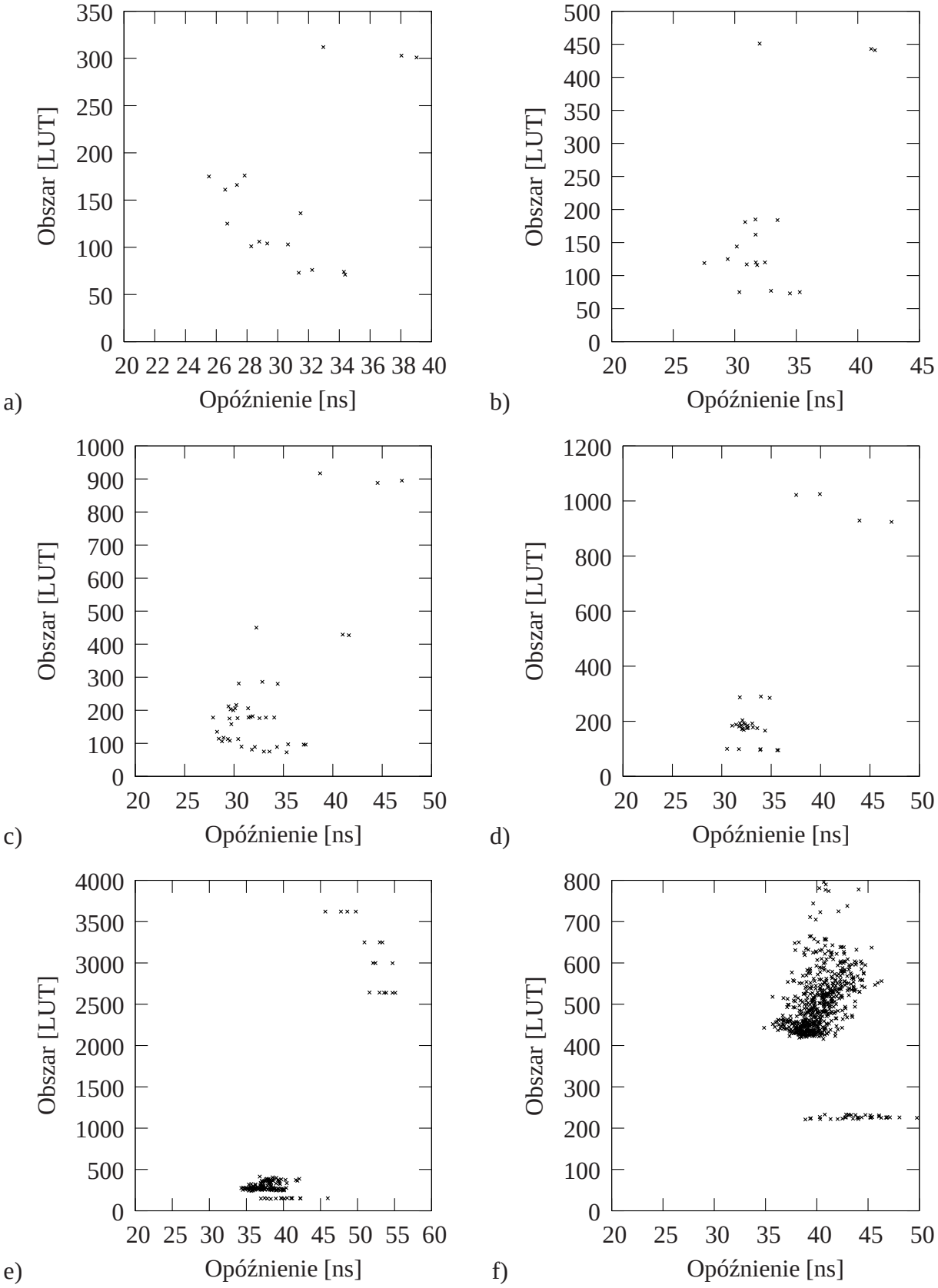
Każdy z przedstawionych na wykresach z rys. 4.2 zestawów jednostek wygenerowanych dla pojedynczego modułu można podzielić na trzy grupy. Pierwsza z nich obejmuje jednostki o minimalnym obszarze, które na wykresach tworzą poziomą linię położoną najniżej. W skład tej grupy wchodzą układy, w których iloczyny częściowe są tworzone wyłącznie za pomocą matryc mnożących $2 \cdot m_*$ lub $3 \cdot m_*$ bitów, w związku z czym struktura układu wytwarzania iloczynów częściowych jest jednakowa dla wszystkich elementów grupy. Różnice w długości ścieżki krytycznej wynikają ze struktury generatora wyniku.

Druga grupa zawiera jednostki, w których w bloku wytwarzania iloczynów częściowych znajdują się także pamięci ROM o niewielkiej pojemności. Jednostki z tej grupy zajmują obszar bezpośrednio nad układami z pierwszej grupy i stanowią najliczniejszy podzbiór wszystkich rozwiązań obejmujący układy, w których oba operandy mnożenia są podzielone na pola.

Ostatnią grupę, zajmującą prawy górny obszar wykresu, tworzą układy zbudowane z pamięci ROM o dużej pojemności. W układach tej grupy co najwyżej jeden z operandów mnożenia jest podzielony na pola, z których przynajmniej część jest szersza niż 3 bity, a drugi operand jest traktowany jako całość. Jednostki wchodzące w skład tej grupy charakteryzują się zarówno dużym obszarem, jak i opóźnieniem, co powoduje, że żadna z nich nie jest interesująca z punktu widzenia implementacji. Na rys. 4.2 e) oraz 4.2 f) jednostki z grupy trzeciej pominięto w celu zachowania czytelności wykresu w użytecznym zakresie.

Przynależność układu o najmniejszym obszarze do grupy pierwszej można dość łatwo uzasadnić. Obszar zajęty przez układ wytwarzania iloczynów częściowych i sumator wstępny jest sumą obszarów tych bloków. Obszar sumatora iloczynów częściowych zależy liniowo od liczby iloczynów częściowych, ponieważ na każdy iloczyn wymagany jest jeden sumator CPA. Obszar pojedynczego układu mnożącego $2 \cdot k$ lub $3 \cdot k$ bitów jest porównywalny z kosztem sumatora k–bitowego. Obszar układu wytwarzania iloczynów częściowych i sumatora wstępnego zależy od $\frac{m_*^2}{2}$ lub $\frac{m_*^2}{3}$, ponieważ układy mnożące tej postaci są stosowane wtedy, gdy tylko jeden z operandów jest dzielony na pola.

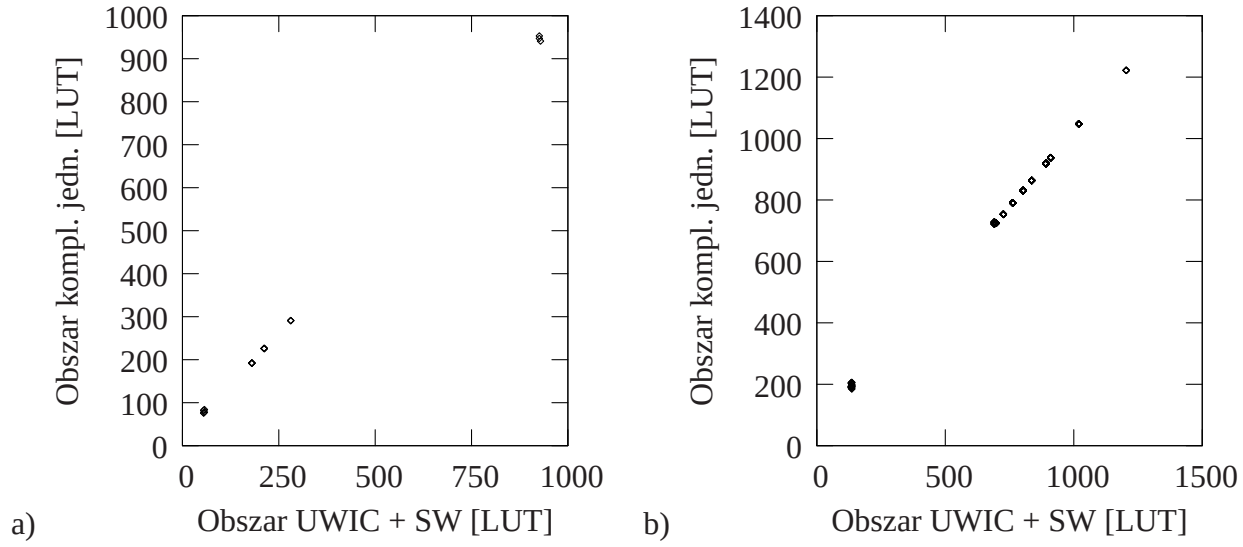
Dla układu wytwarzania iloczynów częściowych zbudowanego z pamięci ROM liczba tych pamię-



Rysunek 4.2. Zależność obszaru od opóźnienia dla a) $M_* = 47$, b) $M_* = 53$, c) $M_* = 67$, d) $M_* = 109$, e) $M_* = 509$ i f) $M_* = 2039$.

ci oraz obszar sumatora iloczynów częściowych zależy od iloczynu liczności dwóch podziałów. Dążenie do zmniejszenia liczby pól powoduje jednak wykładniczy wzrost obszaru zajmowanego przez pamięci ROM. Z tego powodu najmniejszy obszar zajmują układy, w których iloczyny częściowe są wytwarzane dla pól 2 lub 3–bitowych (lematy 3.1.5 i 3.1.6, str. 111). Pamięć adresowana słowem 4/5–bitowym wymaga 1 lub 2 tablice LUT na bit wyniku, zajmuje więc obszar porównywalny z sumatorem. Ponieważ jednak obszar układu wytwarzania iloczynów częściowych i sumatora wstępnego w tym przypadku zależy od $\left(\frac{m_*}{2}\right)^2 \cdot m_*$ lub $\left(\frac{m_*}{3}\right)^2 \cdot m_*$, układy z użyciem pamięci ROM są większe od struktur zawierających układy mnożenia $2/3 \cdot m_*$ bitów.

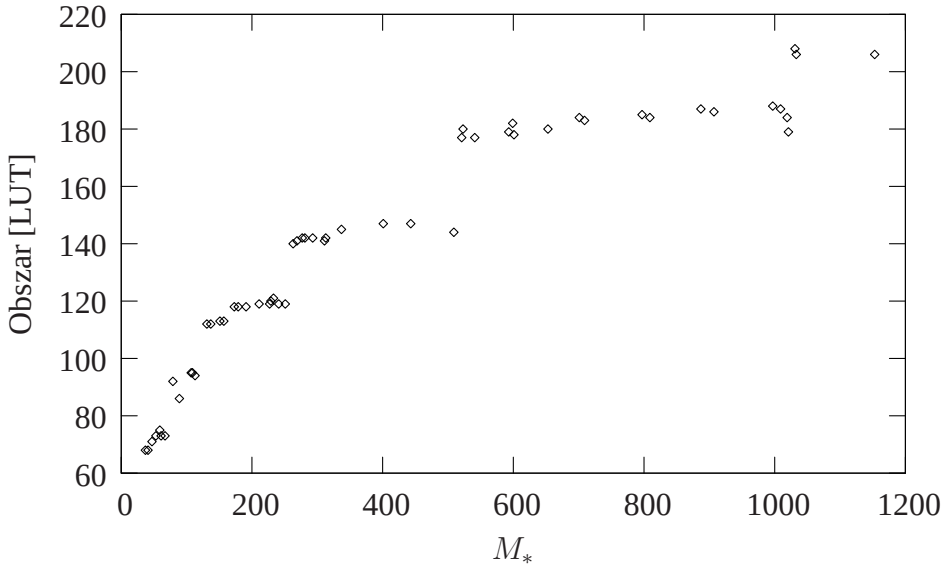
Analizując parametry AT wygenerowanych jednostek należy zwrócić uwagę na ważną cechę tworzonych jednostek pokazaną na rys. 4.3. Dotyczy ona liniowej zależności obszaru zajętego przez kompletną jednostkę od obszaru układu wytwarzania iloczynów częściowych i sumatora wstępnego. Przyczyną takiego stanu jest ograniczenie do 3 lub 4 bitów szerokości pól starszej części wektorów łączących kolejne reduktory modulo. Ograniczenia te powodują wyeliminowanie dużych pamięci ROM ze struktury generatora wyniku, dzięki czemu jego rozmiar jest zdecydowanie mniejszy od sumy obszaru pozostałych bloków. W związku z tym obszar całej jednostki jest zdominowany obszarem zajętym przez układ wytwarzania iloczynów częściowych i sumator wstępny.



Rysunek 4.3. Zależność obszaru całej jednostki od obszaru układu wytwarzania iloczynów częściowych (UWIC) i sumatora wstępnego (SW) dla a) $M_* = 67$ i b) $M_* = 1321$.

Granice parametrów AT

Spośród wszystkich wygenerowanych struktur układu można wybrać zestaw układów o różnej relacji zajmowanego obszaru i wprowadzanego opóźnienia. Parametry wygenerowanych układów są ograniczone od dołu przez dwa przypadki: układ o najmniejszym obszarze i układ o najmniejszym opóźnieniu. Na rys. 4.4 i 4.5 przedstawiono zależność minimalnego obszaru i długości ścieżki krytycznej w funkcji wartości modułu M_*. Można zauważyć, że parametry te są podobne w przypadku jednostek dla modułów tej samej szerokości. Właściwość ta jest szczególnie widoczna na rys. 4.4, niemniej można ją zaobserwować także dla układów z rys. 4.5.



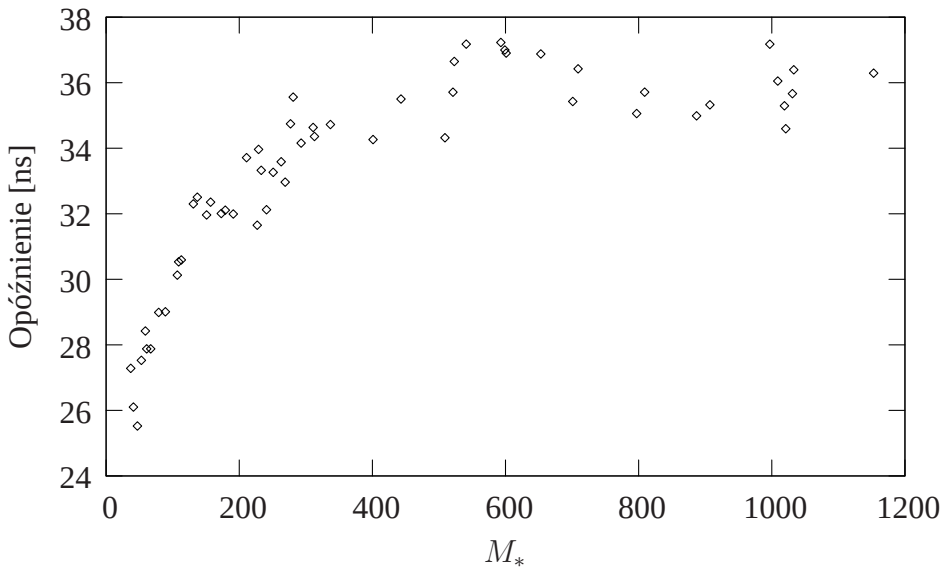
Rysunek 4.4. Minimalny obszar jednostki w funkcji modułu M_*.

Na rys. 4.6 przedstawiono charakterystyki AT jednostek o najmniejszym obszarze w funkcji szerokości modułu. Rysunek 4.6 a) zawiera zależność obszaru zajętego przez najmniejszą jednostkę od szerokości modułu. Dla zbadanego zakresu modułów funkcja ta może być ograniczona od góry przez $22 \cdot m_*$. Dokładniejsza interpolacja prowadzi do wielomianu piątego stopnia postaci

$$A(m_*) \approx -0.458 \cdot m_*^5 + 17.083 \cdot m_*^4 - 251.875 \cdot m_*^3 + 1837.916 \cdot m_*^2 - 6616.666 \cdot m_* + 9524. \quad (4.1)$$

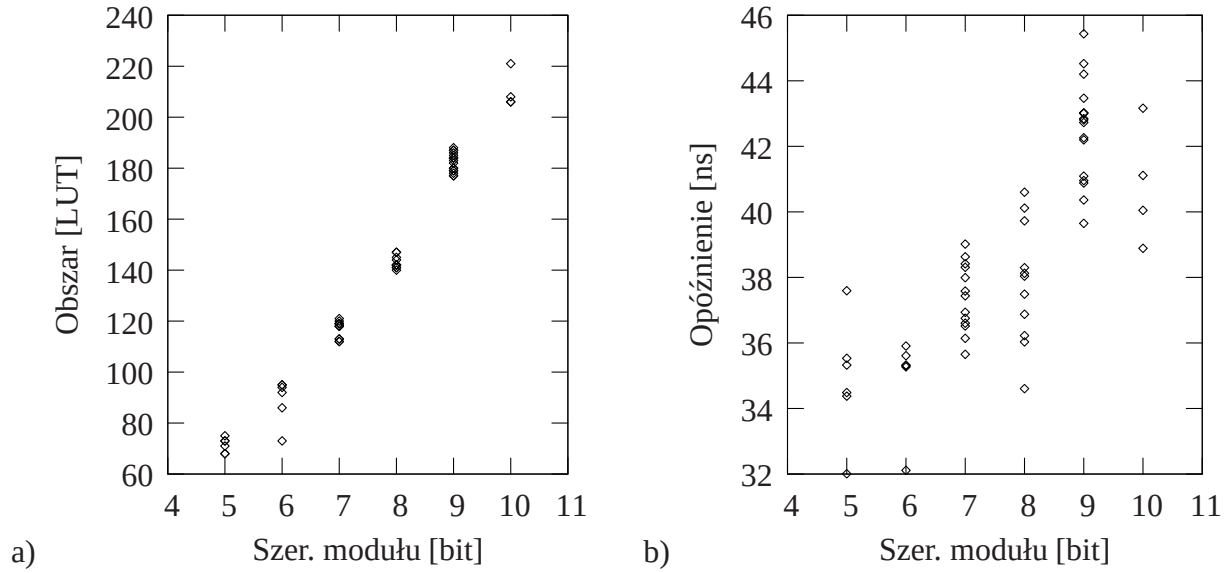
Warto zauważyć, że najmniejsze układy dla modułów o tej samej szerokości charakteryzują się podobnym obszarem ze względu na wspólne cechy struktury – jedyną różnicą jest postać generatora wyniku.

Długość ścieżki krytycznej dla układów o minimalnym obszarze przedstawiono na wykresie z rys. 4.6 b). Należy zwrócić uwagę na duży rozrzut opóźnień spowodowany różną liczbą poziomów



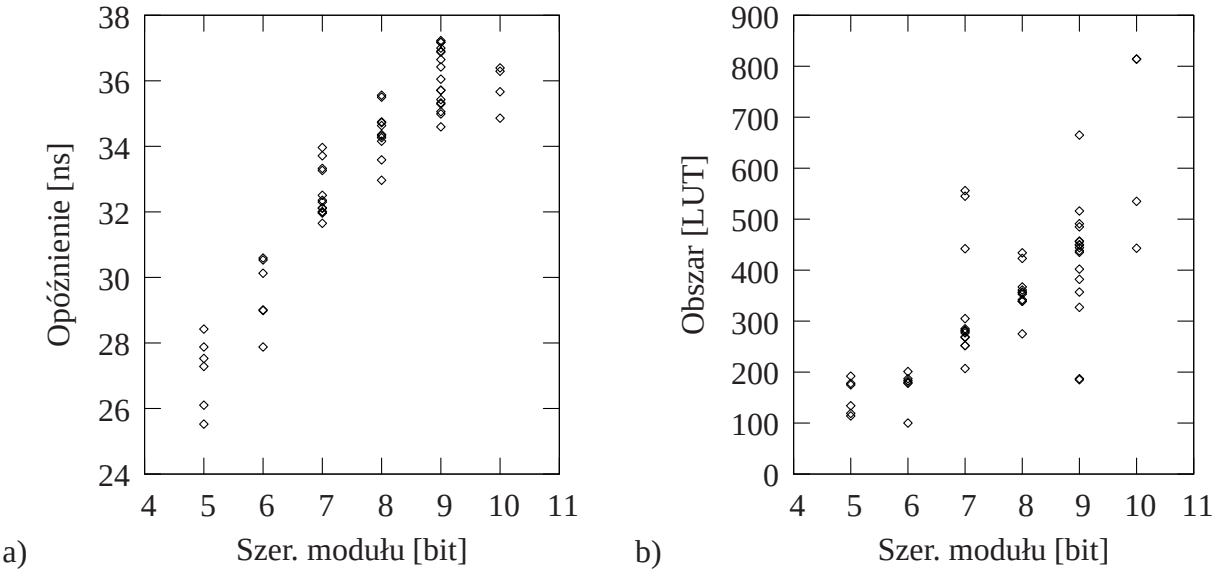
Rysunek 4.5. Minimalna długość ścieżki krytycznej jednostki w funkcji modułu M_*.

generatora wyniku, który w każdym przypadku jest rozwiązaniem o najmniejszym obszarze. Drugą cechą jednostek o najmniejszym obszarze są podobne wartości opóźnień dla sąsiednich szerokości modułu. Powodem takiego zachowania jest konieczność użycia układu mnożącego $3 \cdot k$ bity dla modułów o szerokościach nieparzystych, który to układ $3 \cdot k$ bity wnosi opóźnienie dwukrotnie większe od układu $2 \cdot k$ bity.



Rysunek 4.6. Zależność a) obszaru oraz b) opóźnienia od szerokości modułu dla układów o minimalnym obszarze.

Drugą grupę układów ograniczających parametry tworzonych jednostek są układy o najmniejszym opóźnieniu. Ich charakterystyki AT przedstawiono na rys. 4.7. Długość ścieżki krytycznej tych rozwiązań pokazano na rys. 4.7 a). Dla małych szerokości modułu wzrost długości ścieżki krytycznej w funkcji m_* jest dość szybki, jednak dla $m_* > 8$ bitów zaczyna uwidaczniać się logarytmiczny charakter funkcji. Związany jest on z coraz większym wpływem struktury kaskady sumatorów na opóźnienie wprowadzane przez układ.



Rysunek 4.7. Zależność a) opóźnienia oraz b) obszaru od szerokości modułu dla układów o minimalnym opóźnieniu.

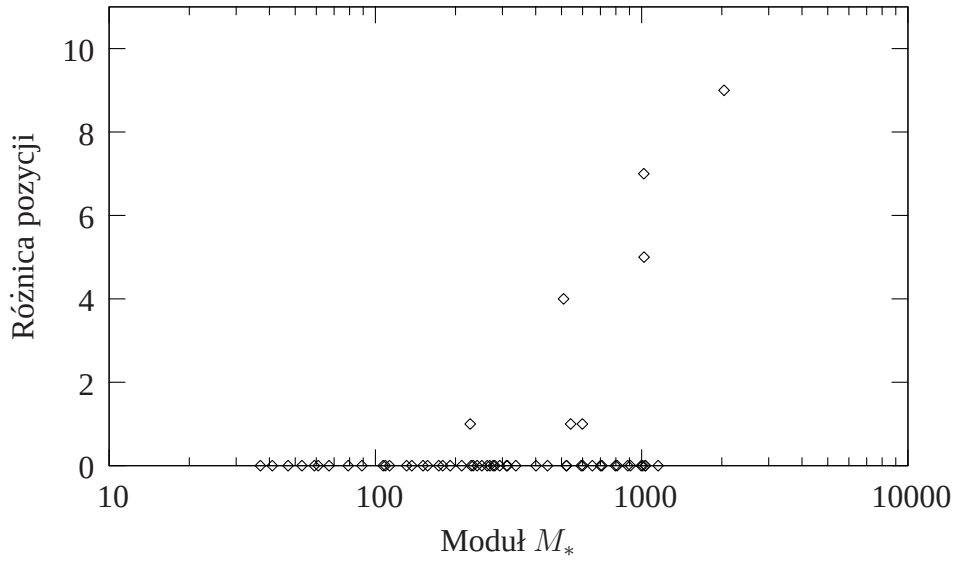
Obserwując wykres z rys. 4.7 a) można zauważyć znacznie mniejszy rozrzut czasów w ramach tej samej szerokości m_* w porównaniu z rys. 4.6 b). Niestety, kryterium najmniejszej długości ścieżki krytycznej powoduje konieczność stosowania pamięci ROM o dużej pojemności, która przekłada się bezpośrednio na zwiększenie zajmowanego obszaru. Dodatkowo, rozrzut zajmowanego obszaru w ramach tej samej szerokości modułu jest znaczny (rys. 4.7 b)), ponieważ dla używanych narzędzi do syntezy VHDL parametry AT pamięci ROM zależą także od jej zawartości.

Przeszukiwanie zbioru rozwiązań

Jednostki o parametrach zaprezentowanych na rys. 4.5 – 4.7 zostały wybrane po zaimplementowaniu wszystkich struktur wygenerowanych przez proponowany algorytm. Niestety, pomimo krótkich czasów potrzebnych na wygenerowanie zbioru jednostek (rys. 4.1), pełna implementacja wszystkich rozwiązań dla modułów kilkunastobitowych wymaga wielu godzin pracy komputera użytego do testów.

W części przypadków możliwe jest jednak znaczne ograniczenie liczby implementowanych jednostek poprzez wstępną selekcję na podstawie charakterystyk AT szacowanych w algorytmie (wzory (3.22) – (3.25) ze str. 109).

Układ o najmniejszym obszarze znajduje się w bardzo bliskim otoczeniu układu o najmniejszym obszarze oszacowanym w algorytmie. Wynika to z dużej dokładności oszacowań stosowanych w algorytmie dla układów, w których nie występują pamięci ROM o dużej pojemności. Dodatkowo, z powodu przynależności układu o najmniejszym obszarze do grupy jednostek, w których iloczyny częściowe są tworzone wyłącznie z użyciem układów mnożenia $2 \cdot k$ lub $3 \cdot k$ bitów, można znacznie ograniczyć w samym algorytmie liczność zbioru badanych układów dla problemu polegającego na wytypowaniu jednostki najmniejszej. Na rys. 4.8 przedstawiono zależność odległości jednostki najmniejszej po implementacji i jednostki najmniejszej w zbiorze posortowanym według parametrów określonych w algorytmie. W większości przypadków jest to ten sam układ, a dla wszystkich zbadanych wartości modułu układ o najmniejszym obszarze znajdował się w odległości co najwyżej 10 pozycji od układu wytypowanego przez algorytm.



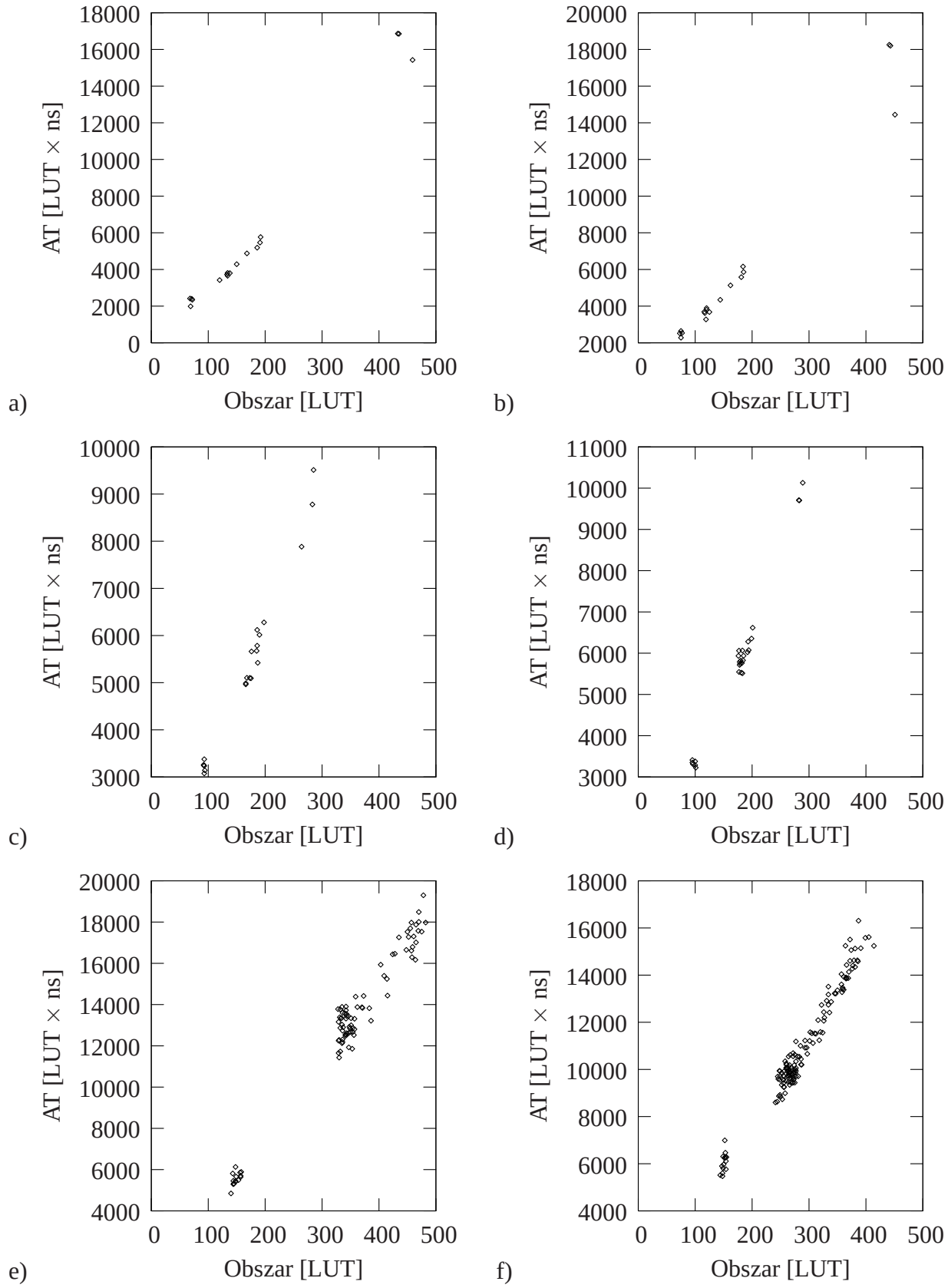
Rysunek 4.8. Pozycja jednostki o najmniejszym obszarze po implementacji względem pozycji układu wybranego przez algorytm w funkcji modułu M_*.

Niekorzystną właściwością wygenerowanych zbiorów jednostek jest zróżnicowanie kształtu krzywej opisującej zależność minimalnego opóźnienia od zajmowanego obszaru. Dla modułów przedstawionych na rys. 4.2 układ o najkrótszej ścieżce krytycznej znajduje się w grupie pierwszej (rys. 4.2

d)), w zbiorze największych lub najmniejszych układów z grupy drugiej (rys. 4.2 a) lub 4.2 b), e) i f)), lub w bliżej nieokreślonym podzbiorze grupy drugiej (rys. 4.2 c)). W związku z tym wytypowanie podzbioru, w którym może się znajdować układ o najmniejszym opóźnieniu, jest niemożliwe na podstawie informacji o obszarze zajmowanym przez daną jednostkę lub o jego położeniu względem pozostałych. Można jedynie zauważyć, że w dużej części przypadków układ najszybszy znajduje się w grupie układów zbudowanych z użyciem niewielkich pamięci ROM. Z tego powodu znalezienie układu o najmniejszej długości ścieżki krytycznej wymaga zbadania dużej liczby jednostek.

Niemożność szybkiego znalezienia układu o najmniejszym opóźnieniu nie utrudnia jednak wyznaczenia układu o najmniejszym iloczynie AT. Analizując wykresy przedstawione na rys. 4.9 można zauważyć, że układ o najmniejszym iloczynie AT znajduje się w zbiorze układów najmniejszych, czyli z grupy pierwszej. Właściwość ta jest łatwo wytłumaczalna - stosunek obszaru pomiędzy układami z grupy pierwszej i drugiej wynosi co najmniej 2, a spadek długości ścieżki krytycznej, o ile występuje, jest rzędu kilkudziesięciu (dla małych modułów) do kilkunastu procent. Stosunek rozmiaru wynika z charakteru zależności opisujących obszar układu wytwarzania iloczynów częściowych dla różnych metod obliczania iloczynów częściowych.

Zmniejszenie długości ścieżki krytycznej układów budowanych z użyciem małych pamięci ROM ma dwie podstawowe przyczyny. Pierwszą z nich jest uproszczenie generatora wyniku wskutek zmniejszenia szerokości wektora wyjściowego sumatora wstępnego. Drugą przyczyną jest zastąpienie sumatorów RCA pojedynczymi tablicami LUT implementującymi pamięci ROM. Dzięki temu ze ścieżki krytycznej zostają wyeliminowane opóźnienia związane z dodatkowymi bramkami XOR (wyznaczającymi bity sumy) i propagacją przeniesień. W sumatorze RCA do opóźnienia wprowadzanego przez LUT należy dodać opóźnienia związane z dodatkowymi bramkami XOR oraz propagacją przeniesień. Należy jednak zaznaczyć, że wzrost szybkości spowodowany różnicami w strukturze układów generujących iloczyny częściowe jest kompensowany opóźnieniami związanymi z koniecznością sumowania większej ich liczby. Dla szerokich modułów opóźnienie spowodowane liczbą wytwarzanych iloczynów częściowych może przekroczyć zyski wynikające z użycia pamięci ROM zamiast układów mnożenia $2 \cdot k$ i $3 \cdot k$ bitów. Oczywiście liczba iloczynów częściowych może zostać zmniejszona na rzecz zwiększenia rozmiaru pamięci ROM, ale wtedy opóźnienie pojedynczej pamięci rośnie z logarytmem szerokości słowa adresowego.



Rysunek 4.9. Zależność iloczynu AT od obszaru dla a) $M_* = 37$, b) $M_* = 53$, c) $M_* = 79$, d) $M_* = 107$, e) $M_* = 263$ i f) $M_* = 509$.

4.2 Jednostki arytmetyczne w strukturach FPGA

Poniżej zaprezentowano porównanie charakterystyk resztowych jednostek arytmetycznych o strukturze zaproponowanej w rozprawie z rozwiązaniami znanymi z literatury. Porównanie przeprowadzono dla układów mnożących, ponieważ są one z reguły najkosztowniejszym elementem jednostek MAC. Pozwoli to także na przebadanie szerszej klasy układów, gdyż dla niektórych jednostek mnożących nie istnieją proste metody poszerzenia o możliwość dodania dodatkowego składnika.

Porównanie parametrów jednostek konstruowanych według algorytmu zaproponowanego w rozprawie (alg. 10, str. 129) zostało przeprowadzone z następującymi rozwiązaniami: układami mnożenia opartymi o transformację grupy multiplikatywnej do grupy addytywnej [MFAA98], układami mnożenia z korekcją modulo [Beu03] oraz układami mnożenia opartymi o prawo różnicy kwadratów [MBGT01]. Ponieważ każdy z autorów podaje różne miary dotyczące parametrów opisywanych układów, dla każdej z metod zostały wygenerowane opisy jednostek w języku VHDL, po czym jednostki te zostały zaimplementowane w układzie XC2S200-6FG456 z użyciem narzędzi wspomnianych na początku rozdziału. Przedstawione wyniki dotyczą parametrów wygenerowanych w raportach po syntezie i implementacji w układzie FPGA.

Podstawowym blokiem układów opisanych w pracach [MFAA98], [Beu03], [MBGT01] są pamięci ROM. Efektywna implementacja takich pamięci jest podstawowym warunkiem uzyskania wiarygodnych wyników. W implementowanych układach zawartość pamięci ROM uwzględnia wyłącznie te kombinacje bitów adresowych, które wystąpią podczas mnożenia operandów z zakresu $[0, M_* - 1]$. Warunek ten dotyczy zarówno układów konstruowanych według dotychczasowych metod, jak i według alg. 10. Ograniczenie liczby bitów zawartych w pamięci ROM powoduje, że stosowany kompilator VHDL może wygenerować strukturę o mniejszym obszarze i opóźnieniu w stosunku do pamięci, w której wszystkie kombinacje bitów adresowych są wykorzystane.

Układy mnożące dla każdej z porównywanych metod zostały zaimplementowane dla tego samego zbioru modułów. Ponieważ metoda mnożenia modulo wykorzystująca transformację do grupy addytywnej [MFAA98] może być stosowana wyłącznie dla modułów będących liczbami pierwszymi, tylko takie moduły zostały wzięte po uwagę. W tabeli 4.1 przedstawiono zbiór modułów, dla których zostały przeprowadzone testy porównawcze. Zbiór ten nie obejmuje modułów szerszych niż 11 bitów z powodu wykładniczej zależności obszaru dotychczasowych rozwiązań od szerokości modułu. Rozwiązania te wykorzystują większość zasobów używanej matrycy FPGA dla modułów z tab. 4.1.

Tabela 4.1. Moduły dla implementowanych resztowych jednostek arytmetycznych.

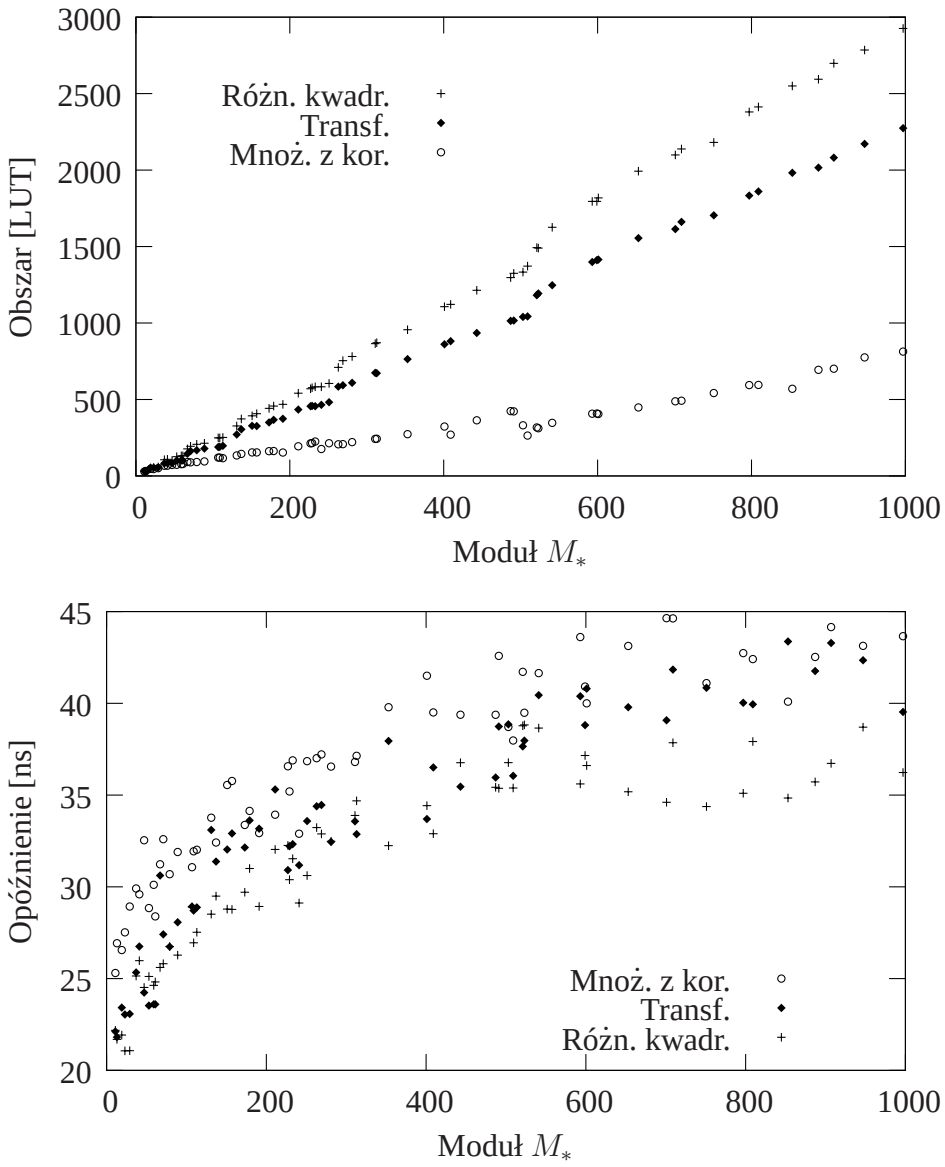
m_*	Wartość modułu M_*												
4	11	13											
5	19	23	29										
6	37	41	47	53	59	61							
7	67	71	79	89	107	109	113						
8	131	137	151	157	173	179	191	211	227	229	233	241	251
9	263	269	281	311	313	353	401	409	443	487	491	503	509
10	521	523	541	593	599	601	653	701	709	751	797	809	853
	887	907	947	997	1009	1019	1021						
11	1031	1033	1153	1163	1291								

4.2.1 Charakterystyki AT dotychczasowych rozwiązań

Na rys. 4.10 – 4.13 przedstawiono porównanie parametrów AT jednostek z prac [MFAA98], [Beu03], [MBGT01]. Część wykresów pozwala na dokładniejszą ocenę jednostek dla modułów 4, 5 i 6–bitowych. Wymienione metody są rzadko stosowane dla większych modułów ze względu na duży obszar układu. Charakterystyki dla pełnego zakresu zmienności modułu stanowią przede wszystkim materiał ilustrujący ogólne tendencje dla porównywanych metod. W praktycznych układach wspomniane metody są rzadko stosowane dla modułów o dużej wartości.

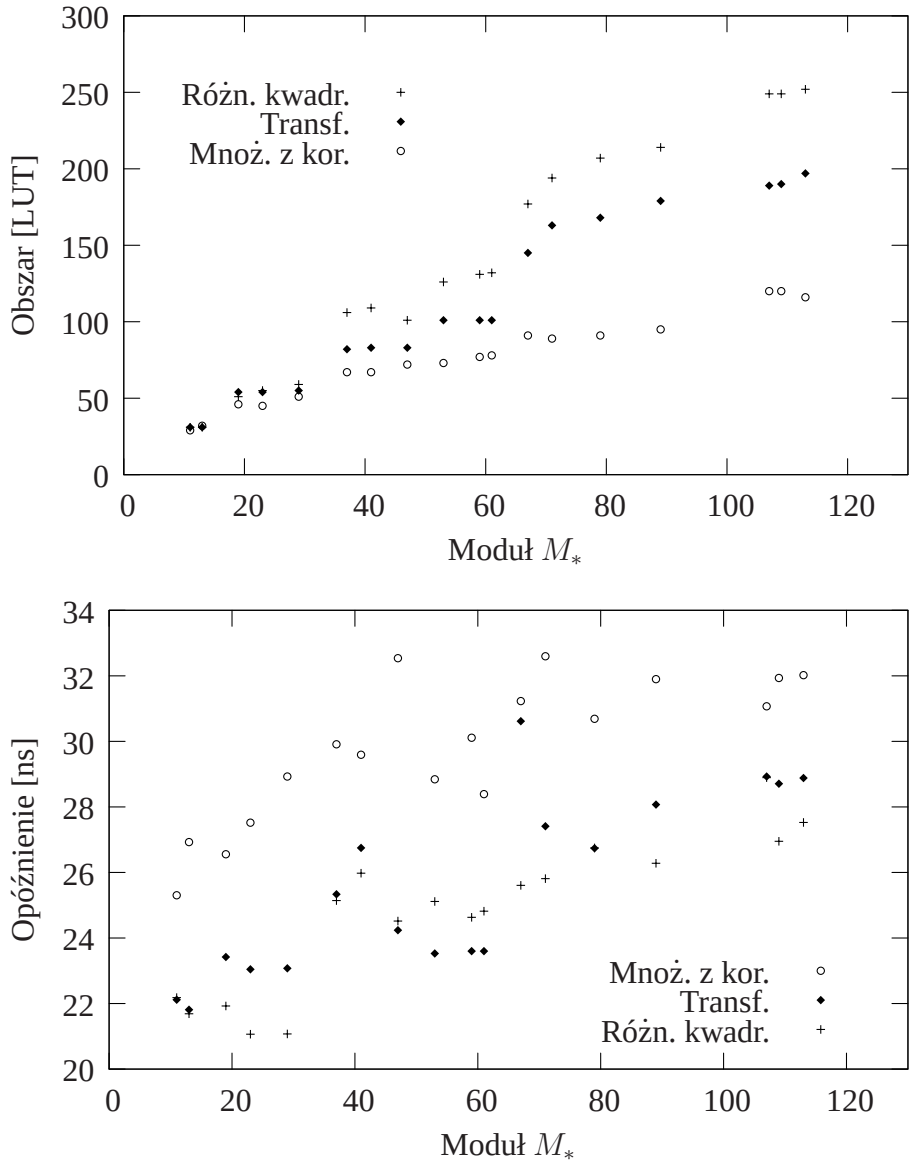
Układy mnożenia z korekcją modulo zaprezentowane w pracy [Beu03] charakteryzują się najmniejszym obszarem w całym zakresie wartości modułu, stanowią jednak grupę o najdłuższej ścieżce krytycznej. Układami najszybszymi w większości przypadków są struktury wykorzystujące prawo różnicy kwadratów opisane w [MBGT01], zajmują one także największy obszar. Jednostki realizujące mnożenie przez dodawanie w izomorficznej grupie addytywnej proponowane w [MFAA98] są układami o nieco mniejszym obszarze od układów z [MBGT01] przy większych opóźnieniach. Wyjątkiem są układy dla modułów 5–bitowych, gdzie rozwiązanie z [MFAA98] wprowadza najmniejsze opóźnienia. Rozwiązania z prac [MBGT01] i [MFAA98] charakteryzują się podobną wartością iloczynu AT, jednak stosowanie układów wykorzystujących transformację na izomorficzną grupę addytywną wiąże się z koniecznością spełnienia szeregu ograniczeń.

Porównanie iloczynów AT i AT^2 pokazuje, że podstawowym czynnikiem wpływającym na ich wartość są obszary zajmowane przez poszczególne jednostki. Charakter zależności iloczynów AT i

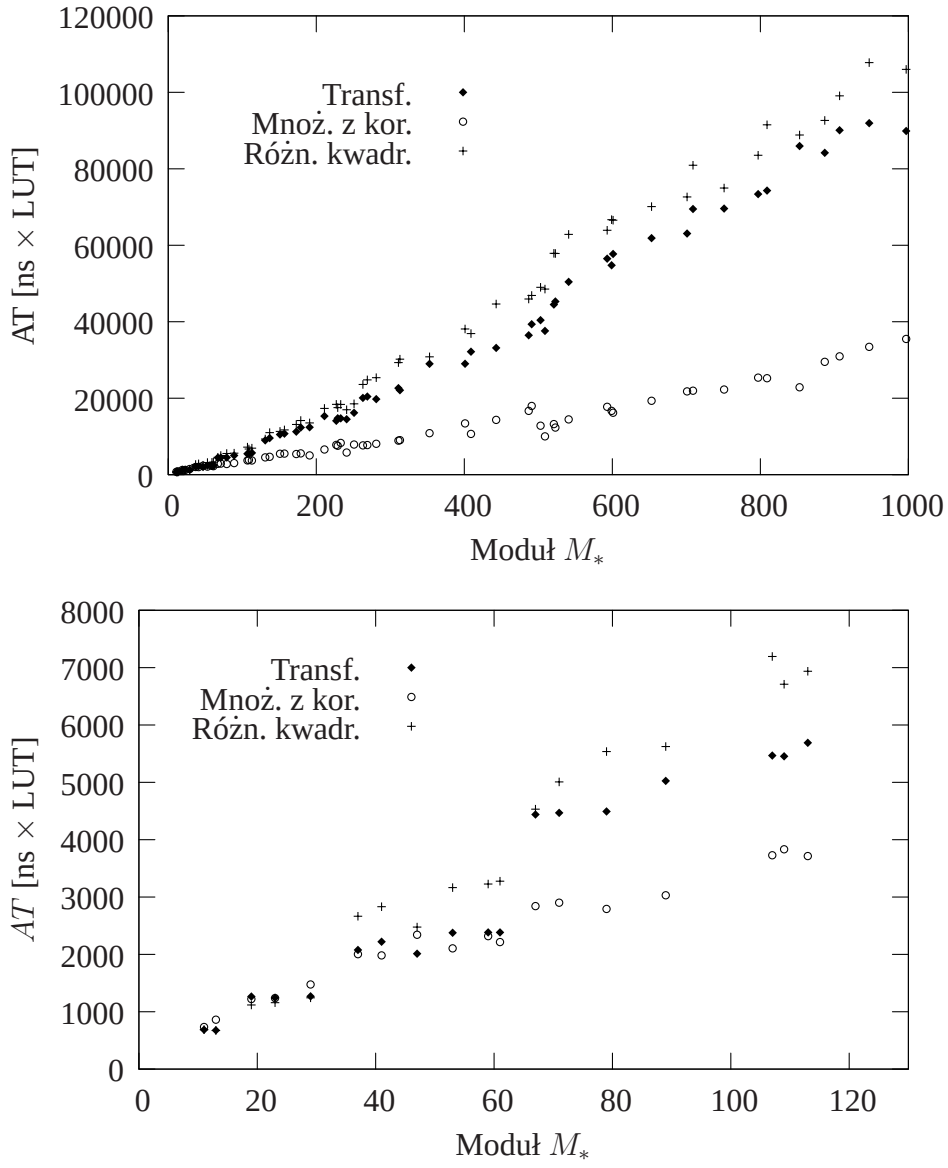


Rysunek 4.10. Porównanie obszaru i opóźnienia dla różnych struktur układów mnożenia modulo.

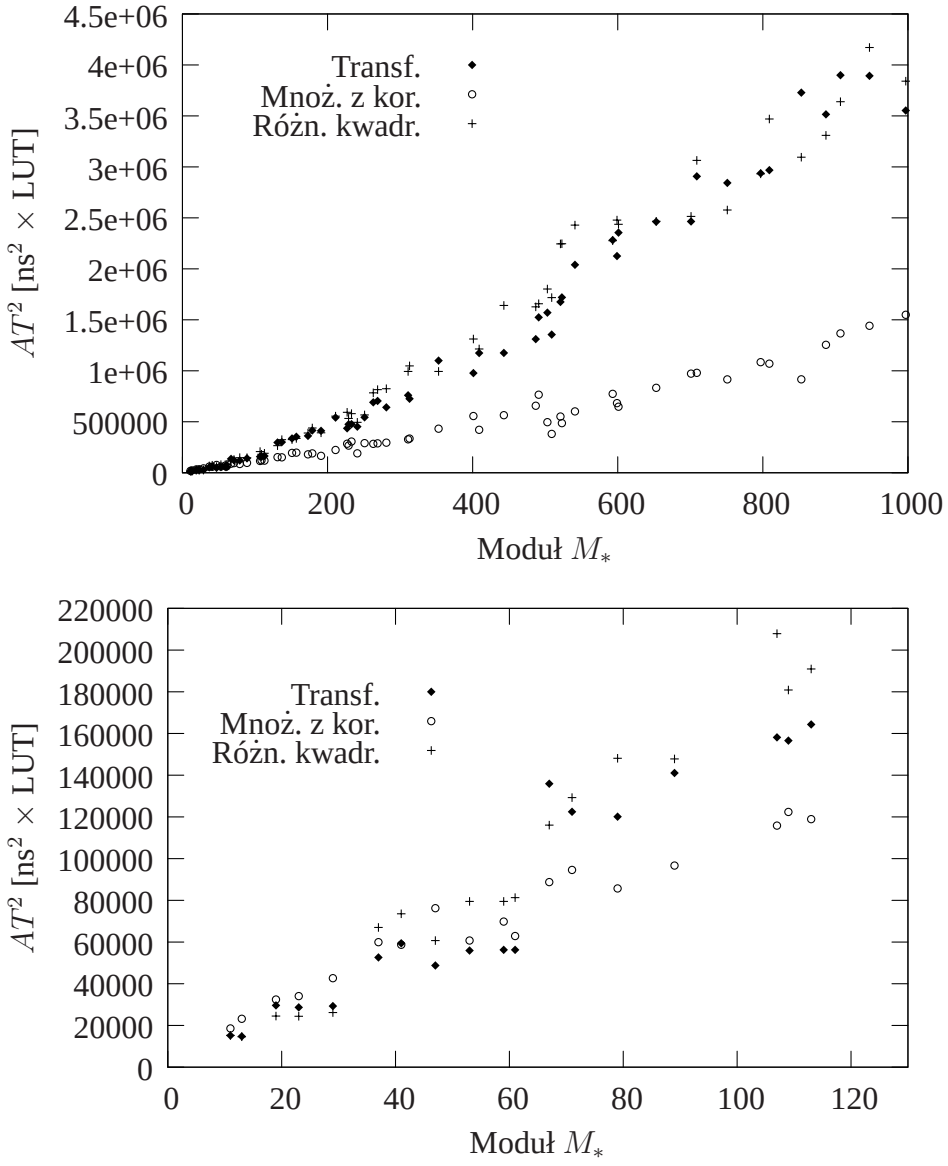
AT^2 od wartości M_* jest bardzo podobny za wyjątkiem modułów najmniejszych ($M_* < 2^5$). Podobieństwo charakterystyk z rys. 4.12 i 4.13 do charakterystyk z rys. 4.10 wynika z niewielkich różnic w długości ścieżki krytycznej w stosunku do różnic w zajmowanym obszarze.



Rysunek 4.11. Porównanie obszaru i opóźnienia dla różnych struktur układów mnożenia modulo.



Rysunek 4.12. Porównanie iloczynu AT dla różnych struktur układów mnożenia modulo.



Rysunek 4.13. Porównanie iloczynu AT^2 dla różnych struktur układów mnożenia modulo.

4.2.2 Charakterystyki AT nowych jednostek

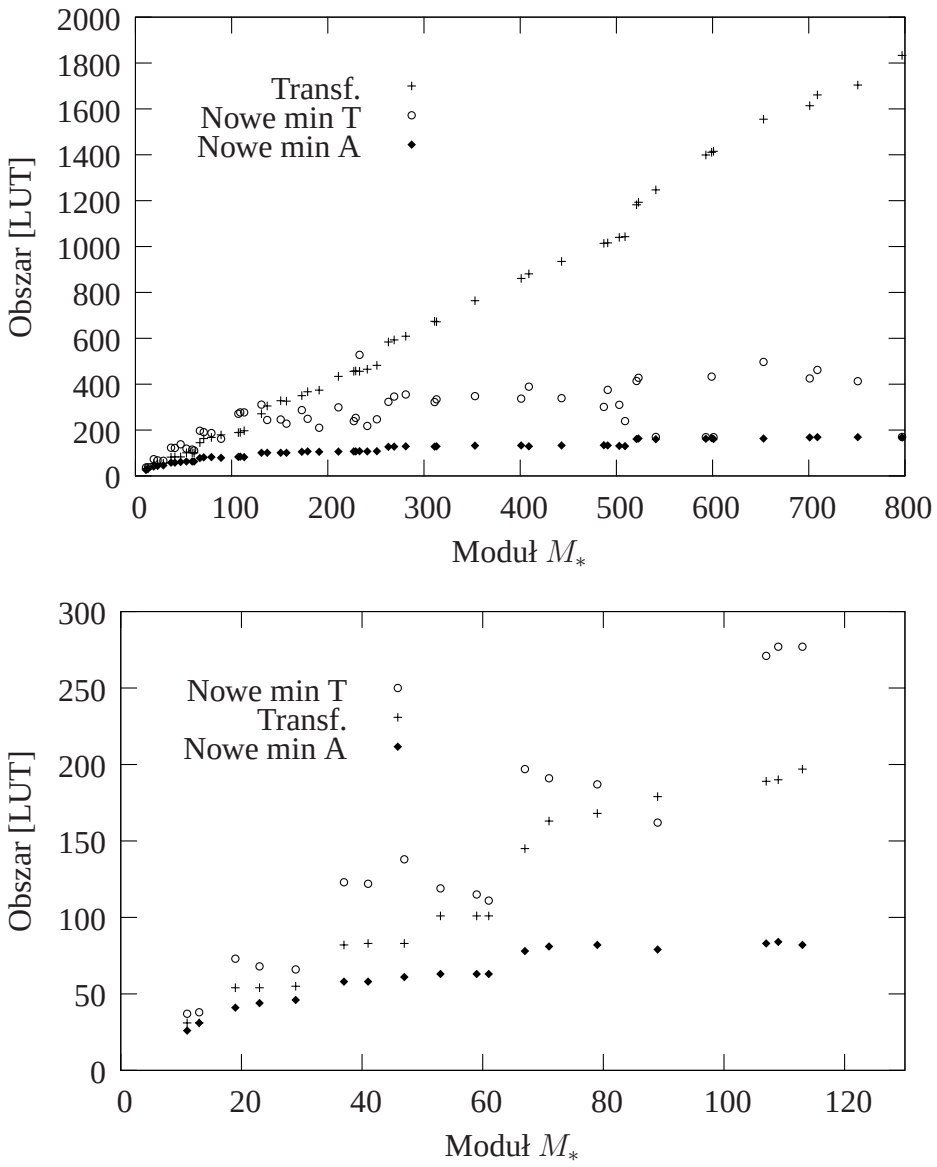
Analiza wpływu struktury jednostek o strukturze ramowej przedstawionej na rys. 3.1 na ich charakterystyki AT jest podstawą ograniczeń liczby rozwiązań w algorytmie automatycznego generowania tych jednostek. Z tego powodu częściowa analiza parametrów AT jednostek arytmetycznych jest zamieszczona w rozdz. 4.1.2. Poniżej zamieszczono porównanie obszaru i opóźnienia jednostek o strukturze zaproponowanej w rozprawie z układami zaprezentowanymi w [MFAA98], [Beu03] i [MBGT01].

Porównanie z układami mnożącymi wykorzystującymi transformację na izomorficzną grupę addytywną

Schemat opisanego w pracy [MFAA98] układu mnożenia wykorzystującego transformację grupy multiplikatywnej na izomorficzną grupę addytywną znajduje się na rys. 2.12 na str. 54. Układ ten składa się z 3 pamięci ROM adresowanych słowem o szerokości modułu i sumatora modulo $M_* - 1$. Dwie z trzech pamięci pracują równolegle, tak więc opóźnienie sygnału jest równe sumie opóźnień wnoszonych przez dwie pamięci i sumator modulo. O ile nie istnieje możliwość konstrukcji pamięci ROM jako jednostek potokowych, maksymalna głębokość potoku w tym układzie wynosi 3 lub 4 etapy, zależnie od postaci sumatora modulo. Istotną wadą takiej struktury układu mnożącego jest ograniczenie wartości modułu do liczb pierwszych.

Na rys. 4.14 przedstawiono porównanie obszaru zajmowanego przez ten układ mnożący z obszarem jednostek proponowanych w rozprawie. W całym zakresie wartości modułu M_* najmniejsze układy o strukturze z rys. 3.1 zajmują mniejszy obszar niż układy wykorzystujące transformację. Dodatkowo, dla modułów $M_* > 128$ także układy najszybsze o nowej strukturze zajmują mniejszy obszar. Jedynie dla modułów $M_* < 128$ jednostki wykorzystujące transformację mogą być mniejsze od najszybszych rozwiązań według nowej koncepcji. Największe różnice są widoczne dla wartości modułu nieznacznie przekraczającej 2^k.

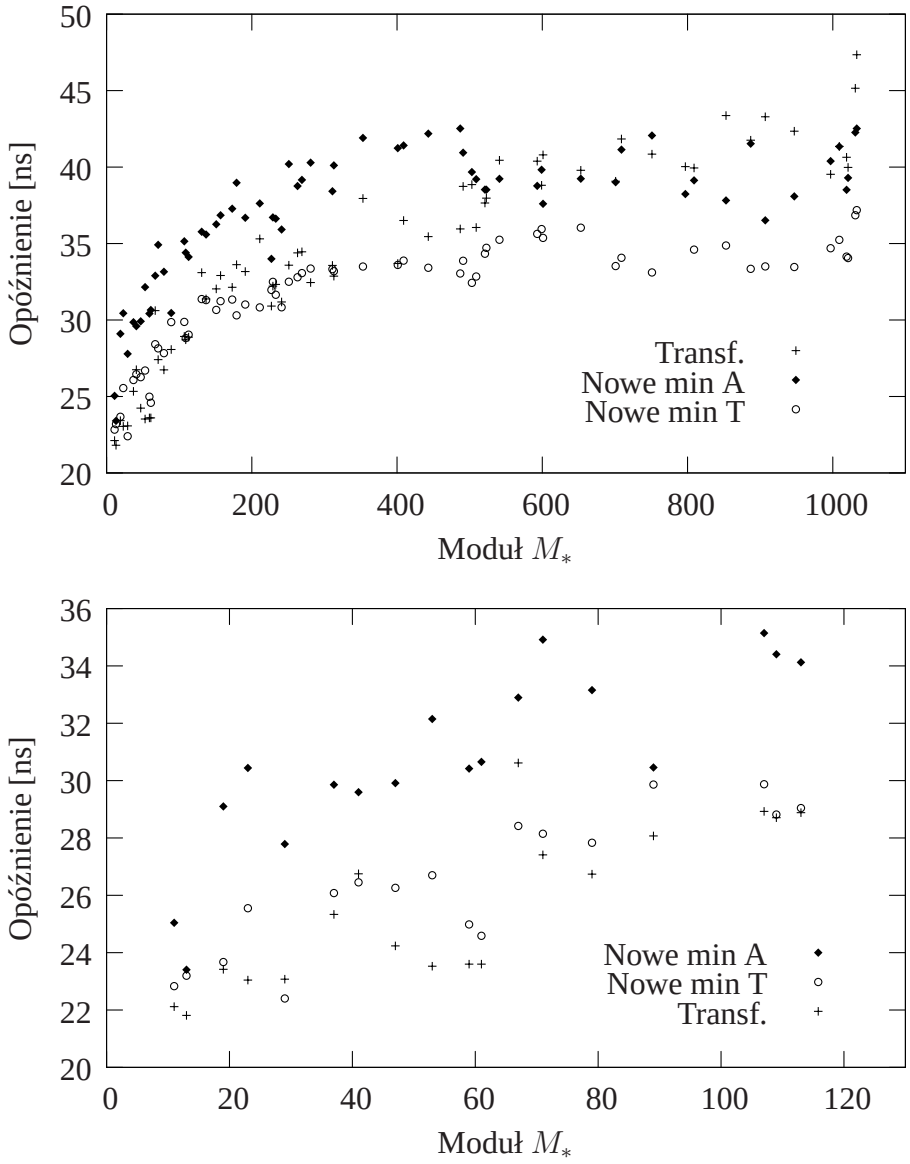
Przyczyną takiego zachowania jest różnica charakteru funkcji opisujących obszar w funkcji M_* dla porównywanych układów. Obszar układu opartego o transformację jest zdominowany przez obszar zajęty przez pamięci ROM. Ponieważ pamięci te zawierają liczbę bitów zależną od M_*, obszar przez nie zajmowany rośnie liniowo z wartością M_*. Dla układów o nowej strukturze rozmiar pamięci ROM jest znacznie ograniczony, dlatego też obszar całego układu dla modułów o tej samej szerokości



Rysunek 4.14. Porównanie zajmowanego obszaru z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną.

jest podobny. Funkcja opisująca zależność obszaru układów proponowanych w rozprawie jest funkcją schodkową, dlatego też różnica jej wartości i funkcji liniowej przyjmuje największe wartości w punktach nieciągłości, czyli dla M_* bliskich 2^k.

Na wykresach z rys. 4.15 przedstawiono porównanie wprowadzanego opóźnienia dla układów mnożących konstruowanych z wykorzystaniem transformacji i układów proponowanych w rozprawie. Najszybsze z nowych struktur mają krótszą ścieżkę krytyczną dla modułów $M_* > 512$. W przypadku modułów z zakresu $(2^7, 2^9)$ opóźnienie jednostek mnożących wykorzystujących transformację przyjmuje wartości pośrednie pomiędzy długością ścieżki krytycznej układów najszybszych i

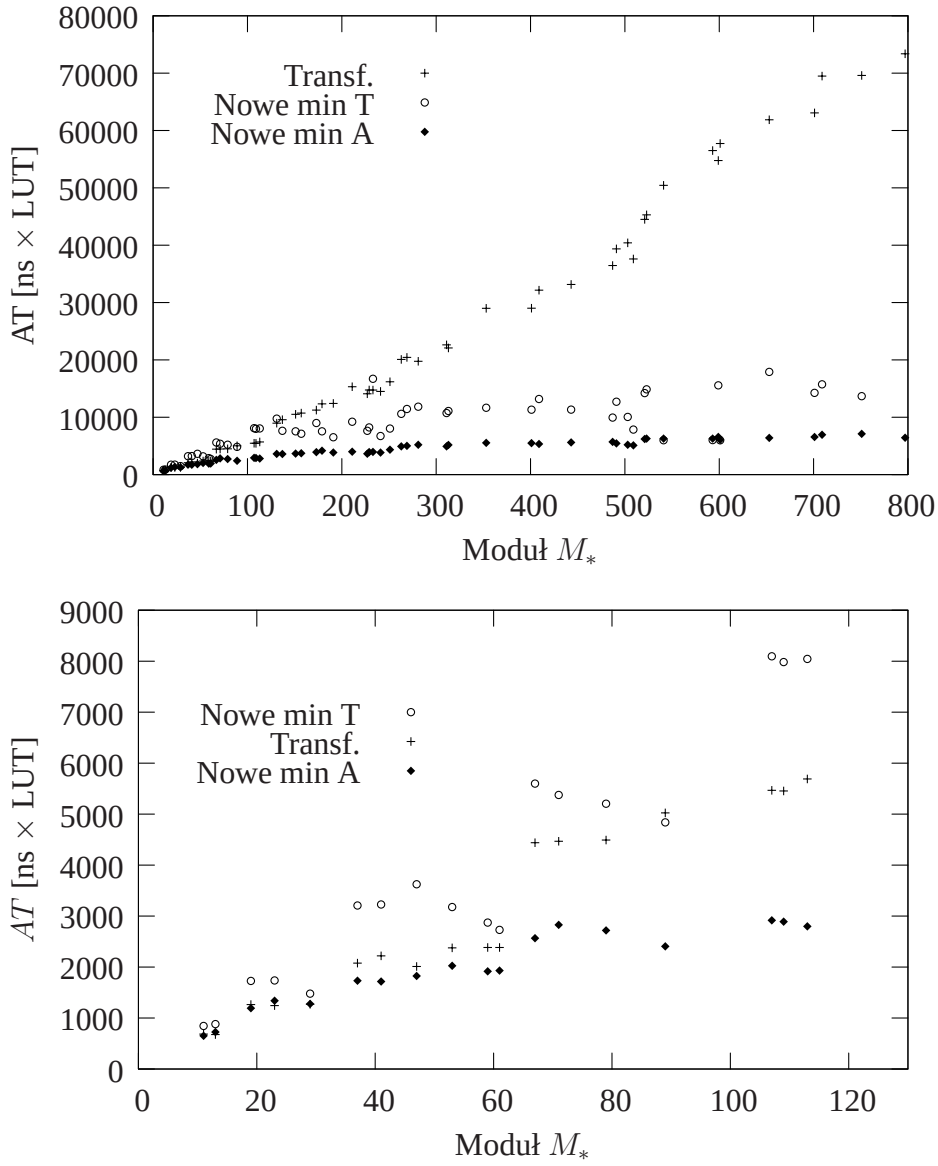


Rysunek 4.15. Porównanie długości ścieżki krytycznej z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną.

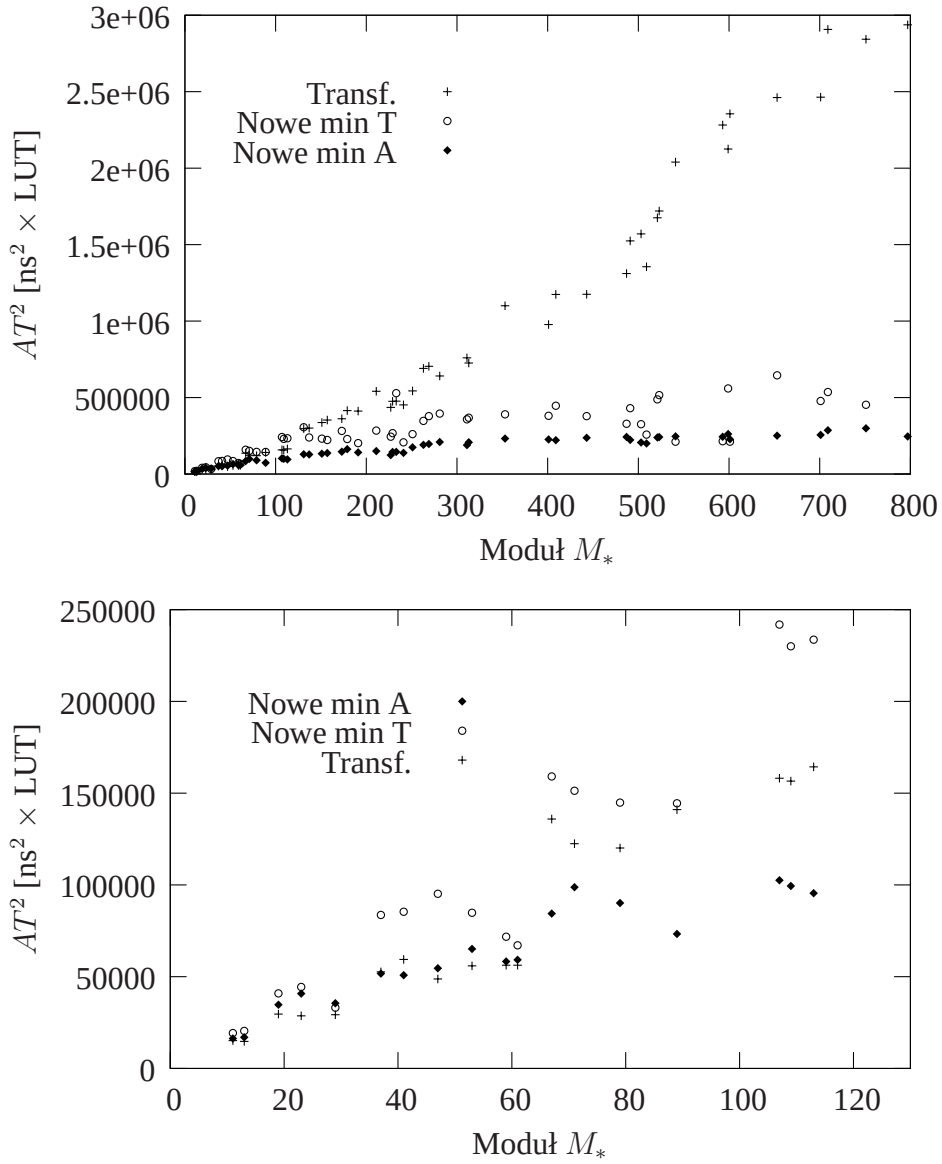
najmniejszych według nowej koncepcji. Ponieważ obszar dla układów według [MFAA98] jest większy niż proponowanych, stosowanie w tym zakresie jednostek mnożących wykorzystujących transformację jest nieopłacalne. Podobna sytuacja występuje dla modułów $M_* > 512$, gdzie układy z pracy [MFAA98] są zarówno wolniejsze, jak i większe od nowych jednostek. Jedynie dla modułów $M_* < 128$ najszybsze z nowych struktur są do 10% wolniejsze.

Na rys. 4.16 i 4.17 przedstawiono porównanie iloczynów AT i AT^2 dla opisywanych jednostek. Wykresy te są podobne do wykresu zawierającego zależność obszaru od M_* ze względu na podobne wartości opóźnień dla porównywanych układów mnożących. Podsumowując, stosowanie jednostek

mnożących według koncepcji z pracy [MFAA98] jest korzystne wyłącznie wtedy, gdy dla modułów $M_* < 128$ wymagane są krótkie ścieżki krytyczne. W pozostałych przypadkach jednostki o strukturze z rys. 3.1 charakteryzują się lepszymi parametrami AT. Dodatkowymi zaletami struktur prezentowanych w rozprawie są możliwość głębszego potokowania, brak konieczności ograniczenia wartości modułu M_* do liczb pierwszych i łatwość poszerzenia układu o dowolną liczbę sumatorów.



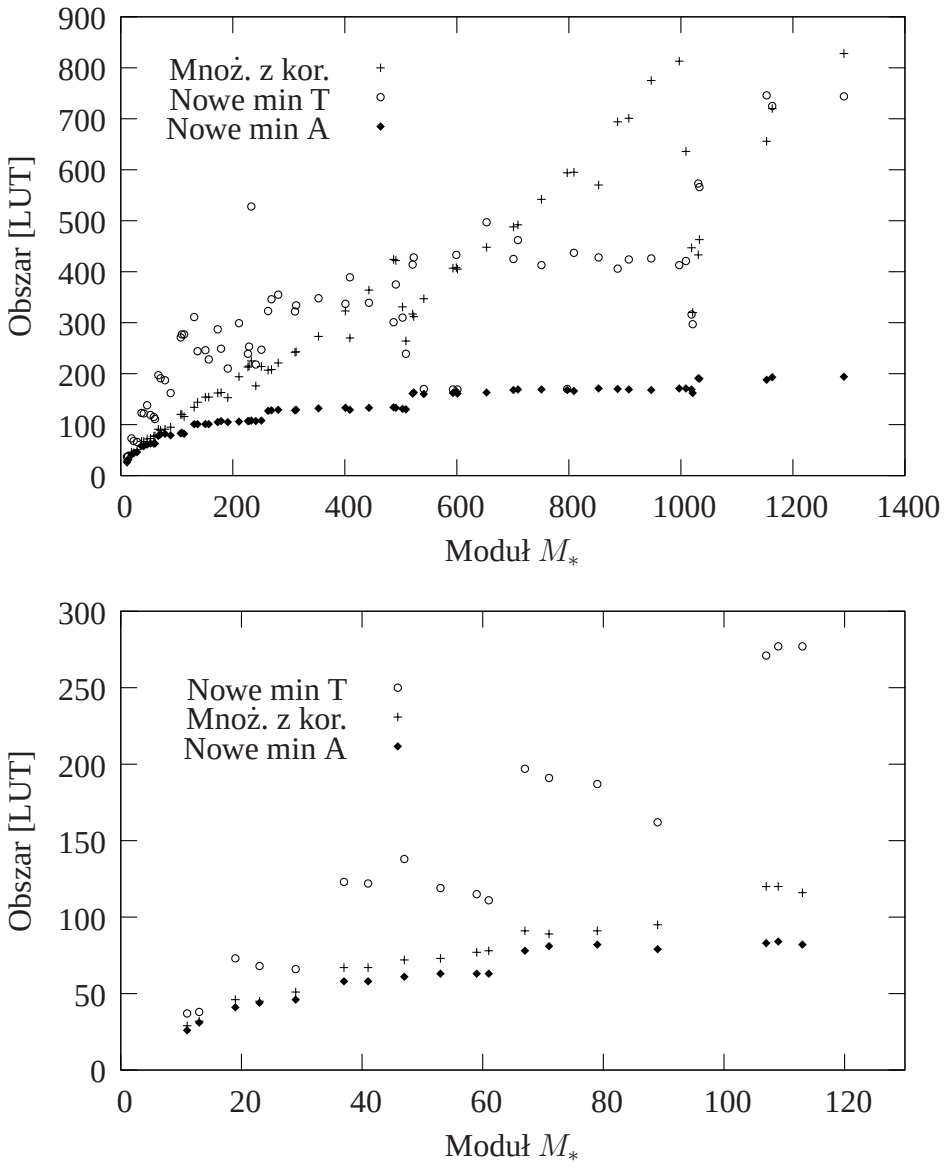
Rysunek 4.16. Porównanie iloczynu AT z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną.



Rysunek 4.17. Porównanie iloczynu AT^2 z układem mnożącym wykorzystującym transformację na izomorficzną grupę addytywną.

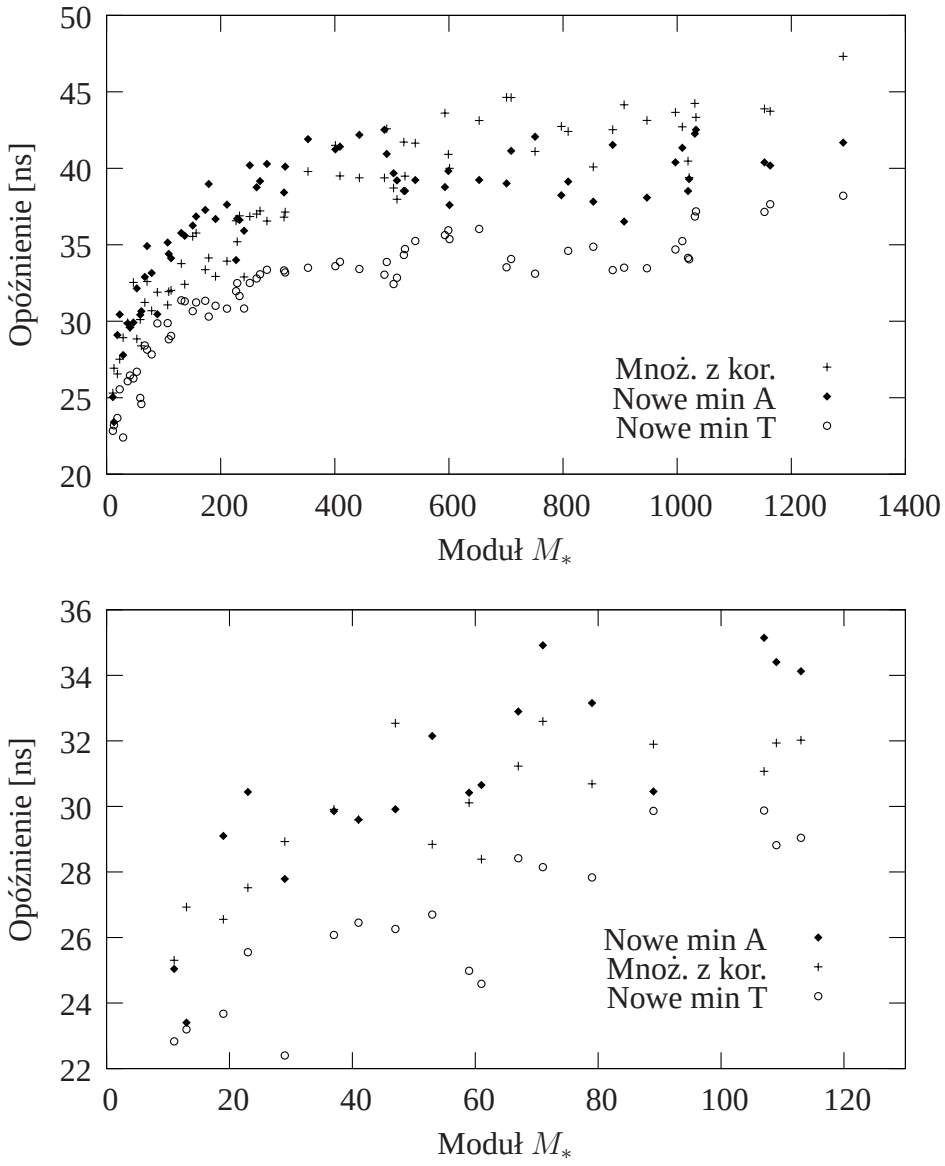
Porównanie z układami mnożącymi z korekcją modulo

Schemat opisanego w pracy [Beu03] układu mnożącego z korekcją modulo znajduje się na rys. 2.11 a) na str. 53. Układy ten zawierają matrycę mnożącą dwa m_*–bitowe operandy, pamięć ROM obliczającą wartość reszty dla wektora m_* bitowego oraz sumator modulo M_*. Długość ścieżki krytycznej układu równa jest sumie opóźnień wprowadzanych przez wymienione bloki. Jeśli pamięć ROM stanowi niepodzielny układ, maksymalna głębokość potoku jest określona przez możliwość potokowania matrycy mnożącej. Układy o tej strukturze są najmniejsze i najwolniejsze wśród dotychczasowych rozwiązań.



Rysunek 4.18. Porównanie zajmowanego obszaru z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo.

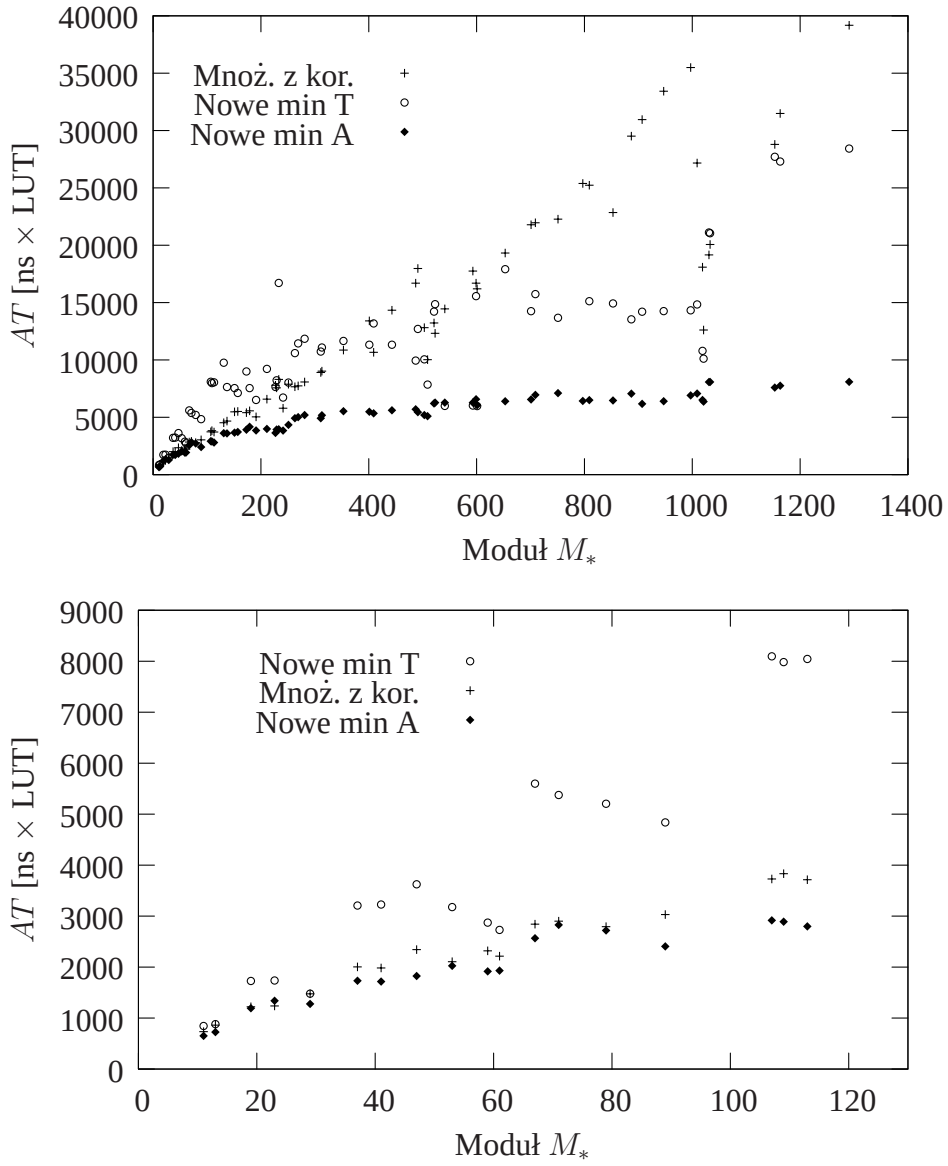
Układ mnożenia z korekcją modulo jest szczególnym przypadkiem układów o strukturze zaproponowanej w rozprawie. Blok generowania iloczynów częściowych zawiera wyłącznie układy mnożenia $2 \cdot k$ i $3 \cdot k$ bity, nie ma możliwości dodania dodatkowych składników (czyli $l = 0$), a generator wyniku składa się z pojedynczej pamięci ROM i subtraktora warunkowego. W algorytmie zaproponowanym w rozprawie (rozdz. 3.2) ograniczenie na maksymalną szerokość pól wydzielonych ze starszej części sumy iloczynów częściowych powoduje pominięcie jednostek o takiej strukturze.



Rysunek 4.19. Porównanie długości ścieżki krytycznej z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo.

Na rys. 4.18 przedstawiono porównanie obszarów układów mnożenia z korekcją i nowych jednostek. Dla wszystkich zaimplementowanych wartości modułu M_* struktura proponowana w rozprawie

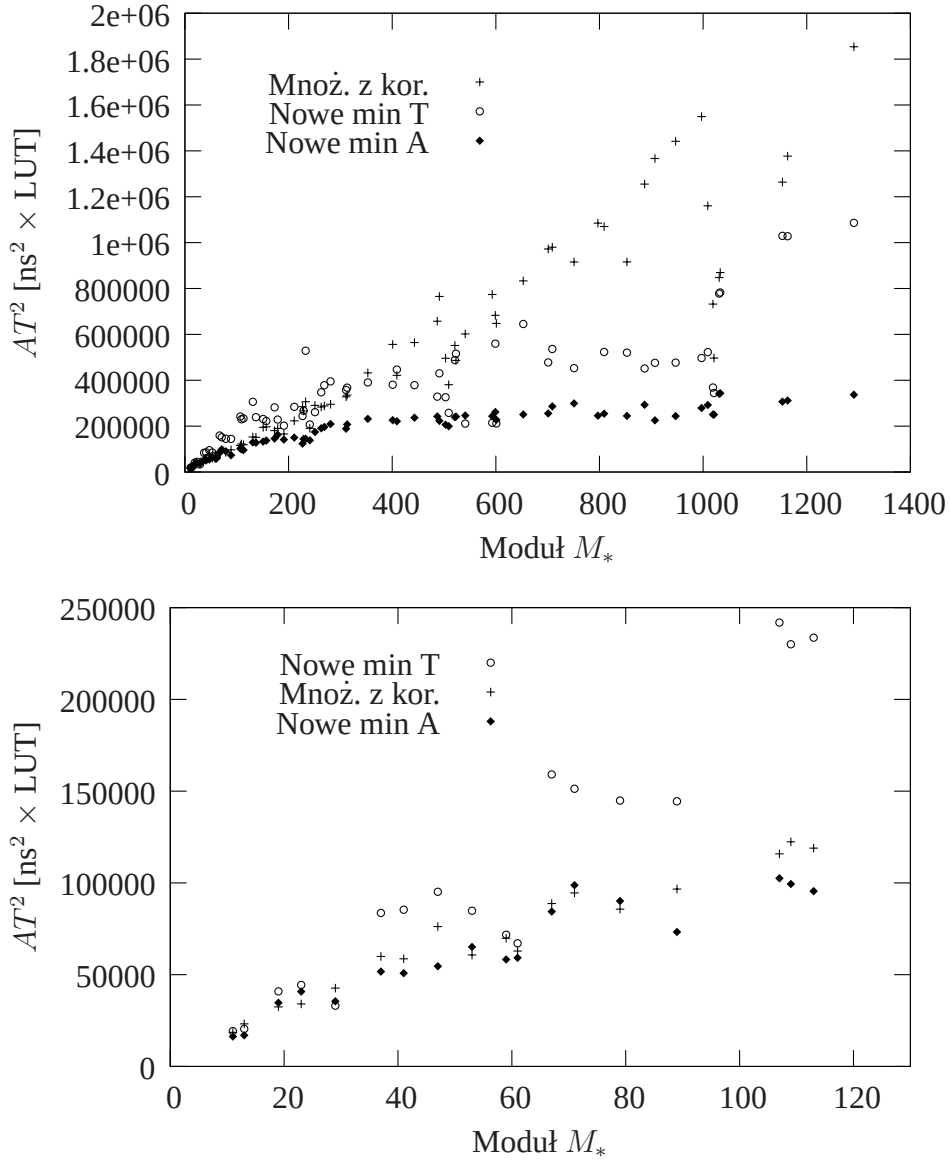
pozwala na skonstruowanie jednostek o najmniejszym obszarze. Wynika to z zastąpienia pojedynczej pamięci ROM w bloku wytwarzania i sumowania iloczynów częściowych rozwiązaniem o mniejszym obszarze. Jednostka mnożąca z korekcją dla modułów $M_* < 512$ zajmuje obszar o wartości pośredniej pomiędzy najmniejszym a najszybszym układem o strukturze z rys. 3.1. W przypadku modułów o wartościach większych od 512 obszar zajmowany przez pamięć ROM staje się dominujący i rozmiar układu mnożącego z korekcją rośnie szybko przekraczając wartości dla układów najszybszych o nowej strukturze.



Rysunek 4.20. Porównanie iloczynu AT z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo.

Na wykresach z rys. 4.19 przedstawiono długości ścieżek krytycznych porównywanych układów

mnożenia. Opóźnienie jednostek mnożących z korekcją modulo jest rozwiązaniem pośrednim pomiędzy najmniejszymi a najszybszymi układami o strukturze z rys. 3.1 dla modułów o wartościach mniejszych od 512. Powyżej tej granicy układy z pracy [Beu03] są wolniejsze od najmniejszych jednostek o strukturze zaproponowanej w rozprawie.



Rysunek 4.21. Porównanie iloczynu AT^2 z układem mnożącym wykorzystującym algorytm mnożenia z korekcją modulo.

Porównanie parametrów AT dla analizowanych jednostek przedstawiono na rys. 4.20. Podobnie jak dla obszaru i opóźnienia, układy mnożenia z korekcją są rozwiązaniami pośrednimi pomiędzy najmniejszymi a najszybszymi jednostkami o nowej strukturze. Dla modułów $M_* > 512$ parametry układów mnożenia z korekcją modulo są gorsze od parametrów jednostek konstruowanych według

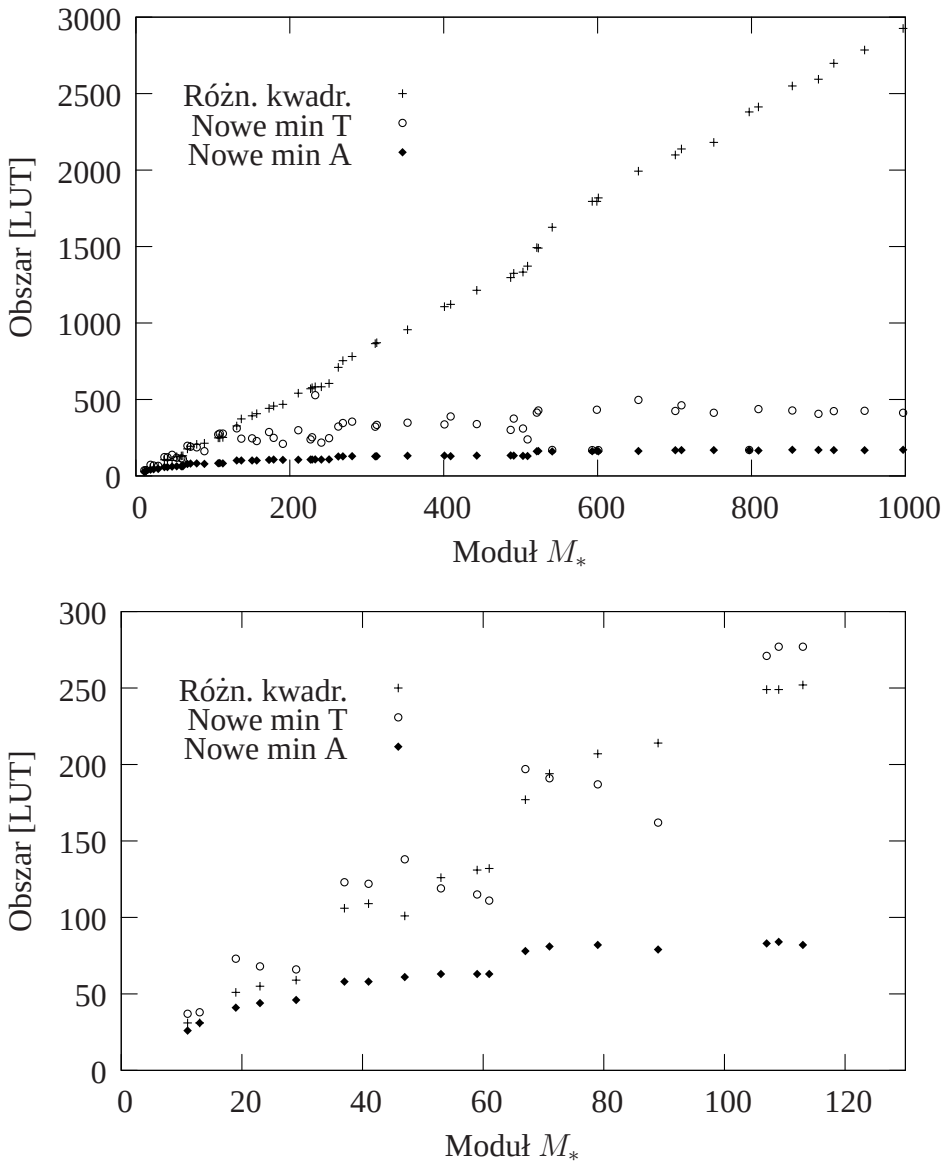
algorytmu zaproponowanego w rozprawie. Podsumowując, układy mnożenia z korekcją stanowią rozwiązanie pośrednie pomiędzy jednostkami najmniejszymi a najszybszymi według nowej koncepcji dla modułów $M_* < 512$. Powyżej tego zakresu M_* struktura opisana w rozprawie pozwala na skonstruowanie układów o lepszych parametrach AT. Wadą układów mnożenia z korekcją może być także mała głębokość potoku ze względu na stosowanie pamięci ROM adresowanych szerokim słowem.

Porównanie z układami mnożenia wykorzystującymi prawo różnicy kwadratów

Schemat zaprezentowanych w pracy [MBGT01] układów mnożenia wykorzystujących prawo różnicy kwadratów znajduje się na rys. 2.11 b) na stronie 53. Składają się one z sumatora i subtraktora m_*–bitowego, dwóch pamięci ROM adresowanych wynikami sumy i różnicy i końcowego subtraktora modulo. W ścieżce krytycznej znajduje się sumator/subtraktor m_*–bitowy, pamięć ROM i końcowy subtraktor modulo. Jeżeli pamięci ROM są układami niepodzielnymi, maksymalna głębokość potoku wynosi 3 lub 4 etapy. Układy o tej strukturze są układami najszybszymi i największymi wśród dotychczasowych rozwiązań jednostek mnożących modulo.

Porównanie zajmowanego obszaru dla tych układów i jednostek o nowej strukturze znajduje się na rys. 4.22. W całym zakresie wartości modułu metoda zaproponowana w rozprawie pozwala skonstruować układ o najmniejszym obszarze. Dla modułów większych od 128, także najszybsze z nowych struktur są mniejsze od układów wykorzystujących różnicę kwadratów. Wynika to z szybkiego wzrostu rozmiaru pamięci ROM adresowanych wynikami sumy i różnicy argumentów. W przypadku modułów mniejszych od 128, obszary układów wykorzystujących różnicę kwadratów i najszybszych według nowej koncepcji są porównywalne. W tym zakresie dla modułów nieznacznie większych od 2^{m_*} najszybsze nowe jednostki są większe, a dla modułów nieznacznie mniejszych od 2^{m_*} większy obszar zajmują układy wykorzystujące różnicę kwadratów.

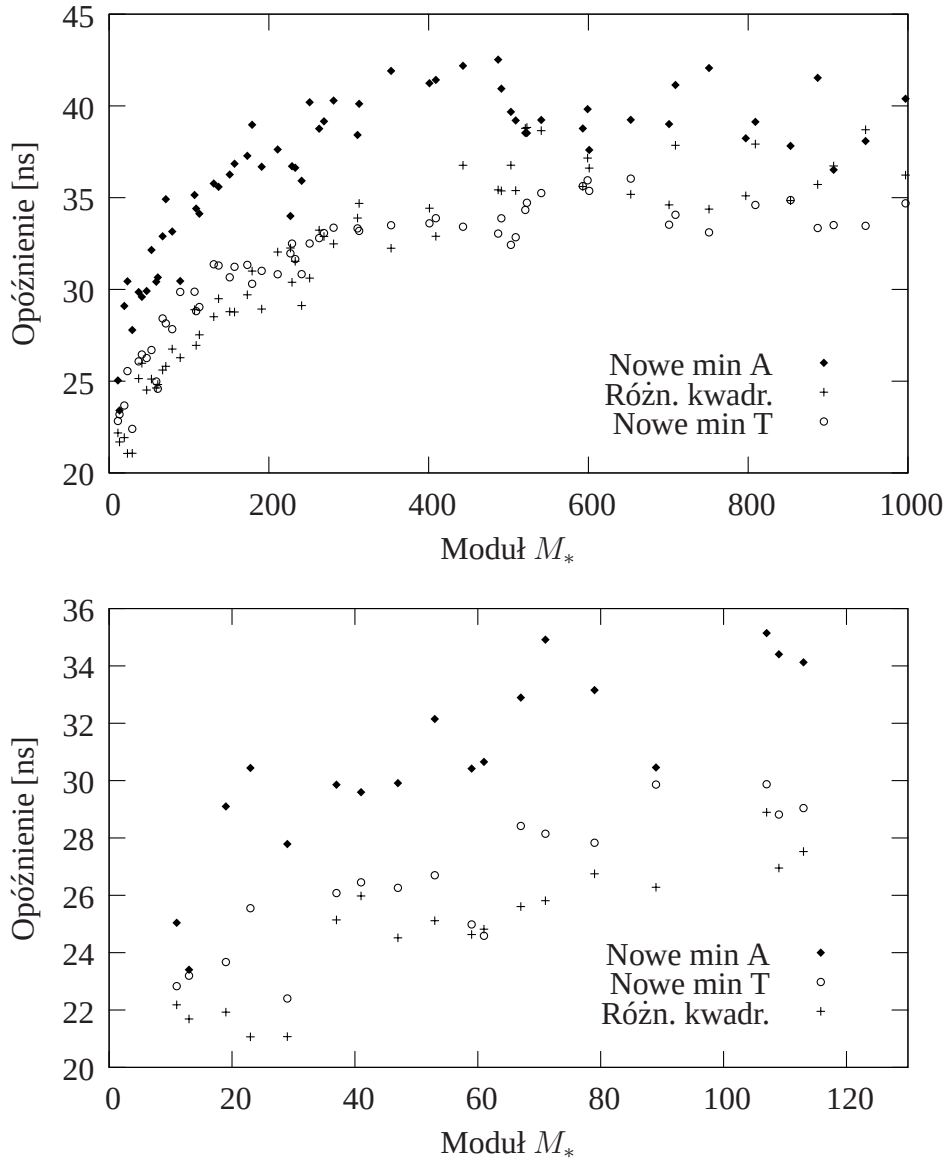
Na rys. 4.23 przedstawiono porównanie długości ścieżki krytycznej dla analizowanych układów. Dla modułów szerszych od 8 bitów długość ścieżki krytycznej układów wykorzystujących różnicę kwadratów zajmuje wartości pośrednie pomiędzy najmniejszymi a najszybszymi układami o nowej strukturze. W tym zakresie metoda zaproponowana w rozprawie pozwala z reguły na znalezienie najszybszego układu. Jeśli $M_* < 512$, opóźnienie wprowadzane przez układy wykorzystujące różnicę kwadratów jest mniejsze. Układy te mogą być szczególnie przydatne, gdy $M_* < 128$, ponieważ pozwalają wtedy zmniejszyć długość ścieżki krytycznej przy obszarze porównywalnym z najszybszymi nowymi strukturami. Powyżej 128, skonstruowanie szybkich jednostek wykorzystujących różnicę



Rysunek 4.22. Porównanie zajmowanego obszaru z układem mnożącym wykorzystującym prawo różnicy kwadratów.

kwadratów wymaga znacznie większego obszaru.

Na rys. 4.24 i 4.25 zamieszczono porównanie iloczynów AT i AT^2 dla badanych układów. Kształt wykresu jest podobny do funkcji opisującej zajmowany obszar, tj. dla modułów $M_* > 128$ następuje szybki wzrost iloczynu AT jednostek wykorzystujących różnicę kwadratów. Podsumowując, układy o nowej strukturze mają zdecydowanie lepsze parametry AT dla modułów $M_* > 512$. W zakresie $M_* \in (128, 512)$ metoda różnicy kwadratów pozwala na zmniejszenie długości ścieżki krytycznej do 10% kosztem znacznego zwiększenia zajmowanego obszaru. Dla modułów $M_* < 128$ długość ścieżki krytycznej układów mnożących wykorzystujących prawo różnicy kwadratów jest najmniejsza

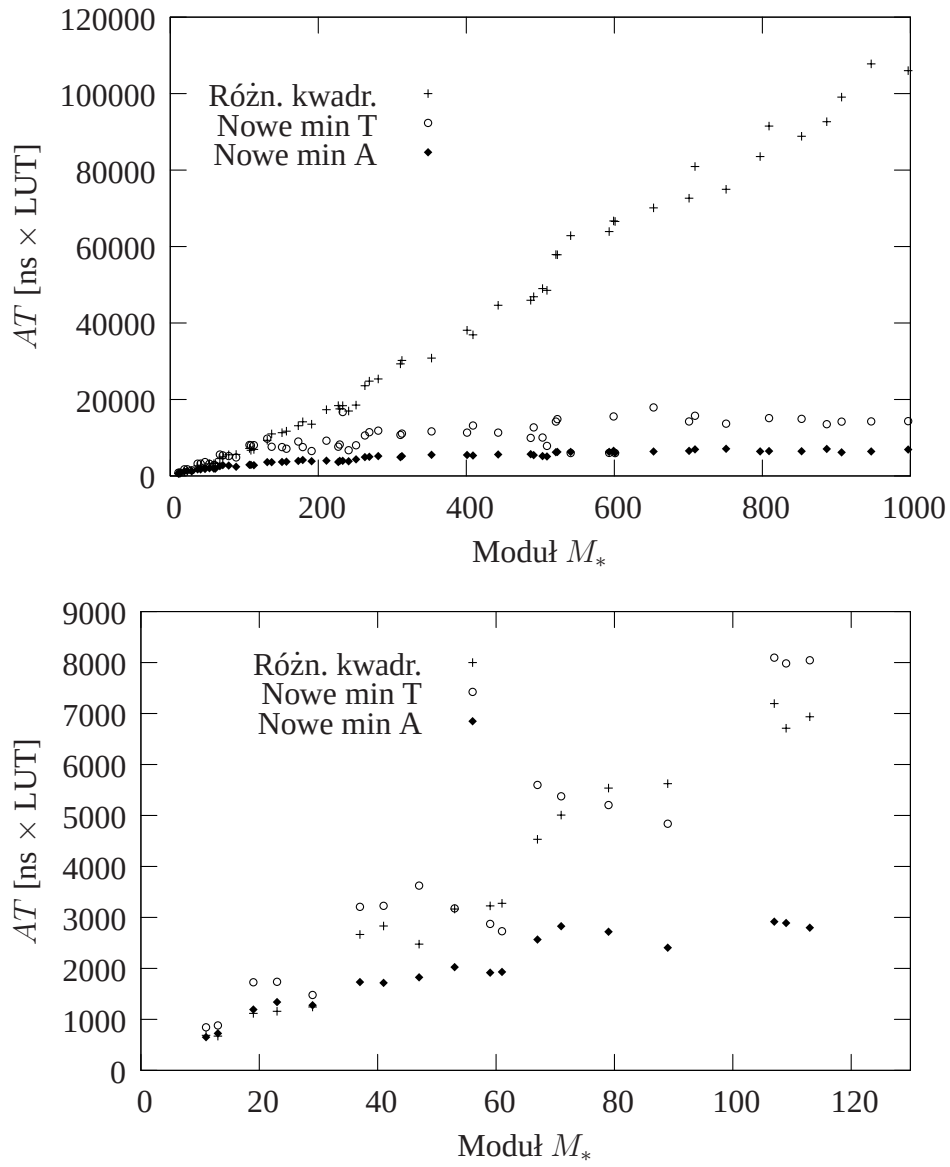


Rysunek 4.23. Porównanie długości ścieżki krytycznej z układem mnożącym wykorzystującym prawo różnicy kwadratów.

przy porównywalnym obszarze. Stosowanie tych układów jest korzystne wszędzie tam, gdzie dla małych modułów wymagane są krótkie ścieżki krytyczne i nie ma potrzeby głębokiego potokowania układu. We wszystkich pozostałych przypadkach lepszym rozwiązaniem są struktury zaproponowane w rozprawie.

Wnioski

Na podstawie powyższego porównania dotychczasowych struktur jednostek mnożących modulo z rozwiązaniem zaproponowanym w rozprawie można stwierdzić, że



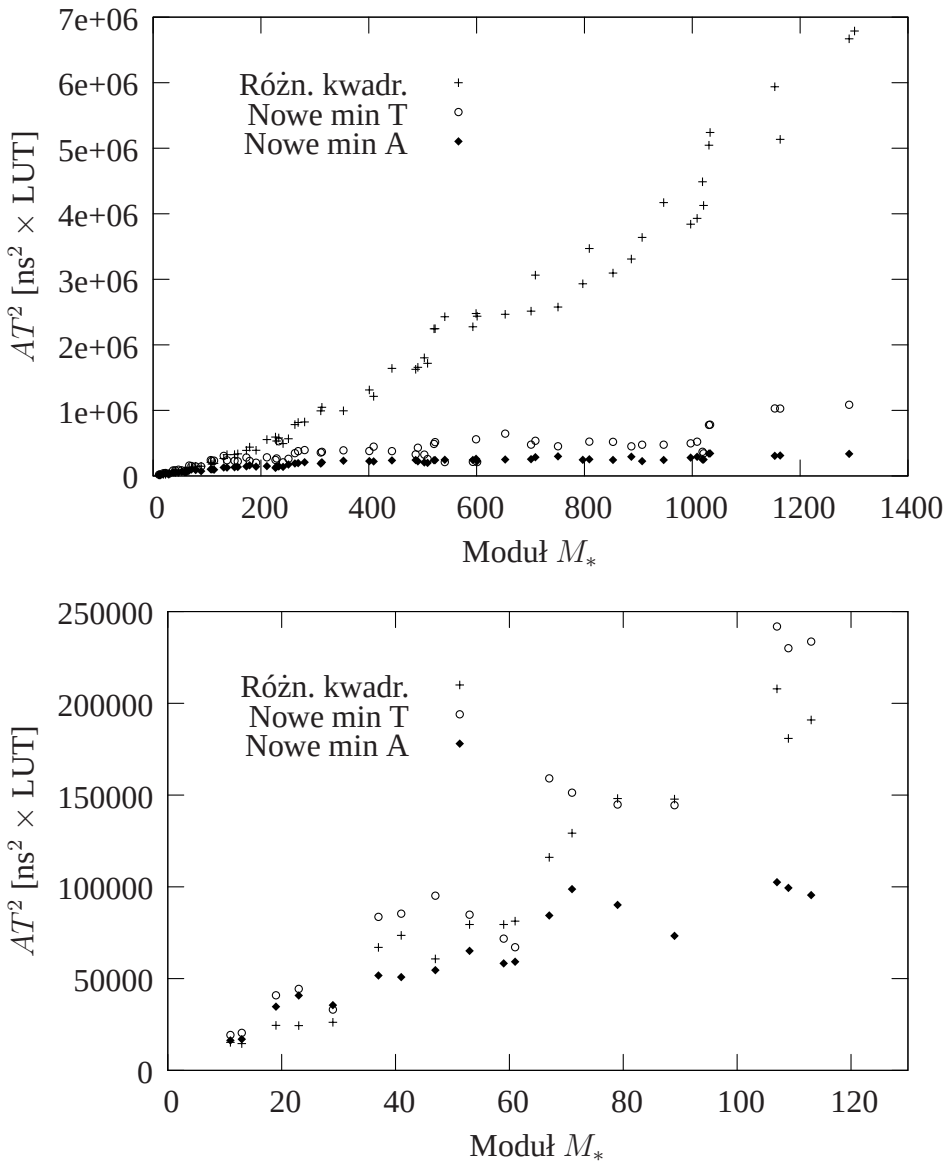
Rysunek 4.24. Porównanie iloczynu AT z układem mnożącym wykorzystującym prawo różnicy kwadratów.

+ w całym zakresie wartości modułu nowe struktury pozwalają na skonstruowanie układów najmniejszych i o najmniejszych wartościach iloczynów AT i AT^2,

+ dla modułów większych od 512 jednostki według nowej koncepcji mogą być także najszybsze,

• jeśli wartość modułu należy do przedziału $[128, 512]$, dotychczasowe rozwiązania są najwyżej 10% szybsze przy znacznie większym obszarze,

– dla modułów mniejszych od 128 dotychczas opracowane struktury są szybsze od najszybszych według nowej metody.



Rysunek 4.25. Porównanie iloczynu AT^2 z układem mnożącym wykorzystującym prawo różnicy kwadratów.

Wynika stąd, że stosowanie nowych układów jest korzystne we wszystkich przypadkach poza tymi, w których dla niewielkich modułów wymagane są bardzo krótkie ścieżki krytyczne. Zasadniczą zaletą struktur zaproponowanych w rozprawie jest możliwość skonstruowania niewielkich i szybkich jednostek dla modułów kilkunastobitowych i większych. Korzystną cechą nowych układów jest też możliwość znalezienia struktury o obszarze bliskim najmniejszemu i opóźnieniu gorszym jedynie o kilka lub kilkanaście procent od rozwiązań najszybszych. Przykładem mogą być jednostki dla modułu 1293, gdzie rozwiązanie najszybsze zajmuje obszar 744 LUT i wprowadza opóźnienie 38.207 ns, a istnieje także układ o obszarze 199 LUT i opóźnieniu 39.615 ns. Wybór odpowiedniej jednostki

do konkretnego zastosowania jest więc zadaniem trudnym i w ogólnym przypadku może wymagać przeanalizowania parametrów wszystkich możliwych konfiguracji.

4.2.3 Wykorzystanie okresowości potęg 2 modulo M_*

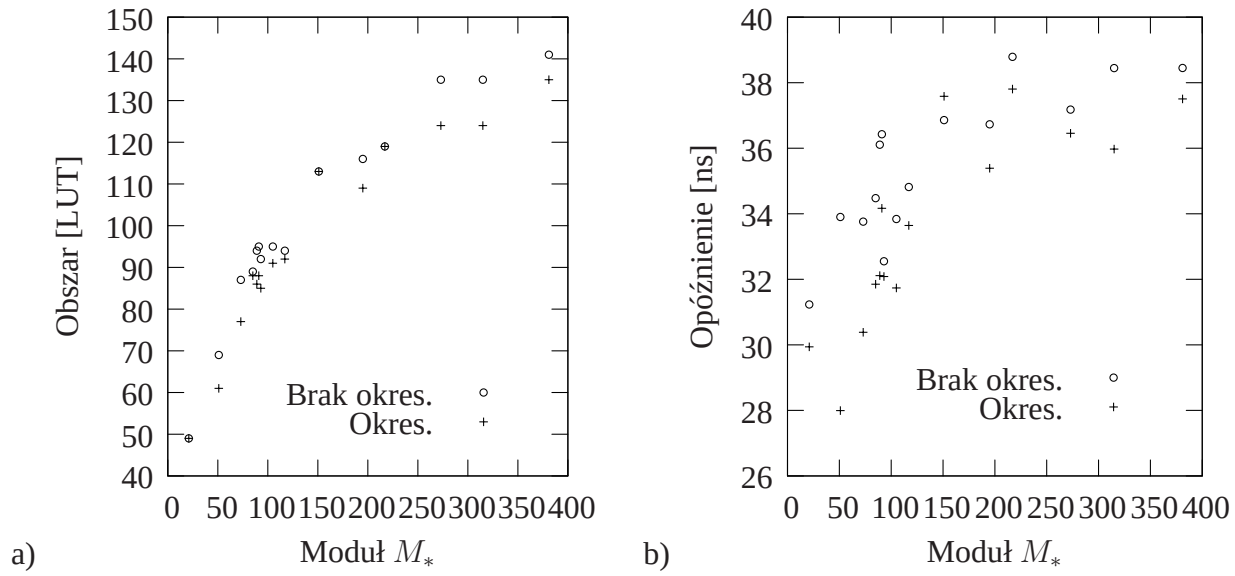
Okresowość potęg 2 modulo M_* może być potencjalnie użyta do redukcji szerokości sumy iloczynów częściowych. Aby zbadać skuteczność tego rozwiązania, zaimplementowano jednostki mnożenia o strukturze przedstawionej na rys. 3.1 w dwóch wersjach: podstawowej oraz z uwzględnieniem okresowości w strukturze sumatora wstępnego. Wykorzystanie okresowości powoduje zmniejszenie szerokości sumy iloczynów częściowych, efektem ubocznym jest więc uproszczenie struktury oraz zmniejszenie liczby możliwych konfiguracji generatora wyniku. Eksperymenty wykonano dla modułów zestawionych w tabeli 4.2. Poszukiwanymi układami były jednostki o minimalnym obszarze i o najkrótszej ścieżce krytycznej. Wyniki eksperymentów przedstawiono na rys. 4.26 i 4.27. Rys. 4.26 dotyczy porównania parametrów układów najmniejszych, a rys. 4.27 układów najszybszych.

Tabela 4.2. Moduły o okresach potęg 2 mod M_* mniejszych od $2 \cdot m_*$.

M_*	21	51	73	85	89	91	93	105	117	151	195	217	273	315	341
m_*	5	6	7	7	7	7	7	7	7	8	8	8	9	9	9
Okres	6	8	9	8	11	12	10	12	12	15	12	15	12	12	10

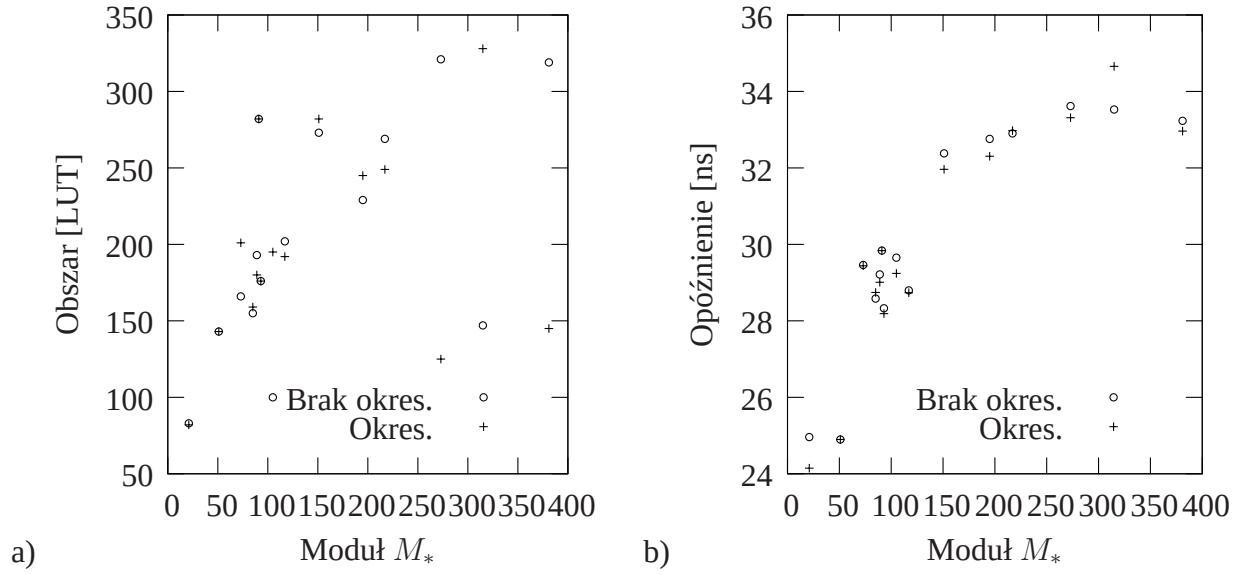
Analizując wykresy z rys. 4.26 można zauważyć, że dla wszystkich zbadanych wartości modułu obszar układu najmniejszego zbudowanego z wykorzystaniem okresowości jest mniejszy lub równy obszarowi układu najmniejszego nie wykorzystującego okresowości. Największy zysk równy 11.5% występuje dla modułu $M_* = 73$. Zmniejszenie zajmowanego obszaru dla układów o minimalnym rozmiarze nie zachodzi jedynie w przypadkach, w których wartość okresu potęg 2 jest porównywalna z maksymalną szerokością sumy iloczynów częściowych (moduły 21, 151 i 217).

W przypadku układów o minimalnym zajmowanym obszarze wykorzystanie okresowości powoduje we wszystkich przypadkach oprócz jednego zmniejszenie długości ścieżki krytycznej. Największy zysk równy 11.8% zaobserwowano dla modułu równego 51. Jedynym przypadkiem, w którym nastąpiło wydłużenie ścieżki krytycznej o niecałe 2% jest $M_* = 151$, czyli jeden z modułów o wartości okresu porównywalnej z szerokością sumy iloczynów częściowych. Można zatem stwierdzić, że przy kryterium najmniejszego obszaru wykorzystanie zjawiska okresowości potęg 2 modulo M_*



Rysunek 4.26. Porównanie a) obszaru oraz b) opóźnienia w funkcji modułu dla układów o minimalnym obszarze.

powoduje równoczesne zmniejszenie zajmowanego obszaru i wprowadzanego opóźnienia. Zaobserwowane zyski w niektórych przypadkach przekraczają 10%.



Rysunek 4.27. Porównanie a) obszaru oraz b) opóźnienia w funkcji modułu dla układów o minimalnym opóźnieniu.

Parametry układów o najkrótszej ścieżce krytycznej przedstawiono na rys. 4.27. W tym przypadku zastosowanie okresowości powoduje w większości przypadków nieznaczne (rzędu 1–2%) zwiększenie obszaru przy równoczesnym minimalnym zmniejszeniu długości ścieżki krytycznej. Istnieją

jednak przypadki, w których wzrost zajmowanego obszaru jest znaczny, np.: dla $M_* = 105$ układ wykorzystujący okresowość jest blisko dwukrotnie większy. Przyczyną jest postać iloczynów częściowych. Dla układów o minimalnym opóźnieniu iloczyny częściowe są obliczane jako reszty modulo M_*. Iloczyny te są zatem reprezentowane przez wektory o szerokości mniejszej od okresu potęg 2 modulo M_*. Wykorzystanie okresowości nie przynosi więc znaczących korzyści dla układów o minimalnej długości ścieżki krytycznej.

Należy jednak zaznaczyć, że wśród układów budowanych z wykorzystaniem okresowości istnieje duża grupa rozwiązań o opóźnieniach porównywalnych z układami najszybszymi, a o wielokrotnie mniejszym obszarze. Przykładem mogą być układy dla modułów 273 i 381, dla których najszybsze struktury wykorzystujące okresowość są ponad dwukrotnie mniejsze przy jednoczesnym nieznacznym skróceniu ścieżki krytycznej. Również w pozostałych przypadkach można wśród jednostek wykorzystujących okresowość znaleźć układy, które przy czasach propagacji większych o ok 1% od układów najszybszych zajmują obszar porównywalny z najmniejszymi rozwiązaniami.

Podsumowując, wykorzystanie okresowości potęg 2 modulo M_* nie powoduje znaczących korzyści dla układów o minimalnej długości ścieżki krytycznej. Zaletą rozwiązań wykorzystujących okresowość jest natomiast możliwość skonstruowania układu o opóźnieniu nieznacznie większym od najszybszego rozwiązania przy zdecydowanie mniejszym obszarze. Dodatkową korzyścią wynikającą z zastosowania okresowości jest niewielki spadek liczby możliwych konfiguracji jednostki.

4.3 Hierarchiczne RNS w strukturach FPGA

Poniżej zaprezentowano rezultaty implementacji resztowych układów arytmetycznych z wykorzystaniem HRNS opisanych w rozdz. 2.3. Implementowaną jednostką jest układ MAC realizujący działanie opisane jako

$$W = X \cdot Y + Z, \tag{4.2}$$

gdzie $X, Y, W, Z \in \mathbb{C}$. Jednostka ta jest stosowana w dalszej części rozprawy jako element układu sprzętowego wspomagania algorytmów oświetlenia globalnego (AOG). Specyfika implementowanego algorytmu narzuca ograniczenia na zakresy dynamiczne poszczególnych operandów. Szerokość wektora bitowego $\mathbf{Y}$ reprezentującego Y jest dwukrotnie większa od szerokości wektora $\mathbf{X}$ reprezentującego X. Szerokość wektora $\mathbf{Z}$ reprezentującego Z jest równa szerokości iloczynu $X \cdot Y$, czyli trzykrotnie większa od szerokości $\mathbf{X}$. Szerokość wyniku $X \cdot Y + Z$ może być o 1 bit większa od sze-

rokości $\mathbf{Z}$, jednak dla uproszczenia układu przyjęto, że szerokość wektora $\mathbf{W}$ jest równa szerokości $\mathbf{Z}$. Oczywiście wymusza to niewielkie ograniczenie zakresu poszczególnych operandów.

Porównanie parametrów jednostek przeprowadzono dla trzech różnych systemów liczbowych: uzupełnieniowego do dwóch U2, RNS o bazie $(2^k - 1, 2^k, 2^k + 1)$ i HRNS o bazie zawierającej czynniki $2^k \pm 1$ i moduł 2^k. Zbudowanie opisywanych HRNS jest możliwe tylko dla niektórych wartości k. Ponieważ w AOG wymagany jest duży zakres dynamiczny, zaimplementowano jednostki dla trzech HRNS o największym zakresie, czyli dla $k \in \{18, 24, 30\}$. Zakres operandów w U2 także został ograniczony do zbioru $\{18, 24, 30\}$ bitów dla wektora $\boldsymbol{X}$, co daje szerokość wyniku równą odpowiednio 54, 72 i 90 bitów. Pozwala to na wiarygodne porównanie parametrów jednostek operujących na liczbach o podobnym zakresie dynamicznym.

Zakres dynamiczny implementowanych RNS i HRNS jest określony jako $2^{3k} - 2^k$. Jest on nieznacznie mniejszy niż zakres oferowany przez system U2 dla wektora $3k$–bitowego. Pomimo tego w poniższej analizie jednostki pracujące w systemie U2 i systemach resztowych są traktowane jako układy o tym samym zakresie dynamicznym. W większości przypadków jest to działanie uzasadnione, gdyż różnica w wartości zakresów dynamicznych jest pomijalna, szczególnie dla dużych k.

4.3.1 Opis struktury jednostek

Porównywane jednostki zostały zaimplementowane z użyciem zestawu narzędzi opisanych na początku rozdziału. Efektywna implementacja struktur układów arytmetycznych wymaga wykorzystania zarówno rozwiązań wydajnie implementowanych w FPGA, jak i możliwości optymalizacji układu oferowanych przez kompilator. Wykorzystanie kompilatora pozwala też zwolnić projektanta/konstruktora z konieczności implementowania układu z prymitywów dostępnych w FPGA, co znacznie przyspiesza i ułatwia budowanie systemów wykorzystujących RNS.

U2

Opis struktur układów mnożenia akumulacyjnego dla systemu U2 przedstawiono na list. 4.2. Pakiety importowane na początku kodu zawierają m.in. przeciążone operatory arytmetyczne. Pozwalają one kompilatorowi na optymalizację układu i użycie dedykowanej logiki zwiększającej wydajność operacji arytmetycznych. Niestety, dokładne dane na temat samego procesu syntezy nie są dostępne.

Oprócz opisu z list. 4.2 podjęto także próby opisania układu na niższym poziomie w celu polep-

Listing 4.2. Opis jednostki mnożenia akumulacyjnego w języku VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity MAC_U2 is
    Port (
      X : in std_logic_vector(17 downto 0);
      Y : in std_logic_vector(35 downto 0);
      Z : in std_logic_vector(53 downto 0);
      W : out std_logic_vector(53 downto 0)
     );
end MAC_U2;

architecture beh of MAC_U2 is
begin
        W <= X * Y + Z ;
end Behavioral;
```

szenia parametrów AT. Modyfikacje dotyczyły zmian szerokości wytwarzanych iloczynów częściowych oraz struktury sumatora iloczynów częściowych. Uzyskane wyniki były podobne do wygenerowanych automatycznie. W nielicznych przypadkach układ opisany na niskim poziomie wprowadzał opóźnienia mniejsze o kilka procent kosztem kilku lub kilkunastoprocentowego wzrostu zajmowanego obszaru.

RNS

Implementacja jednostek w RNS$(2^k - 1, 2^k, 2^k + 1)$ wymaga efektywnego opisu struktury układu dla modułów $2^k \pm 1$ dla k rzędu kilkunastu–kilkudziesięciu bitów. Zaimplementowane jednostki są wzorowane na układach mnożenia modulo $2^k \pm 1$ zaprezentowanych w pracach [Beu02] i [Zim99].

Opis jednostek zamieszczono we wstępie teoretycznym (rozdz. 2.2.3, str. 50).

Algorytm mnożenia modulo $2^k - 1$ według pracy [Zim99] wymaga poszerzenia o możliwość dodania składnika Z. Maksymalna wartość wyrażenia $X \cdot Y + Z$ dla $X, Y, Z \in [0, 2^k - 1)$ wynosi $2^{2k} - 3 \cdot 2^k - 2$. Operacja dodawania może być więc wykonana bez obawy o zwiększenie zakresu wyniku poza wartość możliwą do skorygowania przez końcowy sumator modulo $2^k - 1$ dodający dwa k–bitowe pola wyniku. Opis układu w języku VHDL przedstawiono na list. 4.3. Zapis ten umożliwia kompilatorowi VHDL optymalne wykorzystanie układów mnożących $2 \cdot k$ bity i sumatorów CPA. Struktura połączeń po implementacji nie jest dokładnym odwzorowaniem schematu wynikającego bezpośrednio z list. 4.3. Analiza struktury połączeń bloków wewnątrz układu FPGA po implementacji wykazała, że kompilator dokonuje częściowej redukcji modulo na etapie sumowania iloczynów częściowych.

Listing 4.3. Opis jednostki MAC modulo $2^k - 1$

```
tmp1    <= X * Y + Z ;
tmp2    <= ('0' & tmp1(k-1 downto 0)) +
           ('0' & tmp1(2*k-1 downto k))  ;
W       <= tmp2(k-1 downto 0) + tmp2(k)) ;
```

Jednostka MAC dla modułu $2^k + 1$ jest implementowana na podstawie układu mnożącego zaprezentowanego w pracy [Beu02]. Układ ten zawiera matrycę mnożącą, multiplekser i sumator modulo $2^k + 1$. Można w nim pominąć multiplekser, co pozwala zmniejszyć zajmowany obszar i wprowadzane opóźnienia za cenę wprowadzenia podwójnej reprezentacji zera. Opis układu w języku VHDL przedstawiono na list. 4.4. Również w tym przypadku możliwe było zintegrowanie dodatkowego sumatora z matrycą mnożącą, co pozwala przenieść ciężar optymalizacji układu na kompilator VHDL.

Listing 4.4. Opis jednostki MAC modulo $2^k + 1$

```
tmp1    <= X * Y + Z ;
tmp2    <= ("0" & tmp1(k-1 downto 0)) +
           ("0" & (not tmp1(2*k-1 downto k))) + "1"  ;
W       <= ("0" & tmp2(k-1 downto 0)) +
           ("00" & (not tmp2(k)) ) + tmp1(2*k) ;
```

HRNS

Tor obliczeniowy w opisywanym HRNS składa się z zestawu jednostek dla wszystkich czynników liczb $2^k \pm 1$ i jednostki modulo 2^k. Jednostki dla czynników są generowane przy pomocy algorytmu zaprezentowanego w rozdz. 3.2 (alg. 10, str. 129).

Algorytm 10 generuje dla zadanego modułu zbiór jednostek o różnych parametrach AT. Wybór odpowiedniej wersji jednostek dla prezentowanego HRNS można przeprowadzić na wiele sposobów. Dane zaprezentowane poniżej dotyczą dwóch przypadków. W obu wybór jednostek przeprowadzany jest w dwóch krokach. W pierwszym kroku wybierana jest jednostka dla czynnika, dla którego długość ścieżki krytycznej będzie największa spośród wybranych układów. Wybrany układ wyznaczał granicę szybkości jednostki MAC dla zadanego HRNS.

Po określeniu maksymalnego opóźnienia całego układu wybierane są struktury jednostek MAC dla pozostałych czynników. Wybierane były jednostki najmniejsze spośród układów o ścieżce krytycznej krótszej od układu o największym wprowadzanym opóźnieniu. Reguła ta pozwala na skonstruowanie jednostki o odpowiedniej długości ścieżki krytycznej przy zachowaniu niewielkiego obszaru.

Różnica pomiędzy zastosowanymi metodami doboru jednostek dotyczy sposobu wyboru układu o najdłuższej ścieżce krytycznej. W pierwszym przypadku dla każdego czynnika wybierany jest układ najszybszy, po czym spośród wyników typowany jest układ najwolniejszy. Dzięki temu możliwe jest zbudowanie układu o minimalnym opóźnieniu.

W większości przypadków ograniczeniem szybkości całego układu jest długość ścieżki krytycznej dla jednostki modulo największy czynnik liczb $2^k \pm 1$. Jednostka ta zajmuje duży obszar, a jej znalezienie wymaga sprawdzenia dużej liczby konfiguracji. Z tego powodu zastosowano także drugie kryterium wyboru jednostki ograniczającej szybkość całego układu. Dla każdego czynnika wybierano układ najszybszy, ale tylko spośród jednostek, w których iloczyny częściowe obliczane są za pomocą układów mnożących $2 \cdot k$ lub $3 \cdot k$ bitów. Spośród wybranych układów typowano następnie rozwiązanie o najdłuższej ścieżce krytycznej. Wybrana jednostka zajmowała znacznie mniejszy obszar od układu najszybszego dla danego czynnika. W związku z tym obszar całego układu uległ zmniejszeniu kosztem niewielkiego wzrostu długości ścieżki krytycznej.

4.3.2 Wyniki implementacji

Parametry zaimplementowanych jednostek MAC dla opisywanych HRNS przedstawiono w tabelach 4.3 i 4.4. Tabela 4.3 zawiera parametry jednostek, w których ograniczenie szybkości całego układu wynika z opóźnienia wprowadzanego przez jednostkę najwolniejszą spośród najszybszych dla poszczególnych czynników. W tab. 4.4 zawarto parametry układów, w których jednostką najwolniejszą jest jednostka wybrana spośród struktur, w których iloczyny częściowe są obliczane za pomocą układów mnożących $2 \cdot k$ lub $3 \cdot k$ bitów.

Na kompletny tor resztowy składa się zbiór jednostek MAC dla poszczególnych czynników modułów $2^k \pm 1$ oraz dla modułu 2^k. Wiersze, w których pole moduł zawiera wartość $2^k \pm 1$, zawierają sumę obszaru oraz najdłuższą ścieżkę krytyczną jednostek dla czynników odpowiedniego modułu $2^k \pm 1$. Wiersze z pogrubionymi danymi zawierają sumę obszarów oraz najdłuższą ścieżkę krytyczną jednostek dla wszystkich modułów wchodzących w skład danego HRNS.

Jednostki dla czynników liczb $2^k \pm 1$ zostały wygenerowane za pomocą algorytmu zaproponowanego w rozprawie (alg. 10, str. 129). Wyjątkiem są układy dla modułów 5,7,9,13, w których generator reszty modulo M_* dla sumy iloczynów częściowych składa się wyłącznie z pojedynczej pamięci ROM. Dzięki temu układy te charakteryzują się nieznacznie lepszymi parametrami AT od struktur wygenerowanych przez alg. 10. Różnice w zajmowanym obszarze nie przekraczają kilku tablic LUT, a długość ścieżki krytycznej dla tych modułów jest w obu przypadkach znacznie mniejsza od najwolniejszych układów w danym HRNS. W ogólnym przypadku ręczna optymalizacja struktury jednostek dla małych modułów nie jest więc konieczna.

Różnice pomiędzy tab. 4.3 i 4.4 dotyczą wyłącznie przypadków dla $k \in \{24, 30\}$. Dla $k = 18$ układem najwolniejszym jest jednostka modulo $M_* = 109$, dla której w najszybszej strukturze iloczyny częściowe są obliczane za pomocą układów mnożących. Dane w tab. 4.3 i 4.4 są niemal identyczne, poza układami dla największych czynników. Pomimo tego zmiana sposobu typowania jednostki najwolniejszej ma istotny wpływ na obszar zajmowany przez kompletny tor obliczeniowy.

Ograniczenie jednostek MAC do struktur należących do grupy pierwszej powoduje zmniejszenie zajmowanego obszaru o 20.1% dla $k = 24$ i 22.4% dla $k = 30$. Wzrost długości ścieżki krytycznej wynosi 4.5% w obu przypadkach, tak więc układy zbudowane z jednostek należących do grupy pierwszej charakteryzują się lepszym iloczynem AT. Z powodu niewielkiego wzrostu długości ścieżki krytycznej układy te mogą być przydatne nawet w sytuacjach, w których szybkość jest czynnikiem

Tabela 4.3. Parametry jednostek MAC dla HRNS – minimalne opóźnienie

$k = 18$			$k = 24$			$k = 30$		
Moduł	A [LUT]	T [ns]	Moduł	A [LUT]	T [ns]	Moduł	A [LUT]	T [ns]
7	24	28.106	5	17	18.402	7	24	28.106
19	50	29.181	7	24	28.106	9	28	25.805
27	54	30.134	9	28	25.805	11	35	26.509
73	77	30.384	13	33	24.893	31	47	28.266
$2^{18}-1$	205	30.384	17	50	28.539	151	119	35.454
5	17	18.402	241	124	35.853	331	157	34.968
13	33	24.893	$2^{24}-1$	276	35.853	$2^{30}-1$	410	35.454
37	69	28.842	97	94	34.045	13	33	24.893
109	157	28.317	257	138	35.191	25	52	30.330
$2^{18}+1$	276	28.842	673	434	37.346	41	72	28.666
2^{18}	183	26.789	$2^{24}+1$	666	37.346	61	73	35.324
$\sum$	**664**	**30.384**	2^{24}	316	29.817	1321	587	36.087
			$\sum$	**1258**	**37.346**	$2^{30}+1$	837	36.087
						2^{30}	484	31.964
						$\sum$	**1711**	**36.087**

decydującym.

Analizując dane z tabel 4.3 i 4.4 należy zwrócić uwagę na szczególnie korzystne parametry jednostek MAC dla modułu 2^k. Pomimo jego dużej szerokości, a więc i dużym wpływie na zakres dynamiczny systemu, wprowadzane opóźnienie jest zdecydowanie mniejsze od układów dla największych czynników. Obszar zajmowany przez MAC modulo 2^k jest podobny do sumy obszarów układów dla czynników modułów $2^k \pm 1$. Biorąc pod uwagę łatwość konwersji wejściowej, stosowanie modułu tej postaci jest szczególnie korzystne. Jeśli moduł 2^k miałby być składnikiem dowolnego RNS, wprowadzane opóźnienie jest akceptowalne nawet dla k trzykrotnie większego od szerokości największego z pozostałych modułów. Dotyczy to wszystkich jednostek o strukturze zaproponowanej w rozprawie implementowanych w układach rodziny Spartan 2 i dla przebadanego zakresu modułów.

W tabeli 4.5 przedstawiono parametry jednostek MAC dla modułów $2^k \pm 1$. Podane wyniki po-

Tabela 4.4. Parametry jednostek MAC dla HRNS – zmniejszony obszar

$k = 18$			$k = 24$			$k = 30$		
Moduł	A [LUT]	T [ns]	Moduł	A [LUT]	T [ns]	Moduł	A [LUT]	T [ns]
7	24	28.106	5	17	18.402	7	24	28.106
19	50	29.181	7	24	28.106	9	28	25.805
27	54	30.134	9	28	25.805	11	35	26.509
73	77	30.384	13	33	24.893	31	47	28.266
$2^{18}-1$	205	30.384	17	50	28.539	151	113	37.584
5	17	18.402	241	119	36.935	331	141	37.658
13	33	24.893	$2^{24}-1$	271	36.935	$2^{30}-1$	388	37.658
37	69	28.842	97	94	34.045	13	33	24.893
109	100	30.533	257	135	36.918	25	52	30.330
$2^{18}+1$	219	30.533	673	189	39.023	41	72	28.666
2^{18}	183	26.789	$2^{24}+1$	418	39.023	61	73	35.324
$\sum$	**607**	**30.533**	2^{24}	316	29.817	1321	225	37.714
			$\sum$	**1005**	**39.023**	$2^{30}+1$	455	37.714
						2^{30}	484	31.964
						$\sum$	**1327**	**37.714**

zwalają na porównanie układów implementowanych z użyciem HRNS z opracowanymi dotychczas układami arytmetycznymi modulo $2^k \pm 1$. Porównanie dotyczy jednostek MAC skonstruowanych z użyciem algorytmu mnożenia modulo $2^k + 1$ opisanego w pracy [Beu02] oraz algorytmu mnożenia modulo $2^k - 1$ według koncepcji opisanej w [Zim99]. Dane w tab. 4.5 dotyczą układu modulo $2^k + 1$, w którym pominięto multiplekser. Powoduje to zmniejszenie zajmowanego obszaru i wprowadzanego opóźnienia kosztem wprowadzenia podwójnej reprezentacji zera. Jednostki implementowane z użyciem HRNS składają się ze zbioru układów MAC przedstawionych w tab. 4.3 i 4.4. W ostatnim wierszu tab. 4.5 zawarto parametry jednostki MAC wykonującej operację $X \cdot Y + Z$ dla $X, Y, Z \in [0, 2^k)$ w kodzie naturalnym binarnym NB.

Analizując parametry jednostek z tab. 4.5 można zauważyć, że podstawową zaletą stosowania HRNS jest znaczne zmniejszenia zajmowanego obszaru przy jednoczesnym spadku długości ścieżki

Tabela 4.5. Parametry jednostek MAC realizowanych różnymi metodami dla modułów $2^k \pm 1$

Moduł	Metoda	$k = 18$		$k = 24$		$k = 30$	
		A [LUT]	T [ns]	A [LUT]	T [ns]	A [LUT]	T [ns]
$2^k - 1$	pełne	417	36.853	704	38.107	1065	43.043
	HRNS z tab. 4.3	205	30.384	276	35.853	410	35.454
	HRNS z tab. 4.4	205	30.384	271	36.935	388	37.658
$2^k + 1$	pełne	454	38.647	753	39.382	1126	42.345
	HRNS z tab. 4.3	276	28.842	666	37.346	837	36.087
	HRNS z tab. 4.4	219	30.533	418	39.023	455	37.714
MAC NB		381	31.112	656	32.169	1005	35.199

krytycznej o kilka procent. Użycie HRNS pozwala na ponad dwukrotne ograniczenie zajmowanego obszaru w stosunku do pełnego układu modulo $2^k \pm 1$. Szczególnie duże zyski, zarówno w zajmowanym obszarze jak i wprowadzanym opóźnieniu, zaobserwowano dla modułów postaci $2^k - 1$ i HRNS optymalizowanego pod kątem niewielkich opóźnień. Wynika to z możliwości rozłożenia tych modułów na niewielkie czynniki.

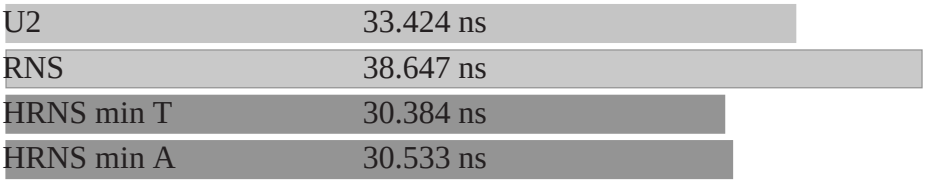
Dla zbadanych modułów postaci $2^k + 1$ jeden z czynników jest znacznie większy od pozostałych, w związku z czym HRNS o najkrótszej ścieżce krytycznej wymaga użycia jednostki o dużym obszarze. Stosowanie HRNS w tym przypadku nadal pozwala polepszyć parametry AT układu, jednak zyski nie są już tak duże.

Dla modułów wchodzących w skład opisywanych HRNS zarówno zajmowany obszar, jak i długość ścieżki krytycznej układów o strukturze proponowanej w rozprawie może być przybliżona zależnością liniową. Zyski wynikające ze stosowania HRNS są więc proporcjonalne do stosunku k do szerokości największego czynnika. Dla przebadanego zakresu modułów korzyści wynikające z zastosowania HRNS były najmniej widoczne dla $k = 24$, gdzie stosunek k do szerokości największego czynnika jest najmniejszy.

Na rys. 4.28 zaprezentowano porównanie parametrów jednostek MAC realizujących operację określoną równaniem (4.2) dla różnych zakresów dynamicznych i struktury układu. Porównanie obejmuje układy zrealizowane w systemie U2, RNS z bazą $(2^k - 1, 2^k, 2^k + 1)$ oraz z użyciem HRNS z tabel 4.3 i 4.4. Skala dla poszczególnych zakresów jest tak dobrana, aby umożliwić porównanie

a) $k = 18$

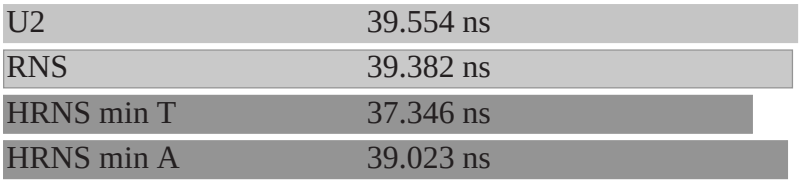
Obszar



Opóźnienie

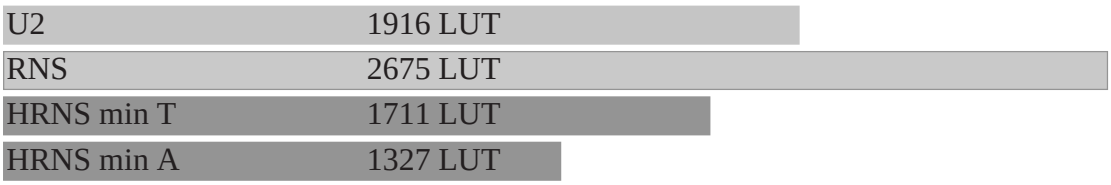
b) $k = 24$

Obszar

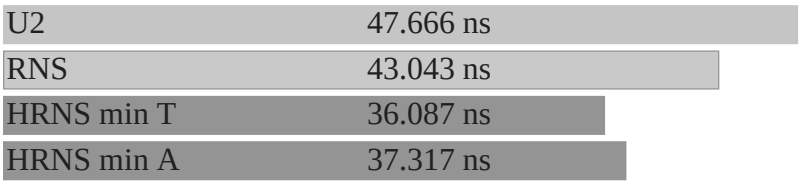


Opóźnienie

c) $k = 30$



Obszar



Opóźnienie

Rysunek 4.28. Porównanie charakterystyk AT różnych implementacji jednostki mnożenia akumulacyjnego dla zakresu dynamicznego a) 54, b) 72 i c) 90 bitów.

zmian parametrów jednostek w stosunku do implementacji w U2 traktowanej jako punkt odniesienia.

Porównując parametry jednostek w systemie uzupełnieniowym i RNS można stwierdzić, że stosowanie RNS $(2^k - 1, 2^k, 2^k + 1)$ jest uzasadnione jedynie w przypadku konieczności zmniejszenia długości ścieżki krytycznej dla dużych zakresów. W omawianym przypadku do wartości $k = 24$ opóźnienie wprowadzane przez jednostki w RNS jest większe lub równe opóźnieniu układów w U2. Dla $k = 30$ zaobserwowano spadek długości ścieżki krytycznej o 9.7%. We wszystkich przypadkach układy wykorzystujące RNS charakteryzują się obszarem większym o ok. 40–50%, przy czym w miarę wzrostu k narzut spowodowany RNS ulega zmniejszeniu. Można więc stwierdzić, że stosowanie RNS $(2^k - 1, 2^k, 2^k + 1)$ w układach FPGA nie przynosi istotnych korzyści, a wręcz może spowodować pogorszenie parametrów układu.

Znacznie korzystniejsza sytuacja występuje dla układów konstruowanych z użyciem opisywanych HRNS. Jednostki wykorzystujące HRNS są zarówno mniejsze, jak i szybsze od jednostek w U2. Zyski zaobserwowano dla obu zaprezentowanych HRNS. Szczególnie interesujące mogą być układy z tab. 4.4, które w stosunku do jednostek wykorzystujących klasyczną arytmetykę uzupełnieniową oferują jednoczesne zmniejszenie obszaru o 15–30% i wprowadzanych opóźnień do 22% dla $k = 30$.

Należy podkreślić, że korzyści wynikające ze stosowania HRNS rosną w miarę zwiększania zakresu dynamicznego i stosunku k do szerokości największego czynnika. Zależność tę łatwo uzasadnić. Obszar układu mnożącego wykorzystującego klasyczną arytmetykę uzupełnieniową rośnie z kwadratem liczby bitów. Pojedyncza jednostka dla czynnika wchodzącego w skład HRNS zajmuje obszar zależny liniowo od liczby bitów koniecznej do zapisania tego czynnika. Zarówno liczba jednostek, jak i rozmiar pojedynczej jednostki są ograniczone stosunkiem szerokości największego czynnika do zakresu dynamicznego systemu. Suma obszarów zajętych przez wszystkie układy w ramach HRNS może zatem być oszacowana na podstawie stosunku szerokości największego czynnika do zakresu systemu. Jeśli dla jednostek o analizowanej strukturze stosunek ten jest mniejszy od 8, użycie HRNS nie powoduje polepszenia charakterystyk AT (przykład dla $k = 24$).

Zyski w długości ścieżki krytycznej także są uzależnione od wspomnianego ilorazu. Wprowadzane opóźnienie dla jednostek w arytmetyce uzupełnieniowej i układów modulo zależy od logarytmu szerokości operandu. Ponieważ najdłuższa ścieżka krytyczna występuje dla układu o największej szerokości argumentów, przyrost szybkości zależy od stosunku szerokości największego czynnika do zakresu systemu.

4.4 Procesor wspomagający obliczenia w AOG

W poprzednim rozdziale przedstawiono techniki pozwalające na zwiększenie wydajności jednostek arytmetycznych wykonujących operacje mnożenia i dodawania. Użycie hierarchicznych resztowych systemów liczbowych umożliwia zmniejszenie długości ścieżki krytycznej o ok. 20 % w stosunku do jednostek wykorzystujących klasyczną arytmetykę uzupełnieniową. Obszar układu w obu przypadkach jest porównywalny. Użycie HRNS wymaga jednak stosowania konwerterów oraz unikania operacji „trudnych". Celem tego rozdziału jest prezentacja sprzętowej implementacji z użyciem arytmetyki resztowej podstawowych problemów obliczeniowych występujących w AOG. Dodatkowo podano przykład nowej, skalowalnej architektury procesora wspomagającego obliczenia w AOG, w którym zastosowano wspomnianą jednostkę testowania przecięć. Architektura ta jest rozwinięciem pomysłu zaprezentowanego w pracy [Tom05c]. Prezentowana struktura może być zaimplementowana z wykorzystaniem arytmetyki resztowej, jak i wyłącznie arytmetyki uzupełnieniowej. Na przykładzie proponowanej struktury zostanie wykazane, że stosowanie arytmetyki resztowej jest uzasadnione i korzystne w układach sprzętowego wspomagania algorytmów oświetlenia globalnego.

Podstawowym celem proponowanej architektury jest analiza możliwości zastosowania arytmetyki resztowej w układach sprzętowego wspomagania AOG. Układ ten pozwala na efektywną implementację dwóch istotnych algorytmów wykorzystywanych w AOG: śledzenia promieni i wspomagania rozwiązywania układów równań oświetlenia w metodzie energetycznej. Opisywana struktura umożliwia implementację tych algorytmów z użyciem arytmetyki resztowej, co powoduje wzrost częstotliwości taktowania rzędu 20%. Podczas projektowania układu szczególną uwagę przywiązano do zachowania cech pozwalających na efektywną implementację AOG. Szczególnie istotne jest odizolowanie fragmentów układu wykorzystujących HRNS od pozostałych bloków procesora. Umożliwia to wykorzystanie zalet arytmetyki resztowej przy jednoczesnym zachowaniu możliwości wykonywania operacji trudnych.

Proponowana struktura składa się z bloków stanowiących samodzielne jednostki z prostymi i wydajnymi magistralami komunikacyjnymi. Zadania wykonywane przez poszczególne jednostki są typowymi zadaniami występującymi w implementacjach AOG. Dodanie dodatkowych operacji, np.: cieniowania, można więc zrealizować wprowadzając do układu dodatkowe podukłady realizujące odpowiedni algorytm.

Przykład implementacji kompletnego procesora zaprezentowano w pracy [Sch06]. Jest to rozwią-

zanie przeznaczone do implementacji w układach ASIC i charakteryzujące się znacznymi wymaganiami sprzętowymi. Standardowa konfiguracja wymaga 192 jednostek FPU, 822 kB rejestrów i 272 kB pamięci podręcznej. W pracy [Sch06] opisano także okrojoną, silnie optymalizowaną implementację prototypu z użyciem układów rodziny Virtex 2 firmy Xilinx. Przedstawione w tej pracy rozwiązanie oparte o arytmetykę resztową charakteryzuje się znacznie mniejszymi wymaganiami sprzętowymi. Ograniczenie zajmowanego obszaru pozwala na zastosowanie szerokiej gamy różnorodnych wersji dostępnych matryc FPGA o mniejszych możliwościach, a więc i koszcie. W przypadku implementacji w układach FPGA o wysokiej wydajności zastosowanie jednostek wykorzystujących arytmetykę resztową umożliwia upakowanie większej liczby jednostek w układzie, co bezpośrednio przekłada się na wydajność.

4.4.1 Założenia

Podstawowym celem prezentowanego układu jest wykazanie przydatności arytmetyki resztowej w sprzętowych implementacjach AOG. Układ ten powinien jednak charakteryzować się cechami porównywalnymi z dotychczas opracowanymi procesorami wspomagającymi. Można to osiągnąć poprzez wykorzystanie podstawowych mechanizmów pozwalających na efektywną implementację AOG. W opisywanej strukturze przyjęto następujące założenia:

- opis modeli w postaci siatek trójkątów,
- stosowanie drzew kd do opisu świata w metodzie śledzenia promieni,
- przeprowadzanie obliczeń równocześnie dla wiązki spójnych promieni,
- zastosowanie arytmetyki stałoprzecinkowej,
- łatwą skalowalność,
- nieskomplikowaną sieć połączeń,
- przechowywanie lokalnych kopii wielokrotnie wykorzystywanych danych.

Większość dotychczas opracowanych sprzętowych implementacji algorytmów grafiki komputerowej dotyczy opisów modeli w postaci siatek trójkątów. Rozwiązanie to pozwala znacznie uprościć struktury układów przy zachowaniu możliwości modelowania różnorodnych brył geometrycznych. Również w proponowanej architekturze zastosowane układy przeglądania drzew zawierających

opis świata oraz testowania przecięć zostały zoptymalizowane pod kątem stosowania trójkątów jako prymitywów geometrycznych. Ponieważ jednak jednostki te są układami programowalnymi, istnieje możliwość ich przyszłych modyfikacji w kierunku poszerzenia zestawu obsługiwanych powierzchni.

Drzewa kd są odmianą drzew BSP (rozdz. 2.1, str. 31), w których płaszczyzny dzielące są prostopadłe do osi układu współrzędnych. Pozwala to znacznie uprościć algorytmy konstruowania i przeglądania takich drzew. Szybkie i nieskomplikowane algorytmy przeglądania drzew kd można zaimplementować za pomocą niewielkich i wydajnych struktur sprzętowych.

W pracy [Sch06] wykazano, że sąsiednie promienie przecinają z reguły ten sam zbiór węzłów drzewa kd. Dotyczy to zarówno promieni pierwotnych przechodzących przez punkt obserwatora, jak i wtórnych modelujących odbicia i załamania światła. Celowe jest zatem przeprowadzanie testów przecięć wszystkich promieni z niewielkiej, spójnej wiązki z tym samym zbiorem trójkątów. W prezentowanym układzie zestaw kilku jednostek obliczeniowych kontrolowany jest przez jedną jednostkę sterującą. Każda z jednostek obliczeniowych przeprowadza testy przecięć pojedynczego promienia ze zbiorem trójkątów dostarczanym przez jednostkę zarządzającą do wszystkich jednostek obliczeniowych równocześnie.

Zastosowanie arytmetyki stałoprzecinkowej powoduje znaczne zmniejszenie obszaru wymaganego przez układy arytmetyczne w porównaniu z rozwiązaniami wykorzystującymi arytmetykę zmiennoprzecinkową. Cecha ta jest szczególnie istotna w przypadku implementacji w FPGA, gdzie zasoby logiczne są ograniczone. Niewielki koszt układów arytmetycznych pozwala na zastosowanie większej ich liczby. Ponieważ AOG charakteryzują się bardzo dobrą skalowalnością w implementacjach na maszynach równoległych, zwiększenie liczby jednostek obliczeniowych przekłada się bezpośrednio na wydajność.

Aby zwiększanie liczby równolegle pracujących jednostek powodowało odpowiedni wzrost wydajności, konieczne jest zaprojektowanie układu w postaci zapewniającej łatwą skalowalność. Charakter implementowanych algorytmów umożliwia użycie struktury pozwalającej na łatwą rozbudowę. Podstawową cechą umożliwiającą skalowalność jest wyizolowanie operacji kosztownych (np. dzielenie i konwersja pomiędzy HRNS a U2) i ich implementacja w postaci bloków wspólnych dla pewnej liczby jednostek arytmetycznych. Korzystne jest także zastosowanie prostych magistral połączeniowych umożliwiających transmisję w trybie rozgłaszania oraz adresowania indywidualnego. Dzięki temu możliwe jest dodawanie kolejnych układów obliczeniowych bez dodatkowych kosztów.

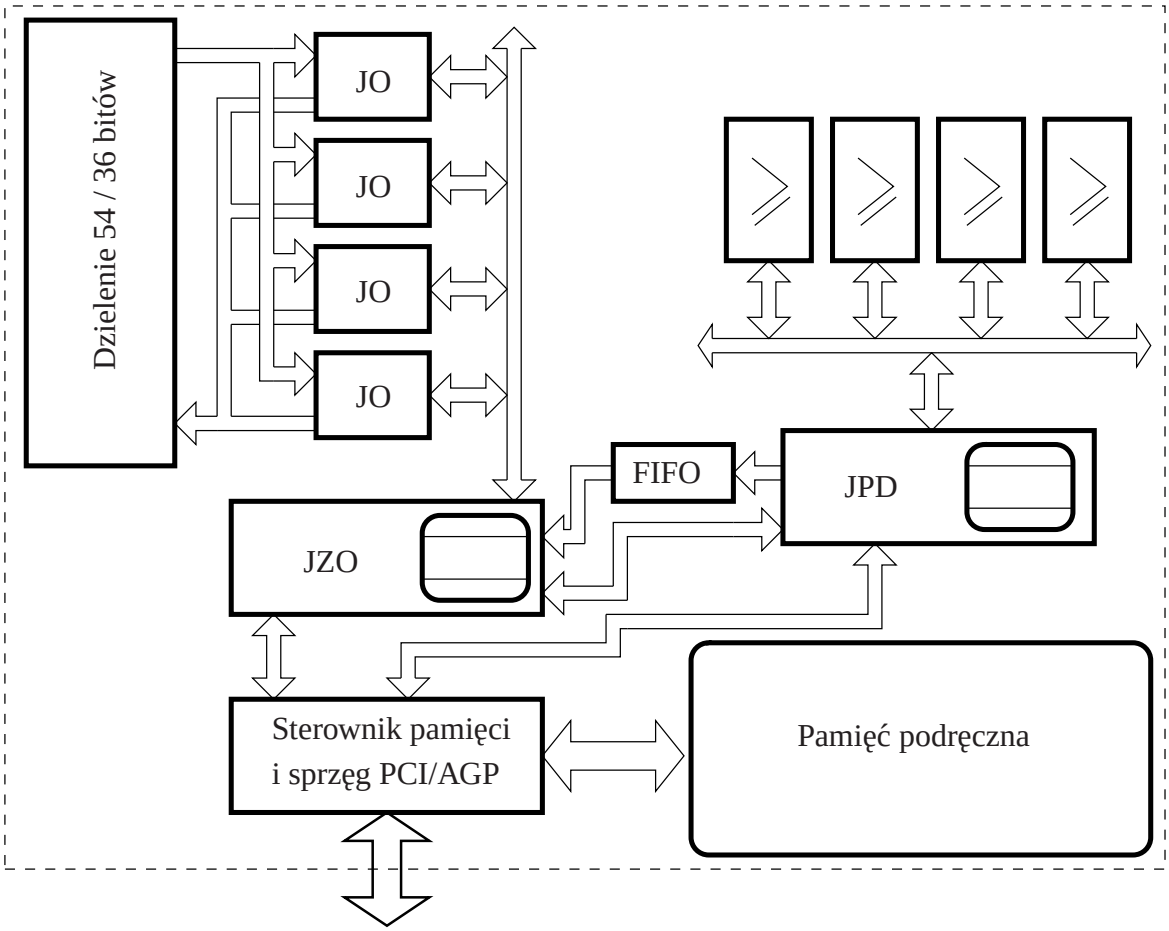
Właściwością algorytmów AOG jest intensywna komunikacja z pamięcią powodującą przeciążenia najszybszych obecnie dostępnych magistral komunikacyjnych. Ponieważ jednak większość odwołań charakteryzuje się dobrą lokalnością przestrzenną i czasową, przechowywanie kopii aktualnie wykorzystywanych danych pozwala na znaczne zmniejszenie obciążenia magistral komunikacji z pamięcią. W prezentowanym rozwiązaniu przechowywanie kopii wykorzystywanych danych realizowane jest na dwóch poziomach. Dane używane wielokrotnie przez konkretną jednostkę są przesyłane jednokrotnie i zapamiętywane w rejestrach lokalnych danej jednostki. Dodatkowo przewidziano stosowanie pamięci podręcznej umożliwiającej przechowanie pewnego podzbioru najczęściej używanych danych o rozmiarze znacznie przekraczającym pojemność pliku rejestrów. Przykładem może być zawartość kilku poziomów drzewa opisu świata, która jest wymagana dla każdego promienia w algorytmie śledzenia promieni.

4.4.2 Struktura procesora

Na rys. 4.29 przedstawiono schemat proponowanej struktury procesora wspomagającego obliczenia w AOG. Struktura ta jest modyfikacją architektury opisanej w [Tom05c]. Podstawowym elementem procesora jest zestaw jednostek obliczeniowych JO zawierających układy mnożenia akumulacyjnego wraz z zestawem rejestrów oraz automatem sterującym. Automat sterujący zrealizowano jako jednostkę programowalną pozwalającą na implementację różnorodnych zadań. Jednostki obliczeniowe są używane jako podstawowe elementy implementujące algorytmy wyznaczania punktu przecięcia prostej z trójkątem oraz wspomagania rozwiązywania układów równań oświetlenia.

W algorytmie wyznaczania punktu przecięcia zachodzi konieczność wyznaczenia ilorazu, dlatego też dla kilku jednostek obliczeniowych przewidziano potokowy układ dzielenia. W zależności od stosunku wydajności jednostek obliczeniowych do wydajności jednostki dzielenia liczba jednostek dzielenia może być dowolnie zmieniana.

Każda jednostka obliczeniowa wyposażona jest w prosty automat sterujący zawierający zestaw zaimplementowanych algorytmów. Za wybór odpowiedniego z nich oraz dostarczenie wymaganych danych odpowiedzialna jest jednostka zarządzania obliczeniami JZO. Jednostka zarządzająca jest także zrealizowana w postaci programowalnego automatu wyposażonego w dodatkowy plik rejestrów. Wszystkie JO są połączone z JZO za pomocą jednej magistrali umożliwiającej transmisję w trybie rozgłaszania i adresowania bezpośredniego.



Rysunek 4.29. Struktura procesora.

Wyznaczenie punktu przecięcia konkretnego promienia z odpowiednim elementem sceny wymaga dwóch kroków. Pierwszym z nich jest wybór potencjalnych kandydatów spośród wszystkich elementów występujących na scenie, drugim krokiem jest wyznaczenie punktów przecięć wybranych elementów z promieniem i wybór spośród nich punktu najbliższego obserwatora. Za pomocą JO realizowany jest krok drugi, natomiast wybór kandydatów dokonywany jest w jednostce przeglądania drzewa JPD.

Zaimplementowanym algorytmem przeglądania drzew jest algorytm zaprezentowany w pracy [HKBZ97]. Jego cechą szczególną jest odpowiednio dobrana kolejność działań pozwalająca w 75% przypadków na wybór odpowiedniego węzła drzewa na podstawie wyłącznie prostych porównań dwóch liczb. W pozostałych 25% przypadków konieczne jest obliczenie punktu przecięcia promienia z płaszczyzną dzielącą dany węzeł. Z powodu ograniczonych zasobów sprzętowych jednostka przeglądania drzewa zawiera wyłącznie zestaw komparatorów pozwalających na wykonanie porównań współrzędnych. W przypadku konieczności wyznaczenia punktu przecięcia zadanie to jest realizowa-

ne przez jednostki obliczeniowe.

W pracy [Sch06] podano, że w implementacji algorytmu śledzenia promieni liczba jednostek przeglądania drzewa powinna być większa od liczby jednostek wyznaczania przecięć. W rzeczywistości z rys. 2.26 na str. 42 w pracy [Sch06] wynika, że stosunek koniecznych do wykonania operacji przeglądania drzewa do operacji wyznaczenia punktu przecięcia zależy od przyjętej maksymalnej głębokości drzewa i złożoności sceny. Struktura z rys. 4.29 zawiera taką samą liczbę JPD, jak JO. Uzasadnieniem takiej decyzji jest kilka argumentów. Po pierwsze, JPD są układami uproszczonymi, a więc ciężar obliczeń przejmują i tak jednostki obliczeniowe. Jednostka przeglądania drzewa jest wyłącznie układem dodatkowym o niskim koszcie złożonym z prostego automatu sterującego i zestawu komparatorów. Po drugie, struktura z rys. 4.29 jest rozwiązaniem bardziej elastycznym, ponieważ JO są wykorzystywane także w implementacji algorytmów opartych o metody energetyczne. Po trzecie, zwiększenie liczby jednostek testowania przecięć w stosunku do JPD pozwala na zmniejszenie głębokości drzewa, co bezpośrednio przekłada się na łatwość jego tworzenia i może uprościć generowanie obrazów dynamicznych.

Komunikacja pomiędzy JPD a JZO zrealizowana jest za pomocą dwóch kanałów. Pierwszy z nich jest kanałem jednokierunkowym z buforem FIFO (*ang.* ***Fi****rst* ***I****n* ***F****irst* ***O****ut*), drugi zawiera magistralę dwukierunkową. Pierwszy kanał jest wykorzystywany do przekazywania współrzędnych trójkątów, dla których należy wykonać testy przecięć z promieniami. Zadaniem drugiego kanału jest szybka komunikacja z pominięciem kolejki FIFO umożliwiająca natychmiastowe obliczenie punktu przecięcia promieni z płaszczyzną dzielącą węzeł drzewa kd.

Zarówno zestaw jednostek obliczeniowych, jak i blok przeglądania drzewa wymagają danych z pamięci zewnętrznej. W związku z tym konieczny jest dodatkowy blok sterownika pamięci oraz układ pozwalający na komunikację z systemem nadrzędnym. Poza tym przewidziano zastosowanie pamięci podręcznej, ponieważ w przypadku implementacji sprzętowej algorytmu śledzenia promieni jej stosowanie pozwala znacznie zmniejszyć natężenie komunikacji z pamięcią zewnętrzną. Dokładne struktury sterownika pamięci i sprzęgu PCI/AGP nie zostały opracowane ze względu na istnienie wielu gotowych rozwiązań w tym zakresie, także komercyjnych.

Układ przedstawiony na rys. 4.29 umożliwia wykorzystanie możliwości równoległego przetwarzania w celu zwiększenia wydajności na dwa sposoby. Zwiększenie liczby JO zarządzanych przez pojedynczą JZO pozwala na zwiększenie liczby promieni przetwarzanych w ramach pojedynczej wiązki. Równocześnie można dodać dodatkowe zestawy składające się z jednostek obliczeniowych

sterowanych przez JZO oraz dodatkowych JPD, których zadaniem będzie prowadzenie obliczeń dla kolejnych wiązek promieni. Jedynymi ograniczeniami liczby takich bloków są dostępne zasoby sprzętowe oraz stopień komplikacji sterownika pamięci i pamięci podręcznej.

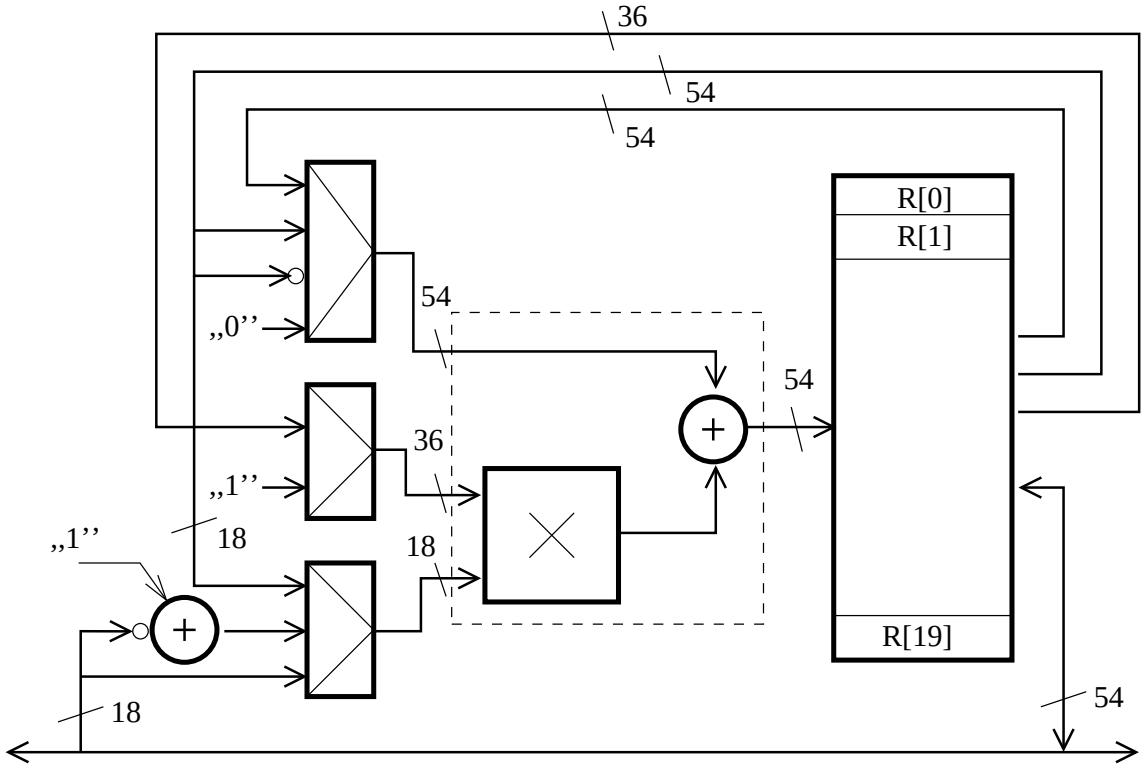
Podstawowym elementem w prezentowanym procesorze jest zestaw JO wraz z JZO. Blok ten jest elementem odpowiedzialnym za przeprowadzanie obliczeń w algorytmach wyznaczania punktu przecięcia prostej z trójkątem oraz rozwiązywania układu równań oświetlenia. Pozostałe jednostki procesora są albo układami wspomagającymi o prostej strukturze, albo standardowymi jednostkami, np.: sterownik pamięci czy pamięć podręczna. Poniżej zostaną opisane wyłącznie JO i JZO oraz przykłady ich wykorzystania do implementacji wyznaczania punktu przecięcia i rozwiązywania układu równań oświetlenia. Także prototypowa implementacja została ograniczona do dwóch wspomnianych jednostek.

Jednostka obliczeniowa JO

Podstawowym zadaniem jednostki obliczeniowej jest implementacja algorytmu wyznaczania punktu przecięcia promienia z trójkątem oraz wspomaganie procesu rozwiązywania równań oświetlenia metodami iteracyjnymi. Ponieważ podstawowymi operacjami arytmetycznymi w obu tych zadaniach są różne kombinacje mnożenia i dodawania, są one realizowane z użyciem układu mnożenia akumulacyjnego. Jednostka obliczeniowa składa się z jednostki mnożenia akumulacyjnego $k \times 2k$ bitów, pliku 20 rejestrów $3k$–bitowych, zestawu multiplekserów podających odpowiednie dane na wejścia jednostki MAC oraz programowalnego układu sterującego. Schemat połączeń układu MAC, multiplekserów i pliku rejestrów pokazano na rys. 4.30. Zastosowanie wieloportowego pliku rejestrów pozwala na dostarczenie kompletu danych dla układu MAC w każdym cyklu zegarowym. Dzięki temu np.: implementacja algorytmu testowania punktu przecięcia promienia z trójkątem według pomysłu z pracy [MT97] wymaga jedynie 27 instrukcji.

Przepływ danych w strukturze z rys. 4.30 jest sterowany przez prosty automat sterujący. Automat ten jest zrealizowany jako jednostka programowalna z programem zapisanym w niewielkiej pamięci ROM. Listę zaimplementowanych rozkazów przedstawiono w tabeli 4.6. Większość rozkazów to rozkazy arytmetyczne dobrane w sposób umożliwiający efektywną implementację algorytmu wyznaczania punktu przecięcia promienia z trójkątem.

Wynik każdego rozkazu z tab. 4.6 jest zapisywany w rejestrze oznaczonym R[rez]. Argumentami mogą być dwa rejestry różne od rejestru wynikowego oznaczone jako R[dod] i R[mnoz] oraz dana z



Rysunek 4.30. Struktura układu arytmetycznego jednostki obliczeniowej. Dane wejściowe są reprezentowane jako wektory 18–bitowe w kodzie $U2$.

magistrali zewnętrznej, lub trzy rejestry, z których jeden jest jednocześnie miejscem przechowywania wyniku. Rozkaz RDWR oznacza przejście do stanu, w którym jednostka zarządzająca ma możliwość zapisu i odczytu poszczególnych rejestrów JO.

Tabela 4.6. Lista rozkazów układu sterującego jednostką obliczeniową

Rozkaz	Operacja
RDWR	zapis/odczyt pliku rejestrów
MADD	R[res] $\Leftarrow$ dana $\times$ R[mnoz] + R[dod]
MUL	R[res] $\Leftarrow$ dana $\times$ R[mnoz]
MSUB	R[res] $\Leftarrow$ dana $\times$ R[mnoz] $-$ R[dod]
INV_MSUB	R[res] $\Leftarrow -$ (dana $\times$ R[mnoz]) $+$ R[dod]
INV_SUB	R[res] $\Leftarrow -$ dana $+$ R[dod]
MUL_REG	R[res] $\Leftarrow$ R[dod] $\times$ R[mnoz]
ACC_MUL_REG	R[res] $\Leftarrow$ R[dod] $\times$ R[mnoz] $+$ R[res]

Część kombinacyjna układu z rys. 4.30, czyli jednostka MAC wraz z multiplekserami, podzie-

lona jest za pomocą rejestrów na trzy etapy. Na schemacie nie zaznaczono rejestrów znajdujących się wewnątrz układu mnożenia akumulacyjnego ze względu na chęć zachowania czytelności rysunku. Poza tym, dokładne umiejscowienie tych rejestrów zależy od przyjętej struktury układu MAC i założonej szerokości słowa. Cała JO jest układem potokowym o głębokości potoku równej trzy, tak więc każda z instrukcji zamieszczonych w tab. 4.6 wymaga trzech cykli zegara. Dalsze zwiększanie głębokości potoku jest nieopłacalne ze względu na występowanie dużej liczby konfliktów danych w implementowanych algorytmach.

Dodatkowym elementem JO jest zbiór rejestrów k–bitowych oraz komparator k–bitowy używane w algorytmie śledzenia promieni do wyboru i zapamiętania współrzędnych najbliższego punktu przecięcia z trójkątem. Zadaniem tych elementów jest porównanie wyniku zwróconego przez jednostkę dzielącą z najmniejszym wynikiem wytworzonym dotychczas. Jeśli nowy wynik opisuje punkt bliższy, dotychczas zapamiętana wartość jest zastępowana rezultatem bieżącym.

Jednostka zarządzania obliczeniami JZO

Jednostka zarządzania obliczeniami realizuje szereg zadań związanych ze sterowaniem i komunikacją z jednostkami obliczeniowymi. Składa się z zestawu rejestrów oraz prostego programowalnego automatu sterującego, którego instrukcje stanowią głównie rozkazy transmisji danych. Komunikacja pomiędzy JZO a JO może być przeprowadzona w dwóch trybach. W trybie rozgłaszania wszystkie JO operują na danych dostarczonych przez JZO, np.: zapisują tę samą wartość do rejestru o tym samym adresie. W trybie adresowania indywidualnego dane przesyłane są pomiędzy JZO a pojedynczą JO. Pozostałe JO nie mogą w tym czasie korzystać z magistrali komunikacyjnej.

Pojedyncza JZO może zarządzać dowolną liczbą JO. Ponieważ jednak komunikacja w trybie adresowania indywidualnego uniemożliwia korzystanie z magistrali przez pozostałe JO, liczba JO sterowanych przez JZO nie powinna być zbyt duża.

Poza zarządzaniem JO, zadaniem JZO jest także komunikacja z jednostką przeglądania drzewa oraz pamięcią zewnętrzną. W algorytmie śledzenia promieni JZO pobiera dane z JPD na dwa sposoby: z użyciem bufora FIFO lub z jego pominięciem (połączenie bezpośrednie). Decyzja o priorytecie poszczególnych kanałów podejmowana jest przez JZO, co pozwala na efektywne zarządzanie zasobami obliczeniowymi JO. Ponieważ zasoby sprzętowe wymagane przez JO stanowią większość zasobów całego układu, niezmiernie istotne jest zapewnienie ciągłej ich pracy.

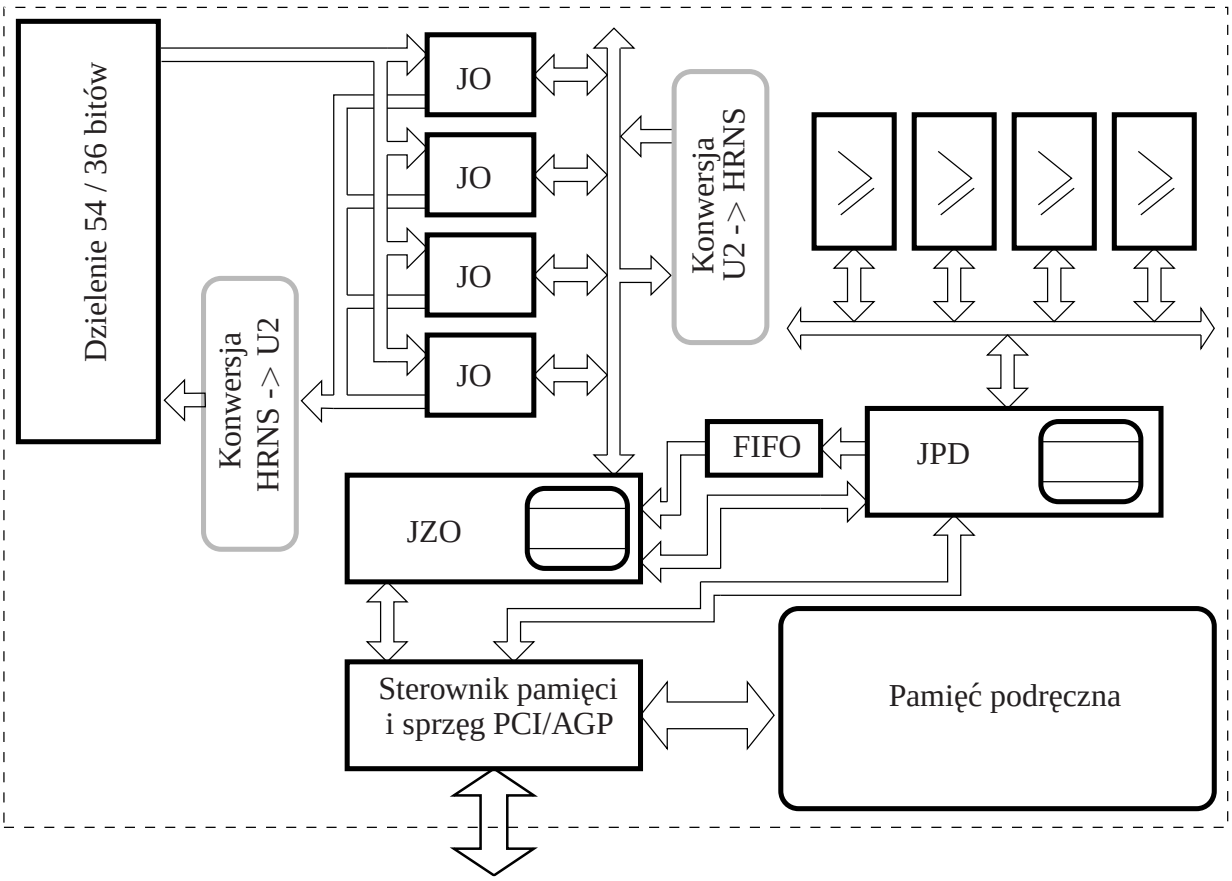
4.4.3 Zastosowanie arytmetyki resztowej

Konstruowanie efektywnych układów z użyciem arytmetyki resztowej wymaga spełnienia dwóch warunków. Pierwszym z nich jest ograniczenie zbioru wykonywanych działań w torze RNS do dodawania, odejmowania i mnożenia ze względu na wysoki koszt implementacji pozostałych operacji. Drugim warunkiem jest dążenie do zwiększenia stosunku liczby wykonywanych operacji arytmetycznych w RNS do liczby konwersji pomiędzy RNS a systemem pozycyjnym z powodu kosztu konwerterów. Struktura z rys. 4.29 może zostać łatwo zmodyfikowana do postaci pozwalającej na wykorzystanie arytmetyki resztowej.

W proponowanym rozwiązaniu arytmetyka resztowa jest używana wyłącznie w jednostkach obliczeniowych zawierających układ mnożenia akumulacyjnego. Dzięki temu możliwe jest zwiększenie częstotliwości ich taktowania o ok. 20%. Ponieważ komunikacja pomiędzy JO a resztą układu realizowana jest za pomocą pojedynczej magistrali, wystarczy zastosowanie jednego konwertera U2 do RNS na każdą jednostkę JZO oraz konwertera RNS do U2 na każdy układ dzielenia. Przykład struktury procesora uzupełnionej o dodatkowe konwertery przedstawiono na rys. 4.31.

Wprowadzenie dodatkowych konwerterów wiąże się z wydłużeniem potoku o liczbę cykli wymaganą do przeprowadzenia konwersji. Opóźnienie wnoszone przez układy konwersji jest co najwyżej dwukrotnie większe od długości ścieżki krytycznej jednostki MAC, tak więc liczba cykli wymaganych do przeprowadzenia konwersji pomiędzy HRNS a U2 jest rzędu 6–7 taktów zegara. Jednokrotne wykonanie procedury wyznaczania punktu przecięcia wymaga 27 cykli zegara i czasu wymaganego na wykonanie dzielenia, a dla każdego promienia testów takich należy przeprowadzić co najmniej kilka lub kilkanaście, zależnie od stopnia złożoności sceny i jakości drzewa kd. Oczekiwanie na wypełnienie potoku jest wymagane jedynie w momencie zmiany zawartości rejestrów JO zawierających współrzędne promienia. Dodatkowe kilka cykli potrzebnych na całkowite wypełnienie potoku może więc zostać pominięte ze względu na zwiększenie częstotliwości taktowania o ok. 20%.

Konieczność stosowania konwerterów pomiędzy HRNS a U2 powoduje zwiększenie zajmowanego obszaru, nie są to jednak wszystkie koszty spowodowane użyciem HRNS. W przypadku reprezentacji w kodzie U2 liczba bitów wykorzystywana do zapisania konkretnej liczby zależy od jej wartości. Możliwe jest zatem ograniczenie szerokości części magistral połączeniowych. Dla układów wykorzystujących arytmetykę resztową wszystkie magistrale połączeniowe muszą mieć szerokość wystarczającą do przesłania liczby o wartości maksymalnej w danym RNS. Wiąże się to także ze



Rysunek 4.31. Struktura procesora wykorzystującego HRNS.

zwiększeniem szerokości stosowanych multiplekserów. Dla układu zaprezentowanego na rys. 4.30 zastosowanie HRNS powoduje wzrost szerokości wszystkich magistral do 58 bitów.

Obszar zajęty przez resztową jednostkę MAC jest mniejszy od obszaru jednostki w kodzie U2. Użycie ok. 10 jednostek obliczeniowych w HRNS kompensuje koszty dodatkowego obszaru wymaganego dla konwersji pomiędzy HRNS a U2. Dla mniejszej liczby JO użycie HRNS także jest opłacalne, ponieważ narzut w stosunku do obszaru całego układu, wliczając sterowniki pamięci i pamięć podręczną, jest równy co najwyżej kilka procent. Ze względu na znaczny wzrost wydajności, stosowanie resztowych systemów liczbowych w zaprezentowanej strukturze jest opłacalne i uzasadnione. W prototypowej implementacji w układzie XC2S200-6FG456 z użyciem środowiska Xilinx ISE Foundation 6.3i zastosowanie resztowej jednostki arytmetycznej w układzie z rys. 4.30 spowodowało wzrost częstotliwości taktowania z 77.3 MHZ do 92.5 MHZ, a więc o ok. 20%.

4.4.4 Implementacje algorytmów oświetlenia globalnego

Uniwersalna struktura JO i JZO pozwala na implementację różnorodnych algorytmów obliczeniowych. Poniżej zostanie podany przykład implementacji algorytmu wyznaczania punktu przecięcia prostej z trójkątem według koncepcji z [MT97]. Podane zostaną także zalecenia dotyczące zastosowania prezentowanej struktury do wspomagania sprzętowego algorytmów wykorzystujących metody energetyczne.

Wyznaczanie punktu przecięcia promienia z trójkątem

Wyznaczanie punktu przecięcia promienia z trójkątem przeprowadzane jest dla wiązki kilku promieni równocześnie. W prezentowanym układzie każda JO zawiera zestaw rejestrów lokalnych, w których zapamiętane są współrzędne jednego promienia oraz wyniki pośrednie. Następnie współrzędne trójkątów wytypowanych w procesie przeglądania drzewa kd są rozsyłane do wszystkich JO równocześnie. Każda JO zawiera dodatkowo zbiór rejestrów pozwalający na zapamiętanie współrzędnych punktu przecięcia promienia z najbliższym trójkątem. Po wykonaniu obliczeń dla pełnego zbioru badanych trójkątów wyniki te są przesyłane do JZO do dalszego wykorzystania.

Wyznaczanie punktu przecięcia prostej z trójkątem w przestrzeni $\mathbb{R}^3$ przeprowadzane jest według algorytmu z pracy [MT97]. Przykładową implementację przedstawiono w rozprawie na stronie 30 (alg. 1). W oryginalnym opracowaniu alg. 1 jest implementowany programowo. W implementacji sprzętowej proponowanej w rozprawie wprowadzono kilka modyfikacji dotyczących kolejności wykonywania działań. Podstawową różnicą pomiędzy implementacją programową a sprzętową jest przesunięcie wszelkich rozgałęzień na koniec procedury. Dzięki temu możliwe staje się użycie jednostki potokowej o ustalonej ścieżce przetwarzania bez konieczności wykrywania sytuacji przedwczesnego przerwania obliczeń. Dodatkową korzyścią jest przesunięcie wszelkich operacji porównywania poza tor obliczeń, co ułatwia implementację części arytmetycznej układu z użyciem arytmetyki resztowej.

Na listingu 4.5 przedstawiono przykład implementacji algorytmu wyznaczania punktu przecięcia promienia z trójkątem. Przed rozpoczęciem algorytmu w rejestrach R0, R1, R2 muszą zostać zapamiętane współrzędne (K_x, K_y, K_z) wektora kierunkowego promienia, a w rejestrach R12, R13 i R14 współrzędne (O_x, O_y, O_z) punktu początkowego promienia. Wynikiem algorytmu są wartości δ, u', v', t' zawarte w rejestrach R15, R16, R17, R18. W trakcie wykonywania algorytmu przez JO zadaniem JZO jest wystawianie na magistralę danych odpowiednich współrzędnych punktu $\mathbf{A}$ oraz

Listing 4.5. Bezpośrednia implementacja algorytmu wyznaczania punktu przecięcia promienia z trójkątem

```
S1  : R[8] = dane *R[1]              // R[8]  += E2.x * K.y
S2  : R[7] = dane *R[2]              // R[7]  += E2.x * K.z
NOP
S3  : R[8] = dane * R[0] - R[8]      // R[8]  =  S.z
S4  : R[6] = dane *R[2]              // R[6]  += E2.y * K.z
S5  : R[7] = R[7] - dane * R[0]      // R[7]  =  S.y
S6  : R[6] = dane * R[1] - R[6]      // R[6]  =  S.x
S7  : R[15] = dane *R[6]             // R[15] += E1.x * S.x
NOP
S8  : R[15] = dane * R[7] + R[15]    // R[15] += E1.y * S.y
NOP
S9  : R[15] += dane * R[8]           // R[15] += E1.z * S.z  :  δ
S10 : R[3] = R[12] - dane            // R[3]  =  T.x
S11 : R[4] = R[13] - dane            // R[4]  =  T.y
S12 : R[5] = R[14] - dane            // R[5]  =  T.z
S13 : R[16] = R[3] * R[6]            // R[16] =  T.x * S.x
NOP
S14 : R[16] += R[4] * R[7]           // R[16] += T.y * S.y
NOP
S15 : R[16] += R[5] * R[8]           // R[16] += T.z * S.z   :  u'
NOP
S16 : R[11] = dane *R[4]             // R[11] =  E1.x * T.y
S17 : R[10] = dane *R[5]             // R[10] =  E1.x * T.z
S18 : R[11] = dane * R[3] - R[11]    // R[11] =  Q.z
S19 : R[9] = dane *R[5]              // R[9]  =  E1.y * T.z
S20 : R[10] = R[10] - dane * R[3]    // R[10] =  Q.y
NOP
S21 : R[9] = dane * R[4] - R[9]      // R[9]  =  Q.x
S22 : R[17] = R[0] * R[9]            // R[17] =  K.x * Q.x
NOP
S23 : R[17] += R[1] * R[10]          // R[17] += K.y * Q.y
NOP
S24 : R[17] += R[2] * R[11]          // R[17] += K.z * Q.z   :  v'
S25 : R[18] = dane *R[9]             // R[18] =  E2.x * Q.x
S26 : R[18] = dane * R[10] + R[18]   // R[18] += E2.y * Q.y
S27 : R[18] = dane * R[11] + R[18]   // R[18] += E2.z * Q.z  :  t'
```

Listing 4.6. Implementacja algorytmu wyznaczania punktu przecięcia promienia z trójkątem

```
S12 : R[5] = R[14] – dane          // R[5]  =  T.z
S1  : R[8] = dane ∗R[1]            // R[8]  += E2.x ∗ K.y
S2  : R[7] = dane ∗R[2]            // R[7]  += E2.x ∗ K.z
S4  : R[6] = dane ∗R[2]            // R[6]  += E2.y ∗ K.z
S19 : R[9] = dane ∗R[5]            // R[9]  =  E1.y ∗ T.z
S11 : R[4] = R[13] – dane          // R[4]  =  T.y
S3  : R[8] = dane ∗ R[0] – R[8]    // R[8]  =  S.z
S17 : R[10] = dane ∗R[5]           // R[10] =  E1.x ∗ T.z
S10 : R[3] = R[12] – dane          // R[3]  =  T.x
S21 : R[9] = dane ∗ R[4] – R[9]    // R[9]  =  Q.x
S6  : R[6] = dane ∗ R[1] – R[6]    // R[6]  =  S.x
S5  : R[7] = R[7] – dane ∗ R[0]    // R[7]  =  S.y
S16 : R[11] = dane ∗R[4]           // R[11] =  E1.x ∗ T.y
S22 : R[17] = R[0] ∗ R[9]          // R[17] =  K.x ∗ Q.x
S25 : R[18] = dane ∗R[9]           // R[18] =  E2.x ∗ Q.x
S20 : R[10] = R[10] – dane ∗ R[3]  // R[10] =  Q.y
S7  : R[15] = dane ∗R[6]           // R[15] += E1.x ∗ S.x
S13 : R[16] = R[3] ∗ R[6]          // R[16] =  T.x ∗ S.X
S18 : R[11] = dane ∗ R[3] – R[11]  // R[11] =  Q.z
S23 : R[17] += R[1] ∗ R[10]        // R[17] += K.y ∗ Q.y
S26 : R[18] = dane ∗ R[10] + R[18] // R[18] += E2.y ∗ Q.y
S8  : R[15] = dane ∗ R[7] + R[15]  // R[15] += E1.y ∗ S.y
S14 : R[16] += R[4] ∗ R[7]         // R[16] += T.y ∗ S.y
S24 : R[17] += R[2] ∗ R[11]        // R[17] += K.z ∗ Q.z   :  v'
S27 : R[18] = dane ∗ R[11] + R[18] // R[18] += E2.z ∗ Q.z  :  t'
S9  : R[15] += dane ∗ R[8]         // R[15] += E1.z ∗ S.z  :  δ
S15 : R[16] += R[5] ∗ R[8]         // R[16] += T.z ∗ S.z   :  u'
```

wektorów $\mathbf{E}_1$ i $\mathbf{E}_2$.

Implementacja przedstawiona na listingu 4.5 wynika bezpośrednio z algorytmu 1. Ponieważ jednostka obliczeniowa jest układem potokowym, bezpośrednia implementacja powoduje powstanie dużej liczby konfliktów danych typu odczyt po zapisie. Konieczne jest więc wstawianie jałowych cykli pozwalających na propagację danych do odpowiednich rejestrów, z których następnie są odczytywane w kolejnej instrukcji. Jałowe cykle są oznaczone jako NOP, przy czym liczba jałowych cykli zależy od głębokości potoku i miejsca ich występowania. W prezentowanym przykładzie liczba ta mieści się w zakresie od 1 do 3 cykli.

Wprowadzanie jałowych cykli jest prostą techniką pozwalającą uniknąć błędów podczas konfliktów typu odczyt po zapisie, jednak wiąże się ze spadkiem wydajności. Korzystniejszym rozwiązaniem jest zmiana kolejności wykonywania operacji pozwalająca na uniknięcie konfliktów danych. Na list. 4.6 przedstawiono przykład implementacji algorytmu z pracy [MT97], w której zmiana kolejności wykonywania działań pozwala uniknąć wszystkich konfliktów danych. Warunkiem poprawności kodu jest ograniczenie maksymalnej głębokości potoku do trzech cykli. Większe głębokości powodują konieczność stosowania dodatkowych instrukcji pustych obniżających wydajność implementacji.

Po znalezieniu wartości δ, u', v', t' należy jeszcze wykonać dzielenia oraz zbadać znaki wyników. Dzielenie jest przeprowadzane w dodatkowej jednostce dzielącej. Ponieważ w algorytmie z pracy [MT97] liczba dzieleń jest zdecydowanie mniejsza od liczby mnożeń i dodawań, jednostkę dzielenia można zrealizować jako układ wspólny dla kilku jednostek obliczeniowych. Wydajność jednostki dzielenia musi pozwolić na przeprowadzenie trzech dzieleń dla każdej JO w czasie potrzebnym na wykonanie kodu z list. 4.6.

Wspomaganie metod energetycznych

Sprzętowe wspomaganie metod energetycznych może dotyczyć dwóch algorytmów: wyznaczania współczynników sprzężenia F_{ji} oraz rozwiązywania układu równań oświetlenia. W pierwszym z nich operacją zajmującą większość czasu jest określanie widoczności pomiędzy płatami, gdzie można wykorzystać algorytm śledzenia promieni. W celu zapewnienia spójności generowanej wiązki promieni należy przeprowadzać test widoczności pomiędzy zbiorami sąsiednich trójkątów.

Algorytm śledzenia promieni może być wykorzystany także dla obliczania konkretnych wartości współczynników sprzężenia metodą próbkowania półsześcianu (rys. 2.8, str. 40). Dla wybranego płatu elementarnego należy wygenerować zbiór promieni przechodzących przez wszystkie elementy wydzielone ze ścian półsześcianu. Z każdym promieniem należy związać odpowiednią wartość współczynnika sprzężenia. Wartość współczynnika sprzężenia pomiędzy płatem będącym początkiem promieni a dowolnym innym płatem i równa jest sumie współczynników dla promieni przecinających płat i. Znane są także metody wyznaczania wartości współczynników sprzężenia z użyciem metod probabilistycznych, gdzie śledzenie promieni odgrywa kluczową rolę [Hal01].

Drugim algorytmem implementowanym sprzętowo jest rozwiązywanie układu równań oświetlenia. Jakkolwiek przy użyciu metod szukania przybliżonych rozwiązań czas tej operacji jest znacznie mniejszy od poprzedniej, ze względu na rozmiar spotykanych równań warto rozważyć wspomaganie

także w tym przypadku. Wyznaczanie promienistości odbywa się według algorytmu progresywnego ulepszania [CW93]. Z każdym płatem elementarnym o indeksie i związana jest wartość energii padającej B_i, energii niewypromieniowanej ΔB_i oraz współczynnik odbicia ρ_j. W kolejnym kroku algorytmu wybiera się element z największą wartością ΔB_i, a następnie dla wszystkich płatów j należy obliczyć wartość otrzymanej energii

$$\Delta rad_j = \Delta B_i \cdot \rho_j \cdot F_{ji}, \tag{4.3}$$

o wartość której powiększają się ΔB_j oraz B_j. W ten sposób po niewielkiej liczbie kroków można znaleźć wartość promienistości dla wszystkich płatów z wystarczającą dokładnością. Algorytm ten wymaga niestety intensywnej komunikacji z pamięcią danych, a co za tym idzie, częstej konwersji w przypadku użycia arytmetyki resztowej. Rozsądną metodą jego realizacji wydaje się być wybranie bloku kilku (zależy od liczby jednostek pracujących równolegle) kolejnych czynników B_j, ΔB_j oraz ρ_j, a następnie zwiększenie B_j i ΔB_j o Δrad_j wyliczone zgodnie ze wzorem (4.3) dla kilku największych wartości ΔB_i. Metoda ta pozwala na zapamiętanie wielkości ΔB_i, ponieważ są one niezmienne aż do wykorzystania wszystkich elementów j. Dzięki przeprowadzaniu obliczeń w tej kolejności jedynymi danymi koniecznymi do transmisji w trakcie wykonywania obliczeń będą współczynniki F_{ji}.

4.4.5 Wyniki implementacji

Struktura bloku zawierającego JO została opisana w języku VHDL i zaimplementowana w układzie XC2S200-6FG456 z użyciem środowiska Xilinx ISE WebPack 6.3i. Do automatu sterującego JO zostały wpisane programy przedstawione na list. 4.5 i 4.6. Po syntezie i implementacji układ został przesymulowany z użyciem symulatora ModelSimXE III 6.0a. Opis w VHDL nie obejmuje jednostki dzielącej oraz mechanizmu komunikacji z tą jednostką. Jednostka zarządzająca została zaimplementowana bez interfejsu komunikacyjnego ze sterownikiem pamięci oraz JPD. Jedynym jej zadaniem jest dostarczanie odpowiednich współrzędnych do JO. Współrzędne te są zapamiętane w rejestrach lokalnych JZO.

Implementacja pojedynczej JO dla operandów zapisanych w 18–bitowym kodzie U2 wymaga 2029 tablic LUT, z czego 720 jest używanych jako rejestry wieloportowe. Kod programu z list. 4.6 zajmuje jedynie 37 tablic LUT, może więc być powielony dla każdej jednostki obliczeniowej. Układ XC2S200-6FG456 udostępnia 5292 cele logiczne zawierające jedną tablicę LUT każda, możliwe jest zatem upakowanie dwóch JO wraz z JZO w pojedynczym układzie. Liczba LUT zajętych przez

pojedynczą JO może zostać zmniejszona, jeśli do implementacji rejestrów zostaną użyte wbudowane w układy FPGA rodziny Spartan 2 bloki pamięci RAM. Niestety, rozmiar wbudowanych blokowych pamięci RAM umożliwia implementację jedynie części pliku rejestrów pojedynczej JO.

Rysunek 4.32. Porównanie częstotliwości taktowania różnych implementacji procesora wspomagającego AOG.

Struktura JO została zrealizowana jako jednostka potokowa o głębokości potoku równej 3 etapy. Maksymalna częstotliwość taktowania dla operandów w kodzie U2 wynosi 77.3 MHz. Po wymianie jednostki mnożenia akumulacyjnego na układ wykorzystujący HRNS $(2^{18}-1, 2^{18}, 2^{18}+1)$ maksymalna częstotliwość taktowania wzrosła do 92.5 MHz, co oznacza zysk równy 19.7%.

Używając procedury wyznaczania punktu przecięcia z list. 4.6 można co 27 cykli zegara wykonywać obliczenia dla kolejnych trójkątów. Pomijając czasy niezbędne dla zainicjowania rejestrów JO i zakładając częstotliwość taktowania równą 90 MHz, można osiągnąć teoretyczną maksymalną wydajność pojedynczej JO na poziomie ponad 3 300 000 testów na sekundę. Opóźnienie wynikające z konieczności inicjowania rejestrów JO można pominąć, ponieważ jest wykonywane dopiero po przetestowaniu pełnego zbioru trójkątów dla pojedynczego promienia.

Prototyp procesora zaprezentowanego w [Sch06] został zaimplementowany w układzie FPGA XC2V6000 rodziny Virtex 2 firmy Xilinx. Układ ten udostępnia 67584 tablice LUT, jest więc ponad dziesięciokrotnie większy od układu użytego do implementacji struktury z rys. 4.30. Szacowana wydajność procesora z [Sch06] jest rzędu 20 milionów testów na sekundę. Wartość ta może zostać osiągnięta w układzie złożonym z 8 JO, które łącznie zajmują poniżej 25% zasobów układu XC2V6000. Dodatkowo użycie układów rodziny Virtex 2 umożliwia zwiększenie częstotliwości taktowania. Zgodnie z danymi katalogowymi [Xil03a], [Xil03b] maksymalna częstotliwość taktowania układów rodziny Spartan 2 wynosi 200 MHz, a dla układów Virtex 2 420 MHz.

Układ z pracy [Sch06] jest rozwiązaniem rozbudowanym zawierającym zmiennoprzecinkowe jednostki arytmetyczne oraz wykonującym zestaw wszystkich operacji wymaganych do generacji obrazów metodą śledzenia promieni. Zadaniem tego rozdziału jest prezentacja metod wspomagania ob-

liczeń w AOG z użyciem arytmetyki resztowej. Pełne porównanie jest zatem niemożliwe, ponieważ w rozwiązaniu wykorzystującym arytmetykę resztową ograniczono się do implementacji jedynie niektórych fragmentów układu. Pozostałe bloki mogą mieć strukturę porównywalną z dotychczasowymi rozwiązaniami.

Podsumowując, zaproponowana architektura umożliwia implementację z użyciem arytmetyki resztowej kilku operacji występujących w AOG. Dzięki odpowiedniej strukturze wykonywanych działań możliwe jest odizolowanie operacji „trudnych" w RNS i tym samym skonstruowanie wydajnego układu. Zastosowanie HRNS w sprzętowych implementacjach AOG pozwala zwiększyć częstotliwość taktowania o ok. 20% w stosunku do układów wykorzystujących arytmetykę uzupełnieniową przy podobnym obszarze. Podane wyniki dotyczą porównania jednostek dla zakresu dynamicznego ok. 54 bitów. W prezentowanym rozwiązaniu stosowanie arytmetyki resztowej wiąże się z nieznacznym zwiększeniem długości potoku, które w większości przypadków można pominąć.

4.5 Podsumowanie

W rozdziale przedstawiono analizę wyników implementacji resztowych jednostek arytmetycznych, algorytmu ich generacji i kluczowych operacji AOG z użyciem arytmetyki resztowej. Zaimplementowany algorytm automatycznej generacji pozwala znaleźć strukturę jednostki o poszukiwanych parametrach AT. Zadanie to realizowane jest poprzez przegląd pewnego podzbioru wszystkich możliwych konfiguracji jednostki arytmetycznej. Złożoność algorytmu jest wykładnicza, jednak pozwala na pracę interakcyjną dla modułów o typowych wielkościach stosowanych w algorytmach DSP. Ograniczenie czasu wykonania algorytmu jest wynikiem początkowego wyeliminowania tych konfiguracji, których parametry AT można wstępnie oszacować jako nieużyteczne. Wynikiem algorytmu może być niewielki zbiór jednostek, wśród których znajduje się rozwiązanie o parametrach bliskich optymalnym. Dla wygenerowanych jednostek szacowane są ich parametry AT i tworzony jest opis struktury w języku VHDL. Liczba badanych jednostek rośnie wykładniczo z szerokością operandów wejściowych, jednak dla modułów kilkunastobitowych generacja pełnego zbioru rozwiązań wymaga czasu poniżej minuty, co pozwala na pracę interaktywną.

Dokładne oszacowanie charakterystyk AT jest trudne ze względu na ich zależność od algorytmów stosowanych w narzędziach syntezy. Co więcej, algorytmy te bardzo często są niedeterministyczne. Jest to szczególnie widoczne dla układów zawierających pamięci ROM o dużej pojemności. Jeśli

bezwzględnie konieczne jest znalezienie układu o najlepszych charakterystykach, po wygenerowaniu opisów w VHDL konieczna jest pełna implementacja pozwalająca dokładnie określić charakterystyki AT analizowanych jednostek. Implementacja kompletnego zbioru rozwiązań jest czasochłonna, możliwe jest jednak znaczne skrócenie tej operacji dla problemu polegającego na znalezieniu układu o minimalnym obszarze. Parametry generowanych jednostek są zależne przede wszystkim od sposobu generowania iloczynów częściowych. Jednostki, w których iloczyny częściowe są tworzone z użyciem układów mnożących $2 \cdot k$ lub $3 \cdot k$ bitów charakteryzują się najmniejszym obszarem oraz iloczynami AT i AT^2. W zbadanym zakresie zmienności modułu obszar przez nie zajmowany może być oszacowany przez liniową funkcję szerokości modułu.

Pomimo, iż niemożliwe jest określenie a priori struktury układu o żądanych parametrach AT, z przeprowadzonych eksperymentów wynika, że dla części przypadków można dość znacznie ograniczyć liczbę badanych struktur. Najprostsze jest to dla zadania polegającego na znalezieniu jednostki o minimalnym obszarze. Układ taki zawsze znajduje się w grupie rozwiązań, dla których iloczyny częściowe są obliczane za pomocą układów mnożących $2 \cdot k$ i $3 \cdot k$ bity. Co więcej, wśród tych układów z reguły znajduje się rozwiązanie o opóźnieniu większym od układu najszybszego o kilka–kilkanaście procent, ale o wielokrotnie mniejszym obszarze. Wytypowanie jednostki o najmniejszym obszarze może być przeprowadzone bardzo szybko, ponieważ miary zastosowane w algorytmie pozwalają na dokładne wzajemne porównanie obszaru jednostek. Dla zbadanych wartości modułu, znalezienie jednostki najmniejszej wymagało zaimplementowania najwyżej 10 układów. Jeszcze lepsze charakterystyki AT mają jednostki dla modułów o małej wartości okresu potęg 2 modulo. W tych przypadkach różnica długości ścieżki krytycznej pomiędzy najmniejszym a najszybszym rozwiązaniem staje się pomijalnie mała.

Jeżeli konieczne jest znalezienie układu o najkrótszej ścieżce krytycznej, należy zbadać znacznie większy podzbiór możliwych konfiguracji. Przeprowadzone doświadczenia pokazały, że w wielu przypadkach układ najszybszy jest w grupie układów, w których iloczyny częściowe są obliczane za pomocą pamięci ROM. Wynika to z dużego wpływu generatora wyniku na opóźnienie całej jednostki. Rezultatem obliczania iloczynów częściowych jako reszt modulo jest zmniejszenie szerokości wektora wejściowego generatora wyniku. Przekłada się to bezpośrednio na zmniejszenie głębokości kaskady reduktorów modulo.

Zaobserwowane zmniejszenie długości ścieżki krytycznej dotyczy jednak wyłącznie przypadków dla modułów o odpowiednio małych wartościach. Liczba iloczynów częściowych rośnie z kwadra-

tem szerokości modułu. Duża liczba iloczynów częściowych powoduje zarówno wzrost głębokości sumatora wstępnego, jak i szerokości wektora wejściowego generatora wyniku. Można zatem przypuszczać, że dla odpowiednio dużej wartości modułu obliczanie iloczynów częściowych za pomocą pamięci ROM jest nieopłacalne. Jednostki, w których iloczyny częściowe są wytwarzane za pomocą układów mnożenia $2 \cdot k$ bitów, różnią się między sobą wyłącznie strukturą generatora wyniku. Zadanie konstrukcji jednostki arytmetycznej dla dużych wartości modułu można więc ograniczyć do poszukiwania odpowiedniej postaci generatora wyniku.

Zależność obszaru jednostek najszybszych od szerokości modułu może być oszacowana, podobnie jak dla układów najmniejszych, funkcją liniową. Funkcja opisująca zależność długości ścieżki krytycznej od szerokości modułu ma charakter logarytmiczny. Obszar zajmowany przez jednostki najszybsze może być kilkukrotnie większy niż obszar dla jednostek najmniejszych. Różnica długości ścieżki krytycznej pomiędzy układem najszybszym a najszybszym spośród najmniejszych z reguły nie przekracza kilkunastu procent. Jeśli szybkość układu nie jest więc czynnikiem decydującym, można znacznie skrócić czas poszukiwania jednostki poprzez badanie tylko tych konfiguracji, w których nie są stosowane pamięci ROM w układzie wytwarzania iloczynów częściowych. Sytuacja taka występuje np. dla jednostek modulo moduły bazy resztowego systemu liczbowego. Ścieżka krytyczna większości z tych jednostek będzie krótsza od ścieżki krytycznej jednostki najwolniejszej.

Z porównania parametrów jednostek zaproponowanych w rozprawie z dotychczasowymi rozwiązaniami wynika, że w większości przypadków stosowanie nowych struktur jest korzystne. Przede wszystkim, w całym zakresie zmienności modułu zaproponowane rozwiązanie pozwala znaleźć układ o najmniejszym obszarze oraz iloczynach AT i AT^2. Co więcej, zależność obszaru od szerokości modułu dla nowych jednostek może być przybliżona funkcją liniową. W porównaniu z dotychczasowymi układami o złożoności wykładniczej, nowa metoda pozwala na stosowanie modułów o znacznie większych wartościach bez istotnego wzrostu kosztów całego układu. Przekłada się to bezpośrednio na zwiększenie zakresu dynamicznego RNS , umożliwia także wybór modułów o wartościach pozwalających na uproszczenie niektórych operacji "trudnych" w RNS. Dodatkowymi zaletami proponowanych układów są możliwość konstrukcji układu realizującego dowolną kombinację mnożenia i dodawania modulo, łatwość głębokiego potokowania oraz brak ograniczeń na wartość modułu.

W przypadku poszukiwania układu o najkrótszej ścieżce krytycznej dotychczasowe rozwiązania mogą być konkurencyjne dla modułów mniejszych od 512. Jednak, stosowanie dotychczasowych rozwiązań jest korzystne jedynie dla niewielkich modułów ($M_* < 128$), gdzie pozwalają one znaleźć

układ szybszy przy obszarze porównywalnym z obszarem najszybszych jednostek o nowej strukturze. Dla modułów z zakresu $(128, 512)$ szybkość dotychczasowych układów okupiona jest znacznym wzrostem zajmowanego obszaru w stosunku do jednostek zaproponowanych w rozprawie. W przypadku większych modułów struktury zaprezentowane w rozprawie są zarówno szybsze, jak i wielokrotnie mniejsze. Zaletą nowych struktur jest też możliwość znalezienia układu o nieznacznie większej długości ścieżki krytycznej od układu najszybszego, ale o wielokrotnie mniejszym obszarze.

Długość ścieżki krytycznej dla niewielkich modułów jest jednak rzadko czynnikiem decydującym. W przypadku RNS opóźnienie wprowadzane przez cały tor obliczeniowy jest zależne wyłącznie od najdłuższej ścieżki krytycznej. Z reguły układem o najdłuższej ścieżce krytycznej jest układ dla modułu o największej wartości. Zaproponowana struktura resztowych jednostek arytmetycznych pozwala na skonstruowanie zestawu jednostek o różnych parametrach AT dla tego samego modułu i wykonywanego działania. Pozwala to na wybranie wersji dopasowanej do reszty układu, co dla dotychczasowych metod konstruowania resztowych jednostek arytmetycznych było niemożliwe. W omawianym przypadku dla mniejszych modułów można zastosować układy wolniejsze, ale o mniejszym obszarze. Umożliwia to zmniejszenie kosztów całego resztowego toru obliczeniowego bez niekorzystnego wpływu na wydajność układu.

Dodatkowe korzyści można zaobserwować wskutek wykorzystania okresowości potęg 2 modulo M_* w strukturze zaproponowanej w rozprawie. Pozwala to na dodatkowe zmniejszenie opóźnień i obszaru układów najmniejszych, dzięki czemu możliwe staje się skonstruowanie jednostki wolniejszej od najszybszych o najwyżej kilka procent przy znacznie mniejszym obszarze. Niestety, właściwość tę można wykorzystać jedynie dla modułów o odpowiednio niewielkim okresie potęg 2 modulo M_*.

Konstruowanie resztowych torów obliczeniowych z użyciem HRNS pozwala na zmniejszenie zajmowanego obszaru oraz zwiększenie szybkości. Zaobserwowane zyski zależą od stosunku szerokości największego czynnika do zakresu systemu i dla zbadanych przypadków mogą przekroczyć 20–30%. Z przeprowadzonych eksperymentów wynika, że stosowanie HRNS do implementacji w FPGA jednostek arytmetycznych o strukturze opisanej w rozdziale jest uzasadnione, jeśli stosunek zakresu systemu do szerokości największego czynnika przekracza 8. Wartość ta może być użyta także jako generalna reguła konstrukcji resztowych systemów liczbowych dla FPGA.

Dla zbadanych układów FPGA, stosowanie RNS o bazie $(2^k - 1, 2^k, 2^k + 1)$ jest w większości przypadków nieopłacalne. Jedynie dla zakresu dynamicznego większego od 72 bitów można zaobserwować zmniejszenie długości ścieżki krytycznej o kilka bądź kilkanaście procent. We wszystkich

zbadanych przypadkach jednostki wykorzystujące RNS $(2^k - 1, 2^k, 2^k + 1)$ charakteryzują się obszarem zwiększonym o 30–40% w porównaniu z układami zbudowanymi z użyciem arytmetyki uzupełnieniowej.

Hierarchiczne RNS zostały użyte w zaproponowanej architekturze procesora wspomagającego obliczenia AOG. Architektura ta umożliwia implementację z użyciem arytmetyki resztowej podstawowych zadań wykorzystywanych AOG. Struktura operacji występujących w AOG umożliwia wyizolowanie działań trudnych w RNS i tym samym skonstruowanie wydajnego układu. Zastosowanie HRNS w zaprojektowanej architekturze pozwala dla zakresu dynamicznego ok. 54 bitów zwiększyć częstotliwość taktowania o ok. 20% w stosunku do układów wykorzystujących arytmetykę uzupełnieniową. Obszar układu w obu przypadkach jest podobny.

Wnioski i dalsze kierunki badań

Praca obejmuje szereg zagadnień związanych ze sprzętową akceleracją algorytmów oświetlenia globalnego w układach FPGA z użyciem arytmetyki resztowej. Opracowano nową metodę tworzenia resztowych układów arytmetycznych w matrycach FPGA. Zaproponowano i zbadano nową klasę hierarchicznych resztowych systemów liczbowych charakteryzujących się niskim kosztem konwersji pomiędzy RNS a systemem pozycyjnym. Podano nowy, szybki algorytm detekcji znaku dla RNS o bazie $(2^k - 1, 2^k, 2^k + 1)$. Zaprojektowano także architekturę procesora wspomagającego obliczenia w algorytmach oświetlenia globalnego, w którym użycie arytmetyki resztowej powoduje wzrost wydajności.

Nowa struktura resztowych jednostek arytmetycznych dla FPGA umożliwia konstruowanie układów o obszarze zależnym od kwadratu i opóźnieniach zależnych od logarytmu szerokości argumentów. Co więcej, dla przebadanego zakresu wartości modułów obszar nowych jednostek można z dużą dokładnością oszacować funkcją liniową. Zmniejszenie obszaru i opóźnienia nowych struktur jest skutkiem wyeliminowania pamięci ROM o dużej pojemności na rzecz efektywnego wykorzystania elementów obecnych w nowoczesnych układach FPGA. Dodatkową zaletą nowej struktury jest możliwość wykorzystania okresowości potęg 2 modulo, co powoduje dalsze polepszenie charakterystyk AT jednostek o najmniejszym obszarze.

Z przeprowadzonych eksperymentów wynika, że nowa struktura pozwala na zbudowanie układów najmniejszych i o najmniejszych iloczynach AT i AT^2 w całym zakresie zmienności modułu i układów najszybszych dla modułów większych od 512. Układy najszybsze dla modułów większych od 512 są także znacznie mniejsze od dotychczasowych rozwiązań. W przypadku modułów o wartościach z przedziału $(128, 512)$, jednostki o nowej strukturze są nieznacznie wolniejsze, ale zajmują wielokrotnie mniejszy obszar od rozwiązań szybszych. Zasadniczą zaletą zaproponowanej struktury jest znaczne zmniejszenie obszaru jednostek resztowych, co pozwala na użycie dużych modułów w bazie RNS. Przekłada się to na zwiększenie zakresu dynamicznego RNS oraz uproszczenie algoryt-

mów konwersji i przeprowadzania operacji trudnych.

Ważną cechą zaproponowanej struktury jest możliwość zbudowania zestawu jednostek o różnej relacji obszaru i opóźnienia dla tego samego działania i modułu. Pozwala to zmniejszyć obszar toru resztowego zawierającego zbiór jednostek dla kilku różnych wartości modułu, ponieważ dla większości z nich nie ma potrzeby użycia struktury najszybszej. Możliwe jest także dopasowanie charakterystyk AT toru resztowego do pozostałej części układu, co powoduje lepsze wykorzystanie zasobów sprzętowych.

Aby umożliwić szybki wybór jednostki o poszukiwanych charakterystykach AT, zaproponowano algorytm automatycznej generacji resztowych jednostek arytmetycznych o nowej strukturze. Pozwala on automatycznie wytypować strukturę o parametrach zbliżonych do poszukiwanym, lub zbiór układów, wśród których znajduje się jednostka najlepsza. Złożoność algorytmu jest wykładnicza, jednak dzięki zaproponowanym regułom ograniczającym liczność zbioru badanych układów możliwa jest praca interakcyjna dla modułów stosowanych w systemach DSP.

Idea zaproponowanego algorytmu polega na oszacowaniu parametrów zbioru jednostek, których struktura jest zależna od kryterium wyboru układu najlepszego. W przypadku poszukiwania rozwiązania o najmniejszym obszarze pozwala to znacznie ograniczyć liczbę badanych jednostek, co wpływa na zmniejszenie złożoności obliczeniowej algorytmu. Co więcej, na podstawie analizy struktur układów o różnych parametrach AT można stwierdzić, że w wielu przypadkach wystarczająco dobre charakterystyki AT mają jednostki, w których iloczyny częściowe są obliczane za pomocą układów mnożących. Wśród tych jednostek można często znaleźć układ o opóźnieniu tylko kilka procent większym od układu najszybszego, ale o znacznie mniejszym obszarze. Dodatkowo, z teoretycznych oszacowań złożoności nowych jednostek wynika, że stosowanie pamięci ROM na etapie obliczania iloczynów częściowych powoduje znaczny wzrost zajmowanego obszaru dla modułów większych od pewnej wartości. Problem konstruowania jednostek arytmetycznych dla dużych modułów można więc sprowadzić do problemu konstruowania generatora reszty modulo M_* dla wektora ok. $2m_*$–bitowego.

Możliwość konstrukcji resztowych jednostek arytmetycznych o niewielkim obszarze i opóźnieniu dla dużych wartości modułów pozwala na konstruowanie baz RNS z większych liczb. Na podstawie porównania parametrów jednostek resztowych z klasycznymi układami U2 można stwierdzić, że dla układów rodziny Spartan 2 wzrost szybkości przetwarzania jest możliwy, jeśli stosunek liczby bitów zakresu dynamicznego systemu do szerokości największego modułu w bazie przekracza 8. Nie dotyczy to modułów o postaciach umożliwiających użycie struktur o mniejszej złożoności, np. 2^k.

Ograniczenie maksymalnej szerokości modułu w bazie RNS wymusza wzrost liczności bazy, co komplikuje algorytmy konwersji i przeprowadzania operacji trudnych. Istnieją jednak takie bazy, dla których możliwa jest prosta konwersja przy dużej liczbie modułów. Przykładem jest zaproponowana w rozprawie nowa klasa HRNS. Zaproponowane równania konwersji odwrotnej pozwalają na skonstruowanie konwerterów o niewielkiej złożoności. Resztowe układy arytmetyczne wykorzystujące proponowany HRNS są przy tym do 20–30% mniejsze i szybsze od układów w U2 o tym samym zakresie dynamicznym. Dodatkową zaletą nowego HRNS jest możliwość adaptacji efektywnych algorytmów przeprowadzania operacji trudnych w RNS $(2^k - 1,^k , 2^k + 1)$, np. nowej, szybkiej metody detekcji znaku.

Zaprezentowany w rozprawie HRNS został zastosowany w nowej architekturze procesora wspomagającego AOG. Architektura ta pozwala na efektywną implementację kluczowych problemów obliczeniowych w AOG. Dzięki zastosowaniu arytmetyki stałoprzecinkowej jej wymagania sprzętowe są ograniczone do poziomu umożliwiającego implementację w układach FPGA. Zasadniczą zaletą nowej architektury jest możliwość zwiększenia wydajności układu poprzez zastosowanie arytmetyki resztowej. Nie wymaga to znaczących zmian w strukturze układu, ponieważ implementowane operacje są łatwo modyfikowalne do postaci pozwalającej użyć arytmetykę resztową. W prototypowym rozwiązaniu użycie HRNS spowodowało wzrost częstotliwości taktowania o ok. 20% przy podobnym obszarze układu. Stosowanie arytmetyki resztowej jest więc możliwe i celowe w układach sprzętowego wspomagania AOG.

Otwarte problemy badawcze

W rozprawie zaproponowano nową strukturę resztowych jednostek arytmetycznych dostosowaną do implementacji w układach FPGA. Struktura ta jest zbudowana z elementarnych bloków składowych układów cyfrowych, można więc bez żadnych modyfikacji użyć jej w układach ASIC. Teoretyczne oszacowania charakterystyk AT nowych struktur są na tyle dobre, że uzasadnione jest przeanalizowanie przydatności nowych struktur do implementacji wydajnych układów ASIC. Wskazane są zarówno zbadanie parametrów nowych jednostek po implementacji, jak i ewentualne modyfikacje w kierunku zwiększenia wydajności na nowej platformie.

Wzrost wydajności jednostek arytmetycznych wskutek zastosowania resztowych systemów liczbowych zależy od stosunku kosztów konwersji i operacji trudnych do zysków osiągniętych w reszto-

wym torze obliczeniowym. W rozprawie zaprezentowano nową klasę RNS, która przy niskim koszcie konwersji pozwala na konstrukcję układów arytmetycznych o większej wydajności od rozwiązań wykorzystujących system U2. Uzasadnione jest zatem poszukiwanie innych kryteriów doboru bazy RNS pozwalających na dalsze polepszanie parametrów układu. Jednym z możliwych kierunków badań może więc być analiza innych zestawów modułów pod kątem przydatności do implementacji w FPGA. Problem ten jet o tyle istotny, że zaproponowana nowa struktura resztowych jednostek arytmetycznych umożliwia użycie modułów o znacznie większych wartościach od dotychczasowych rozwiązań.

Wybór struktury resztowej jednostki arytmetycznej o parametrach dopasowanych do pozostałej części systemu jest zadaniem nietrywialnym. W rozprawie podano algorytm poszukiwania takich jednostek i reguły ich konstrukcji dla wybranych przypadków. Niestety, wykładnicza złożoność algorytmu powoduje, że dla dużych wartości modułu znalezienie rozwiązania najlepszego może wymagać długiego czasu. Wskazane jest zatem poszukiwanie innych metod przeglądania zbioru rozwiązań. Interesującym kierunkiem wydaje się zastosowanie metod probabilistycznych czy wykorzystujących np. algorytmy genetyczne.

Rozprawa zawiera szereg zagadnień związanych z implementacją sprzętową AOG z użyciem arytmetyki resztowej. Implementację ograniczono jedynie do tych fragmentów struktury procesora, w których użycie RNS wymaga wprowadzenia modyfikacji. Interesujących wyników mogłyby także dostarczyć badania kompletnego prototypu, ponieważ sam problem projektowania wydajnych procesorów wspomagających AOG jest zagadnieniem dość nowym.

Bibliografia

[AA88] S. Andraos, H. Ahmad. A new efficient memoryless residue to binary converter. *IEEE Transactions on Circuits and Systems*, 35(11):1441–1444, November 1988.

[AJ68] I. J. Akušskij, D. Judickij. *Mašinnaja aritmetika w ostatočnych klassach*. Sovetskoje Radio, Moskwa, 1968.

[AMD00] Advanced Micro Devices, Inc. *3DNow!^{TM}Technology Manual*, March 2000. 21928G/0.

[AMD06] Advanced Micro Devices, Inc. *AMD64 Architecture Programmer's Manual Volume 2:System Programming*, September 2006. 24593–Rev. 3.12.

[AM98] G. Alia, E. Martinelli. Sign detection in residue arithmetic units. *Journal of System Architecture*, 45:251–258, 1998.

[App68] A. Appel. Some techniques for shading machine renderings of solids. *AFIPS 1968 Spring Joint Computer Conference Proceedings*, volume 32, str. 37–45, 1968.

[Arv88] J. Arvo. Linear-time voxel walking for octrees. *The Ray Tracing News*, 1(5), March 1988.

[BA88] R. Bogart, Jeff Arenberg. Ray/Triangle intersection with barycentric coordinates. *The Ray Tracing News*, 1(11), November 1988.

[Bad90] D. Badouel. *Graphics gems*, chapter An efficient ray-polygon intersection, str. 390–393. Academic Press Professional, Inc., 1990.

[BEPP97] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, str. 174–182. ACM Press, 1997.

[Ber85] P. Bernardson. Fast memoryless, over 64 bits, residue–to–binary convertor. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, CAS-32(3):298–300, March 1985.

[Beu02] J.-L. Beuchat. Modular multiplication for FPGA implementation of the IDEA block cipher. Technical Report 2002-32, Laboratoire de l'Informatique du Parallélisme, September 2002.

[Beu03] J.-L. Beuchat. Some modular adders and multipliers for field programmable gate arrays. *Proceedings of the 17th International Parallel & Distributed Processing Symposium*. IEEE Computer Society, 2003.

[BGZ97] R. Bastos, M. Goslin, H. Zhang. Efficient radiosity rendering using textures and bicubic reconstruction. *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, str. 71–ff., New York, NY, USA, 1997. ACM Press.

[Bit99] J. Bittner. Hierarchical techniques for visibility determination. Technical Report DC-PSR-99-05, Department of Computer Science and Engineering, Czech Technical University in Prague, March 1999. Also available as http://www.cgg.cvut.cz/~bittner/publications/minimum.ps.gz.

[BJ78] A. Baraniecka, G. A. Jullien. On decoding techniques for residue number system realizations of digital signal processing hardware. *IEEE Transactions on Circuits and Systems*, CAS-25:935–936, November 1978.

[BM04] J.-L. Beuchat J.-M. Muller. Modulo m multiplication-addition: Algorithms and FPGA implementation. *Electronics Letters*, 40(11):654–655, May 2004.

[BPS98] M. Bhardwaj, A. B. Premkumar, T. Srikanthan. Breaking the $2n$-bit carry propagation barrier in residue to binary conversion for the $[2^n - 1, 2^n, 2^n + 1]$ modula set. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 45(9):998–1002, September 1998.

[BWAK04] S. Bi, W. Wang, A. Al-Khalili. Modulo deflation in $(2^n + 1, 2^n, 2^n - 1)$ converters. *ISCAS '04. Proceedings of the 2004 International Symposium on Circuits and Systems*, volume 2, str. 429–432, May 2004.

[CC02] A. Chalmers, K. Cater. Realistic rendering in real-time. *Euro-Par '02: Proceedings of the 8th International Euro-Par Conference on Parallel Processing*. str. 21–28, Springer-Verlag, London, UK, August 2002.

[CCS03] B. Cao, C.-H. Chang, T. Srikanthan. An efficient reverse converter for the 4-moduli set $(2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1)$ based on the New Chinese Remainder Theorem. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 50(10):1296–1303, October 2003.

[CHH02] N. A. Carr, J. D. Hall, J. C. Hart. The ray engine. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, str. 37–46. Eurographics Association, 2002.

[CRL98] G. C. Cardarilli, M. Re, R. Lojacono. RNS–to–binary conversion for efficient VLSI implementation. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 45(6):667–671, June 1998.

[CRR04] C. Cassagnabere, F. Rousselle, C. Renaud. Path tracing using the ar350 processor. *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. str. 23–29, ACM Press, New York, NY, USA, 2004.

[CW93] M. F. Cohen, J. R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993.

[CN99] R. Conway, J. Nelson. Fast converter for 3 moduli RNS using new property of CRT. *IEEE Transactions on Computers*, 48(8):852–860, August 1999.

[CHL04] G. Coombe, M. J. Harris, A. Lastra. Radiosity on graphics hardware. *GI '04: Proceedings of the 2004 conference on Graphics interface*, str. 161–168, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[DDM02] C. Damez, K. Dmitriev, K. Myszkowski. State of the art in global illumination for interactive applications and high-quality animations. *Computer Graphics Forum* vol. 22, nr 1, str. 55–78, 2003.

[Dhu98] A. Dhurkadas. Comments on "A high speed realization of a residue to binary number system converter". *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 45(3):446–447, March 1998.

[DJM98] V. S. Dimitrov, G. A. Jullien, W. C. Miller. A fast and robust RNS algorithm for evaluating signs of determinants. *Computers Math. Applic.*, 35(8):9–14, 1998.

[Eri97] J. Erickson. Plücker coordinates. *The Ray Tracing News*, 10(3), December 1997.

[FvDF+01] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, R. L. Philips. *Wprowadzenie do grafiki komputerowej*. Wydawnictwa Naukowo-Techniczne, Warszawa, 2001.

[Gib06] W. Wayt Gibbs. A great leap in graphics. *Scientific American*, August 2006.

[GJ99] A. Garcia, G. A. Julien. Comments on "An arithmetic free parallel mixed–radix conversion algorithm". *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 46(9):1259–1260, September 1999.

[GKP01] R. L. Graham, D. E. Knuth, O. Patashnik. *Matematyka konkretna*. Wydawnictwo Naukowe PWN, Warszawa, 2001.

[GPS97] D. Gallaher, F. E. Petry, P. Srivasan. The digit parallel method for fast RNS to weighted number system conv ersion for specific moduli $\left(2^k-1, 2^k, 2^k+1\right)$. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 44(1):53–57, January 1997.

[Gri01] R. P. Grimaldi. Compositions without the summand 1. *Proceedings Thirty-second Southeastern International Conference on Combinatorics, Graph Theory and Computing (Baton Rouge, LA, 2001). Congr. Numer. 152*, str. 33–43, 2001.

[GTGB84] C. M. Goral, K. E. Torrance, D. P. Greenberg, B. Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH 84*, str. 213–222, 1984.

[HA96] G. Humphreys, C. S. Ananian. Tigershark: A hardware accelerated raytracing engine. Technical report, Princeton University, 1996.

[Hal01] D. Hall. The AR350: Todays's ray trace rendering processor. *Proceedings of The Eurographics/SIGGRAPH workshop on Graphics Hardware–Hot 3D Session 1*, str. 13–19, August 2001.

[HKBZ97] V. Havran, T. Kopal, J. Bittner, J. Zara. Fast robust BSP tree traversal algorithm for ray tracing. *Journal of Graphics Tools: JGT*, 2(4):15–23, 1997.

[HLG99] A. Hubeli, L. Lippert, M. Gross. The global cube a hardware-accelerated hierarchical volume radiosity technique. CS Technical Report #331, Institute of Scientific Computing, ETH Zurich, October 1999.

[Hua83] C. H. Huang. A fully parallel mixed–radix conversion algorithm for residue number applications. *IEEE Transactions on Computers*, C-32(4):398–402, April 1983.

[Hur05] J. Hurley. Ray tracing goes mainstream. *Intel Technology Journal*, 09(02), May 2005.

[IS88] K. M. Ibrahim, S. N. Saloum. An efficient residue to binary converter design. *IEEE Transactions on Circuits and Systems*, 35(9):1156–1158, September 1988.

[Kaj86] J. T. Kajiya. The rendering equation. *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, str. 143–150. ACM Press, 1986.

[Kel97] A. Keller. Instant radiosity. *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, volume 31, str. 49–56, 1997.

[KiSK+01] H. Kobayashi, K. ichi Suzuki, Y. Kaeriyama, Y. Saida, N. Oba, T. Nakamura. 3DCGi-RAM: An intelligent memory architecture for photo-realistic image synthesis. *Proceedings of 2001 IEEE International Conference on Computer Design*, str. 462–467, September 2001.

[KiSSO02] H. Kobayashi, K. ichi Suzuki, K. Sano, N. Oba. Interactive Ray-Tracing on the 3DCGi-RAM architecture. *Proceedings of ACM/IEEE MICRO-35 4th Workshop on Media and Streaming Processors*, str. 53–59, 2002.

[Knu02] D. E. Knuth. *Sztuka programowania*, volume 2 Algorytmy seminumeryczne. Wydawnictwa Naukowo-Techniczne, Warszawa, 2002.

[Kob95] N. Koblitz. *Wykład z teorii liczb i kryptografii*. Wydawnictwa Naukowo-Techniczne, Warszawa, 1995.

[Kre97] K. Kreeger. Parallel architectures for real-time volume rendering. citeseer.ist.psu.edu/article/kreeger97parallel.html, March 1997.

[Lan04] B. R. Landon. Using graphics hardware to accelerate progressive refinement radiosity. Technical report, Brown University, Providence, RI, Feb 2004.

[Łub03] *Synteza układów cyfrowych.* Praca zbiorowa pod redakcją prof. Tadeusza Łuby. Wydawnictwa Komunikacji i Łączności, Warszawa, 2003.

[Mat68] Mathematical Application Group, Inc. 3-D simulated graphics offered by service bureau. *Datamation*, 13(1), February 1968.

[MB01] U. Meyer-Base. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer-Verlag, Berlin Heidelberg, 2001.

[MBGT01] U. Meyer-Bäse, A. García, F. Taylor. Implementation of a communications channelizer using FPGAs and RNS arithmetic. *The Journal of VLSI Signal Processing*, 28(1–2):115–128, 2001.

[MFAA98] Luiz C. B. Maltar, Felipe M. G. França, Vladimir C. Alves, Cláudio L. Amorin. Implementation of RNS addition and RNS multiplication into FPGAs. *IEEE Symposium on FPGAs for Custom Computing Machines*, str. 331–332, 1998.

[MM98] D. F. Miller, W. S. McCornick. An arithmetic free parallel mixed–radix conversion algorithm. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 45(1):158–162, January 1998.

[Moh98] P. V. Ananda Mohan. Evaluation of fast conversion techniques for binary–residue numbers systems. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 45(10):1107–1109, October 1998.

[Moh01] P. V. Ananda Mohan. Comments on "Breaking the $2n$–bit carry propagation barrier in residue to binary conversion for the $[2^n - 1, 2^n, 2^n + 1]$ modula set". *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 48(8):1031, August 2001.

[MT97] T. Möller, B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

[NC02] K. H. Nielsen, N. J. Christensen. Fast texture-based form factor calculations for radiosity using graphics hardware. *J. Graph. Tools*, 6(4):1–12, 2002.

[NN85] T. Nishita, E. Nakamae. Continuous tone representation of three–dimensional objects taking account of shadows and interreflection. *SIGGRAPH 85*, str. 23–30, 1985.

[OLG+05] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Eurographics 2005, State of the Art Reports*, str. 21–51, Dublin, Ireland, August 29–September 2, 2005.

[PBMH02] T. J. Purcell, I. Buck, W. R. Mark, P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[Pie94] S. J. Piestrak. Design of residue generators and multioperand modular adders using carry-save adders. *IEEE Transactions on Computers*, 423(1):68–77, January 1994.

[Pie95] S. J. Piestrak. A high-speed realization of a residue to binary number system converter. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 42(10):661–663, October 1995.

[PKS01] V. Paliouras, K. Karagianni, T. Stouraitis. A low-complexity combinatorial RNS multiplier. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 48(7):675–683, July 2001.

[PMS+99] S. Parker, W. Martin, P.-P. Sloan, P. Shirley, B. Smits, C. Hansen. Interactive ray tracing. *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, str. 119–126, ACM Press, New York, NY, USA, 1999.

[PPL+99] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Computer Graphics and Visualization*, (3), str. 238–250, May 1999.

[Pre92] A. B. Premkumar. An RNS to binary converter in $2n+1, 2n, 2n-1$ moduli set. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 39(7):480–482, July 1992.

[Pre95] A. B. Premkumar. An RNS to binary converter in a three moduli set with common factors. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 42(4):298–301, April 1995.

[Pre04] A. B. Premkumar. Corrections to "An RNS to binary converter in a three moduli set with common factors". *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, 51(1):43, January 2004.

[Pur04] T. J. Purcell. *Ray Tracing On A Stream Processor*. PhD thesis, Stanford University, Department of Computer Science, March 2004.

[RPK00] S. K. Raman, V. Pentkovski, J. Keshava, Implementing Streaming SIMD Extensions on the Pentium III Processor. *IEEE Micro:*, 20(4):47–57, 2000.

[RGMBL02] J. Ramirez, A. García, U. Meyer-Bäse, A. Lloris. Fast RNS FPL–based communication receiver design and implementation. *Proc. 12th Int. Conf. Field Programmable Logic and Applications FPL'2002*, volume 2438, str. 427–481. Lecture Notes in Computer Science, September 2002.

[RGTL03] J. Ramirez, A. García, F. Taylor, A. Lloris. Implementation of RNS–based distributed arithmetic discrete wavelet transform architectures using Field–Programmable Logic. *J. VLSI Signal Processing*, 33(1–2):171–190, February 2003.

[Saa07] SaarCOR - a hardware architecture for real time ray tracing. *http://www.saarcor.de/*, 2007.

[SA99] A. Skavantzos, M. Abdallah. Implementation issues of the two-level Residue Number System with pairs of conjugate moduli. *IEEE Transactions on Signal Processing*, 47(3), str. 826–838, March 1999.

[Sch06] J. Schmittler. *SaarCOR A Hardware–Architecture for Realtime Ray Tracing*. PhD thesis, Saarland University, Saarbrücken, Germany, January 2006.

[SF01] R. J. Segura, F. R. Feito. Algorithms to test ray-triangle intersection. Comparative study. *WSCG'2001 - the 9-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2001*, str. 76–81, February 2001.

[SH81] R. Siegel, J. R. Howell. *Thermal Radiation Heat Transfer*. McGraw Hill, New York, 1981.

[SJJT86] M. A. Sorderstrand, W. Kenneth Jenkins, G. A. Jullien, F. J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, NY, 1986.

[SL02] H. Styles, W. Luk. Accelerating radiosity calculations using reconfigurable platforms. *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, str. 279–281, IEEE Computer Society Press, 2002.

[ST67] N. S. Szabo, R. I. Tanaka. *Residue arithmetic and its application to computer technology*. Mc Graw-Hill, New York, 1967.

[Sto06] J. H. Stokes. Ray tracing soon to go real–time for 3D rendering. *http://arstechnica.com/news.ars/post/20060805-7430.html*, May 2006.

[SWS02] J. Schmittler, I. Wald, P. Slusallek. Saarcor: a hardware architecture for ray tracing. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, str. 27–36. Eurographics Association, 2002.

[TL01] T. Todman, W. Luk. Reconfigurable designs for ray tracing. *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001.

[Tom05a] T. Tomczak. Fast sign detection for RNS $(2^n - 1, 2^n, 2^n + 1)$. preprinty nr 32/2005, Instytut Informatyki, Automatyki i Robotyki Politechniki Wrocławskiej, listopad 2005. Wstępnie zaakceptowany w IEEE Transactions on Circuits and Systems I.

[Tom05b] T. Tomczak. Hierarchiczne resztowe systemy liczbowe w strukturach FPGA. *Inżynieria komputerowa. Praca zbiorowa pod red. Wojciecha Zamojskiego*, str. 405–414. Wydawnictwa Komunikacji i Łączności, 2005.

[Tom05c] T. Tomczak. Metody i układy sprzętowego wspomagania obliczeń w algorytmach oświetlenia globalnego. *Reprogramowalne układy cyfrowe. RUC '2005. Materiały VIII krajowej konferencji naukowej*, str. 139–148. Szczecin : Instytut Informatyki PSzczec., 2005.

[Tom06a] T. Tomczak. Residue arithmetic in FPGA matrices. *Proceedings of International Conference on Dependability of Computer Systems. DepCoS - RELCOMEX 2006, Szklarska Poręba, 25-27 May 2006*, str. 297–304. IEEE Computer Society [Press], 2006.

[Tom06b] T. Tomczak. Układy arytmetyki resztowej w matrycach FPGA. *Pomiary, Automatyka, Kontrola*, (nr 7bis wyd. spec. dod.):92–94, 2006. referat z IX konferencji nt. Reprogramowalne układy cyfrowe RUC ’06. Szczecin, 18-19.05.2006.

[Ulm83] Z. D. Ulman. Sign detection and implicit-explicit conversion of numbers in residue arithmetic. *IEEE Transactions on Computers*, C-32(6):590–594, June 1983.

[Ven96] R. Venkatesan. FPGA implementation of RNS structures. PhD thesis, University of Windsor, Windsor, Ontario, December 1996.

[VR94] B. Vinnakota, V. V. Bapeswara Rao. Fast conversion techniques for binary–residue numbers systems. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 41(12):927–929, December 1994.

[Vu85] T. V. Vu. Efficient implementations of the Chinese Remainder Theorem for sign detection and residue decoding. *IEEE Transactions on Computers*, C-34(7):646–651, July 1985.

[Wan00] Y. Wang. Residue-to-binary converters based on New Chinese Remainder Theorems. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 47(3):197–205, March 2000.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[WJM00] Z. Wang, G. A. Jullien, W. C. Miller. An improved residue-to-binary converter. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 47(9):1437–1440, September 2000.

[WS01] I. Wald, P. Slusallek. State-of-the-art in interactive raytracing. *EUROGRAPHICS 2001*, State of the Art Reports, str. 21–42, 2001.

[WSA99] Y. Wang, M. N. S. Swamy, M. Omar Ahmad. Residue–to–binary number converters for three moduli sets. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 46(2):180–183, February 1999.

[WSAS02] Y. Wang, X. Song, M. Aboulhamid, Hong Shen. Adder based residue to binary number converters for $(2^n-1, 2^n, 2^n+1)$. *IEEE Transactions on Signal Processing*, 5(7):1772–1779, July 2002.

[WSAW00] W. Wang, M. N. S. Swamy, M. O. Ahmad, Y. Wang. A high-speed residue-to-binary converter for three-moduli $\left(2^k, 2^k-1, 2^{k-1}-1\right)$ RNS and a scheme for its VLSI implementation. *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, 47(12):1576–1581, December 2000.

[WSAW03] W. Wang, M. N. S. Swamy, M. Omair Ahmad, Yuke Wang. A study of the residue–to–binary converters for the three–moduli sets. *IEEE Transactions on Circuits and Systems—Part I: Fundamental Theory and Applications*, 50(2):235–243, February 2003.

[WSB01] I. Wald, P. Slusallek, C. Benthin, M. Wagner. Interactive distributed ray tracing of highly complex models. *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. str. 277–288, Springer-Verlag, London, June 2001.

[WSBW01] I. Wald, P. Slusallek, C. Benthin, M. Wagner. Interactive rendering with coherent ray tracing. *EG 2001 Proceedings*, volume 20(3), str. 153–164. Blackwell Publishing, 2001.

[Xil03a] Xilinx Inc. *Spartan-II 2.5V FPGA Family: Complete Data Sheet*, September 2003.

[Xil03b] Xilinx Inc. *Virtex^{TM}-II Platform FPGAs: Complete Data Sheet*, October 2003.

[Xil04] Xilinx Inc. *Spartan-3 FPGA Family: Complete Data Sheet*, February 2004.

[Xil05] Xilinx Inc. *Virtex-4 Family Overview: Preliminary Product Specification*, June 2005. DS112 (v1.4).

[Yas92] H. M. Yassine. Hierarchical residue numbering system suitable for VLSI arithmetic architectures. *ISCAS 1992*, str. 811–814. IEEE, 1992.

[Zim99] R. Zimmermann. Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia)*, str. 158–167, Los Alamitos, CA, 1999. IEEE Computer Society Press.