POLITECHNIKA GDAŃSKA
Wydział Elektroniki, Telekomunikacji i Informatyki

Marek Strachacki

Projektowanie i optymalizacja sprzętowo-programowych wbudowanych systemów przetwarzania danych

Rozprawa doktorska

Promotor:
dr hab. inż. Stanisław Szczepański, prof. nadzw. PG
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska

Gdańsk, 8 lutego 2012

Podziękowanie

Składam serdeczne podziękowanie panu profesorowi Stanisławowi Szczepańskiemu za jego zaangażowanie, nieustające wsparcie i nieocenioną pomoc udzieloną podczas pisania niniejszej rozprawy doktorskiej. Wiele godzin przeprowadzonych dyskusji zaowocowało lepszym ujęciem tematu, rozszerzeniem badań i osiągnięciem lepszej spójności tez przedstawionych w pracy.

Niniejszą pracę dedykuję mojej żonie Agnieszce, synowi Bartkowi i córce Agacie, którzy wykazali dużo cierpliwości podczas niezliczonych godzin poświęconych na pracę naukową. Dziękuję również moim rodzicom za motywację i wiarę w powodzenie całego przedsięwzięcia.

Spis Treści

Spis Algorytmów

Spis Definicji

Spis Rysunków

Spis Tabel

Skróty i oznaczenia

ABS	(ang. *Anti-Lock Braking System*) – układ stosowany w pojazdach mechanicznych w celu zapobiegania blokowaniu się kół podczas hamowania;
AC3	(ang. *Audio Compression-3*) – standard stratnego kodeka dźwięku używany w większości płyt DVD-Video;
ACE	(ang. *Adaptive Cryptographic Engine*) – adaptacyjny koprocesor kryptograficzny [69];
AES	(ang. *Advanced Encryption Standard*) – nowy algorytm szyfrowania zastępujący DES;
ALAP	(ang. *As Last As Possible*) – algorytm kolejkowania zadań, wybierający zadania o najpóźniejszym terminie zakończenia;
AMBA	(ang. *Advanced Microcontroller Bus Architecture*) – architektura magistrali zaproponowana przez ARM [12];
AoC	(ang. *Algorithm on Chip*) – algorytm zaimplementowany w jednym układzie scalonym [238];
API	(ang. *Application Programming Interface*) – interfejs programistyczny;
ASB	(ang. *Advanced System Bus*) – magistrala AMBA dla procesorów [12];
ASIC	(ang. *Application Specific Integrated Circuit*) – typ elektronicznych układów scalonych zaprojektowanych do realizacji z góry ściśle określonego zadania;
ASIP	(ang. *Application Specific Integrated Processor*) – procesor specyficzny dla danego zastosowania;
ATM	(ang. *Asynchronous Transfer Mode*) – szerokopasmowa technologia komunikacyjna, dzięki której jest możliwe przesyłanie danych interakcyjnych;
B&B	(ang. *Branch and Bound*) – metoda optymalizacji podziału i ograniczeń;
BBN	(ang. *Bayes Belief Network*) – sieci Bayesa;
BeFS	(ang. *Best First Search*) – szukanie ku najlepszemu;
BFC	(ang. *Bin Fraction Curve*) – krzywa częściowa dla zbioru implementacji [139];
BFM	(ang. *Bus Functional Model*) – model z interfejsem do magistrali
BFS	(ang. *Breadth First Search*) – szukanie wszerz
BSE	(ang. *Bus and Synchronization Event*) – zdarzenia na magistrali i synchronizacyjne;
CAD	(ang. *Computer Aided Design*) – komputerowe wspomaganie projektowania;
CAP	(ang. *Crypto Asynchronous Processor*) – moduł wykonujący instrukcje asynchronicznie do CPU [79];
CBC	(ang. *Cipher Block Chaining*) – tryb szyfrowania wiązania bloków zaszyfrowanych;
CDFG	(ang. *Control/Data Flow Graph*) – graf przepływu danych i sterowania;
CFB	(ang. *Cipher FeedBack*) – tryb szyfrowania ze sprzężeniem zwrotnym;
CFSM	(ang. *Codesign Finite State Machine*) – skończona maszyna stanów dla Codesign;
CG	(ang. *Cluster Growth*) – metoda rosnącego klastrowania;
CISC	(ang. *Complex Instruction Set Computer*) – komputer o złożonej liście instrukcji;
CLB	(ang. *Configurable Logic Block*) – konfigurowalny blok logiczny w układach FPGA Xilinx [277];
CMOS	(ang. *Complementary Metal Oxide Semiconductor*) – technologia wytwarzania układów scalonych za pomocą komplementarnych tranzystorów MOS;
Codex-dp	(ang. *Codesign of Communication Systems Using Dynamic Programming*) – współbieżne projektowanie systemów komunikacyjnych z użyciem programowania dynamicznego [51];
CPI	(ang. *Coprocessor Interface*) – interfejs koprocesora [265];
CPU	(ang. *Central Processor Unit*) – procesor główny;
CUDA	(ang. *Compute Unified Device Architecture*) - uniwersalna architektura procesorów wielordzeniowych (głównie kart graficznych);
DAC	(ang. *Direct Attached Crypto*) – moduł wykonujący instrukcje synchronicznie do CPU [79];
DAG	(ang. *Directed Acyclic Graph*) – acykliczny graf skierowany;
DES	(ang. *Data Encryption Standard*) – pierwszy standard szyfrowania danych;
DFG	(ang. *Data Flow Graph*) - graf przepływu danych;
DFS	(ang. *Depth First Search*) – szukanie wgłąb;
DP	(ang. *Dynamic Programming*) – programowanie dynamiczne;
DPA	(ang. *Differential Power Analysis*) – różnicowa analiza mocy [150];

DSP	(ang. *Digital Signal Processing*) – cyfrowe przetwarzanie sygnałów;
DVD	(ang. *Digital Versatile Disc*) – standard zapisu danych na optycznym nośniku danych o większej pojemności niż CD-ROM uzyskanej dzięki zwiększeniu gęstości zapisu;
EAB	(ang. *Embedded Array Block*) – blok pamięci wbudowanej w układach Altera [4];
ECC	(ang. *Elliptic Curve Cryptosystem\Cryptography*) – system kryptograficzny oparty o krzywe eliptyczne;
EDIF	(ang. *Electronic Design Interchange Format*) – format przechowywania schematów i list połączeń w projektach elektronicznych;
EPC	(ang. *Energy Per Cycle*) – metryka energii [145];
EPI	(ang. *Energy Per Instruction*) – metryka energii [268];
FIPS	(ang. *Federal Information Processing Standard*) – federalny standard przetwarzania informacji ogłaszany przez NIST;
FL	(ang. *Fuzzy Logic*) – logika rozmyta [281];
FM	(ang. *Fiduccia-Mattheyses*) – heurystyka Fiduccia-Mattheysesa [94];
FPE	(ang. *Fuzzy Performance Estimator*) – rozmyty estymator wydajności [49];
FPGA	(ang. *Field Programmable Gate Array*) – reprogramowalny układ logiczny o strukturze macierzowej;
FPSLIC	(ang. *Field Programmable System Level Integration Circuit*) – układ SoC produkcji Atmel zawierający 8-bitowy procesor AVR i FPGA;
GA	(ang. *Genetic Algorithms*) – algorytmy genetyczne;
GC	(ang. *Global Criticality*) – metryka globalnej krytyczności w systemie Ptolemy;
GCLP	(ang. *Global Criticality Local Phase*) – heurystyka stosowana w systemie Ptolemy;
GDL	(ang. *Graph Description Language*) – język opisu grafów [143];
GF	(ang. *Galois Field*) – ciało Galois;
GML	(ang. *Graph Modelling Language*) – język modelowania grafów [117];
GraphML	(ang. *Graph Markup Language*) – język znaczników grafów [44];
GSM	(ang. *Global System for Mobile Communication*) – najpopularniejszy obecnie standard telefonii komórkowej;
HDL	(ang. *Hardware Description Language*) – język opisu sprzętu;
HDTV	(ang. *High Definition TV*) – telewizja wysokiej rozdzielczości (1080i);
HECC	(ang. *Hyperelliptic Curve Cryptosystem\Cryptography*) – system kryptograficzny oparty o krzywe hypereliptyczne;
HES	(ang. *Hardware Embedded Simulation*) – karta FPGA akceleratora symulacji HDL;
HiPART	(ang. *Hierarchical Partitioning*) – podział hierarchiczny [119];
HW	(ang. *Hardware*) – sprzęt (w niniejszej pracy nie dotyczy to CPU oraz pamięci);
IBS	(ang. *Implementation Bin Selection*) – wybór ze zbioru implementacji [139];
IBM	(ang. *International Business Machine*);
ICSF	(ang. *Integrated Cryptographic Service Facility*) [79];
IDEA	(ang. *International Data Encryption Algorithm*) – algorytm szyfrowania danych;
IEEE	(ang. *Institute of Electrical and Electronics Engineers*) – instytut inżynierów elektryków i elektroników;
ILP	(ang. *Instruction Level Parallelism*) – równoległość na poziomie instrukcji [166];
ILP	(ang. *Integer Linear Programming*) – całkowitoliczbowe programowanie liniowe;
IO	(ang. *Input-Output*) – system, układy lub porty wejścia-wyjścia;
IP	(ang. *Internet Protocol*) – protokół internetowy;
IPA	(ang. *Inferential Power Analysis*) – wnioskowana analiza mocy;
IPCore	(ang. *Intellectual Property Core*) – moduł lub komórka możliwa do powtórnego użycia w projektach układów scalonych;
IPSec	(ang. *Internet Protocol Security*) – zbiór protokołów do zabezpieczania komunikacji IP przez autentykację i szyfrowanie każdego pakietu;
ISDN	(ang. *Integrated Services Digital Network*) – sieć cyfrowa z integracją usług;
ISO	(ang. *International Organization for Standarization*) – Międzynarodowa Organizacja Standaryzacyjna;
ISS	(ang. *Instruction Set Simulator*) – symulator zbioru instrukcji procesora;
ITPG	(ang. *In Tree Precedence Graph*) – graf zależności zadań;
JPEG	(ang. *Joint Photographic Experts Group*) – format kompresji statycznych obrazów rastrowych, przeznaczony głównie do stratnego zapisu obrazów naturalnych;

JTAG	(ang. *Joint Test Action Group*) – standard IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture używany głównie do testowania płyt drukowanych i układów scalonych oraz jako interfejs debugowy;
KL	(ang. *Kernighan-Lin heuristics*) – heurystyka Kernighana-Lina [141];
LB	(ang. *Lower Bound*) – dolne ograniczenie
LC	(ang. *Logic Cell*) – komórka logiczna w układach FPGA Altera [4];
LE	(ang. *Logic Element*) – element logiczny w układach FPGA Altera [4];
LoC	(ang. *Line of Code*) – linia kodu źródłowego;
LP	(ang. *Linear Programming*) – programowanie liniowe;
LP	(ang. *Local Phase*) – faza lokalna heurystyki GCLP w systemie Ptolemy;
LU	(ang. *Loop Unrolling*) – architektura szyfrów iteracyjnych z rozwinięciem pętli;
LUT	(ang. *Look Up Table*) – tablica asocjacyjna realizowana jako pamięć ROM w układach FPGA;
LYCOS	(ang. *LYngby CO-Synthesis System*) – system współbieżnej syntezy Lyngby [147];
Mbps	(ang. *Megabit per second*) – 10^6 bitów na sekundę;
MDS	(ang. *Maximum Distance Separable*) – macierz maksymalnych odległości [224];
MIBS	(ang. *Implementation Bin Selection*) – wybór ze zbioru implementacji [139];
MILP	(ang. *Mixed Integer Linear Programming*) – programowanie liniowe całkowitoliczbowe mieszane;
MLFM	(ang. *Multi Level Fiduccia Mattheyses*) – wielopoziomowa heurystyka Fiduccia-Mattheyses [200];
MMU	(ang. *Memory Management Unit*) – moduł zarządzania pamięcią procesora ARM;
MMX	(ang. *MultiMedia eXtension*) – rozszerzenie zbioru instrukcji o 57 instrukcji SIMD dla procesora Pentium™;
MPEG	(ang. *Moving Picture Experts Group*) – zatwierdzona przez ISO grupa powszechnie stosowanych formatów zapisu danych zawierających obraz i dźwięk;
NIST	(ang. *National Institute of Standards and Technology*) – Narodowy Instytut Standaryzacji i Technologii;
NP	(ang. *Nondeterministic Polynomial*) – niedeterministycznie wielomianowa złożoność obliczeniowa;
OFB	(ang. *Output FeedBack*) – tryb szyfrowania z wyjściowym sprzężeniem zwrotnym;
OPB	(ang. *On-chip Peripheral Bus*) – magistrala układów peryferyjnych CoreConnect wewnątrz układu SoC w MicroBlaze;
PACE	(ang. *Partitioning Algorithm with Communication Emphasis*) – algorytm podziału uwzględniający komunikację [147];
PE	(ang. *Processing Element*) – element przetwarzający;
PHT	(ang. *Pseudo Hadamard Transform*) – transformacja pseudo-Hadamarda [224];
PKSC	(ang. *Public Key Security Control*) [79];
PLI	(ang. *Programming Language Interface*) – interfejs dołączania programów napisanych w języku C do kodu Verilog;
PWM	(ang. *Pulse Width Modulator*) – modulator szerokości impulsu;
RAM	(ang. *Random Access Memory*) – pamięć o dostępie swobodnym;
RDI	(ang. *Remote Debug Interface*) – interfejs zdalnego debugowania [11];
RISC	(ang. *Reduced Instruction Set Computer*) – komputer o zredukowanej liście instrukcji;
ROM	(ang. *Read Only Memory*) – pamięć tylko do odczytu;
RSA	(ang. *Rivest, Shamir, Adleman*) – algorytm asymetryczny oparty na trudności faktoryzacji dużych liczb;
RTL	(ang. *Register Transfer Level*) – poziom abstrakcji opisu sprzętu na poziomie transferu między rejestrowego;
RTOS	(ang. *Real Time Operating System*) – komputerowy system operacyjny, opracowany tak, aby spełnić wymagania narzucone na czas wykonywania zadanych operacji;
SA	(ang. *Simulated Annealing*) – symulowane wyżarzanie;
SBox	(ang. *Substitution Box*) – nieliniowe podstawienie tablicowe;
SDF	(ang. *Standard Delay Format*) – format definiowania opóźnień dla języków opisu sprzętu HDL;
SDF	(ang. *Synchronous Data Flow*) – model synchronicznego przepływu danych;
SHAPES	(ang *Software-Hardware Partitioning Expert System*) – system ekspertowy do podziału HW/SW [168];

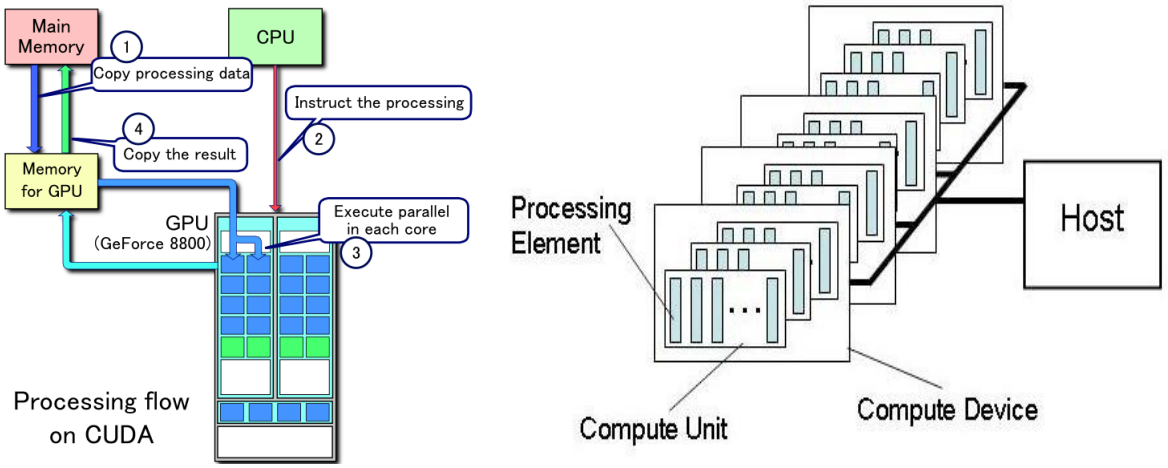
SIMD	(ang. *Single Instruction Multiple Data*) – architektura przetwarzania wielu strumieni danych w oparciu o pojedynczy program;
SNR	(ang. *Signal to Noise Ratio*) – stosunek sygnału do szumu;
SoC	(ang. *System on Chip*) – układ scalony zawierający kompletny system elektroniczny (układy cyfrowe, analogowe i cyfrowo-analogowe);
SPA	(ang. *Simple Power Analysis*) – prosta analiza mocy [150];
SPACE	(ang. *Simplified PACE*) – uproszczony algorytm PACE [136];
SRAM	(ang. *Static Random Access Memory*) – pamięć statyczna o dostępie swobodnym;
SSH	(ang. *Secure Shell*) – standard protokołów komunikacyjnych używanych w sieciach komputerowych;
SW	(ang. *Software*) – oprogramowanie;
TAS	(ang. *Timing Accurate Simulator*) – symulator modelujący opóźnienia czasowe;
TCPC	(ang. *Toggle Count Per Clock*) – metryka energii [215];
TG	(ang. *Task Graph*) – graf zadań [20];
TS	(ang. *Tabu Search*) – metoda optymalizacji poszukiwania tabu;
UART	(ang. *Universal Asynchronous Receiver Transmitter*) – uniwersalny układ nadajnika i odbiornika asynchronicznej transmisji szeregowej;
UB	(ang. *Upper Bound*) – funkcja ograniczenia górnego;
VCD	(ang. *Value Change Dump*) – tekstowy format zrzutu przebiegów sygnałów generowany przez symulatory HDL;
VHDL	(ang. *Very High Speed Integrated Circuits Hardware Description Language*) – język opisu sprzętu używany w komputerowym projektowaniu układów cyfrowych i mieszanych;
VHPI	(ang. *VHDL Programming Interface*) – interfejs dołączania programów do kodu VHDL;
VLIW	(ang. *Very Long Instruction Word*) – architektura procesora z bardzo długimi rozkazami (z reguły dłuższymi niż szerokość magistrali), wykorzystując równoległość na poziomie instrukcji ILP;
VLSI	(ang. *Very Large Scale Integration*) – układ scalony o dużej skali integracji;
WDDL	(ang. *Wave Dynamic Differential Logic*) – styl projektowania FPGA wyrównujący pobór mocy [258];
WSN	(ang. *Wireless Sensor Network*) – bezprzewodowa sieć sensorowa [35];
XML	(ang. *eXtensive Markup Language*) – język znaczników;
XOR	(ang. *Exclusive OR*) – logiczna suma wyłączająca.

1. Wstęp

Systemy wbudowane (ang. *embedded systems*) są częścią większego systemu, dedykowane dla danego zastosowania, realizujące stałą funkcjonalność w ściśle określonym środowisku, instalowane jednokrotnie, rzadko przeprogramowane i pracujące niezależnie [18][38][73][112][192]. Zastosowanie systemów wbudowanych rozciąga się od sieci sensorowych (np. WiseNET [87]), elektroniki samochodowej (np. kontrola ABS), automatyki i robotyki (np. kontrola linii przemysłowej), kompresji i kodowania dźwięku (np. dolby AC3, MPEG2), obrazu (np. MPEG2, JPEG, MPEG4, DVD, HDTV), mobilnej komunikacji bezprzewodowej (np. GSM), telekomunikacji (np. przełączniki ATM, Gigabit Ethernet, urządzenia ISDN), do zastosowań medycznych, lotniczych i wojskowych [20][39][192]. Wymienione dziedziny zastosowań wymagają przetwarzania i przesyłania danych w czasie rzeczywistym [192]. Najczęstszymi wymaganiami stawianymi systemom wbudowanym są: niska cena, niski pobór mocy, krótki czas wytwarzania, duża wydajność i przepustowość, duża niezawodność.

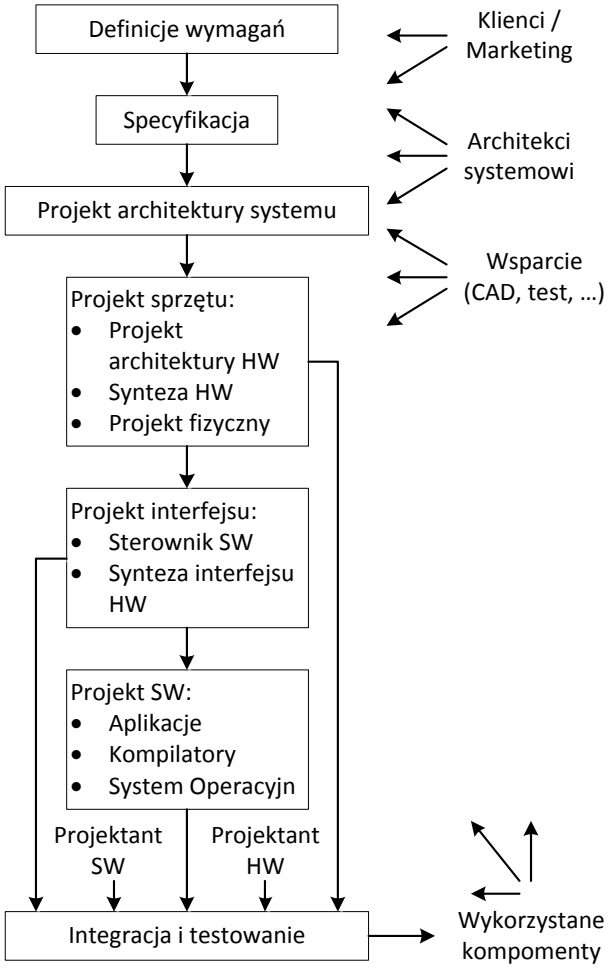
1.1. Wprowadzenie do współbieżnego projektowania

Coraz wyższe wymagania aplikacji w przypadku architektur homogenicznych prowadzą do konieczności podjęcia kompromisowych decyzji projektowych [19]. Złożoność systemów wbudowanych, obniżanie kosztów projektowania, łatwość śledzenia i testowania oraz skrócenie czasu dostarczenia produktu na rynek powoduje konieczność użycia procesora, na którym wykonywane jest oprogramowanie [192]. Realizacje programowe mimo niskiego kosztu projektowania, lepszych możliwości debugowania i testowania nie zapewniają osiągnięcia wymaganej wydajności. Implementacje sprzętowe mimo dużej wydajności są zbyt drogie, za mało elastyczne i zbyt trudno wprowadzać w nich zmiany. Z tego powodu pojawiła się tendencja do wykorzystania architektur heterogenicznych, gdzie oprócz procesora stosowane są moduły sprzętowe ASIC lub FPGA, które przyspieszają wykonanie zadań krytycznych czasowo [192]. Implementacje heterogeniczne wykorzystując synergię sprzętu i oprogramowania uzyskują większą wydajność, niższy pobór mocy [241] i niższy koszt niż dla implementacji homogenicznych [16][175][234], gdyż zwiększa się przestrzeń projektowa i dostępne spektrum implementacji [16][22]. Rysunek 1 przedstawia przykład heterogenicznych systemów komputerowych do wspomagania obliczeń za pomocą wydajnych kart graficznych: DirectCompute API w standardzie DirectX11 firmy Microsoft [185], architektura CUDA firmy nVidia [185], środowisko OpenCL firmy Apple grupy Khronos [9].

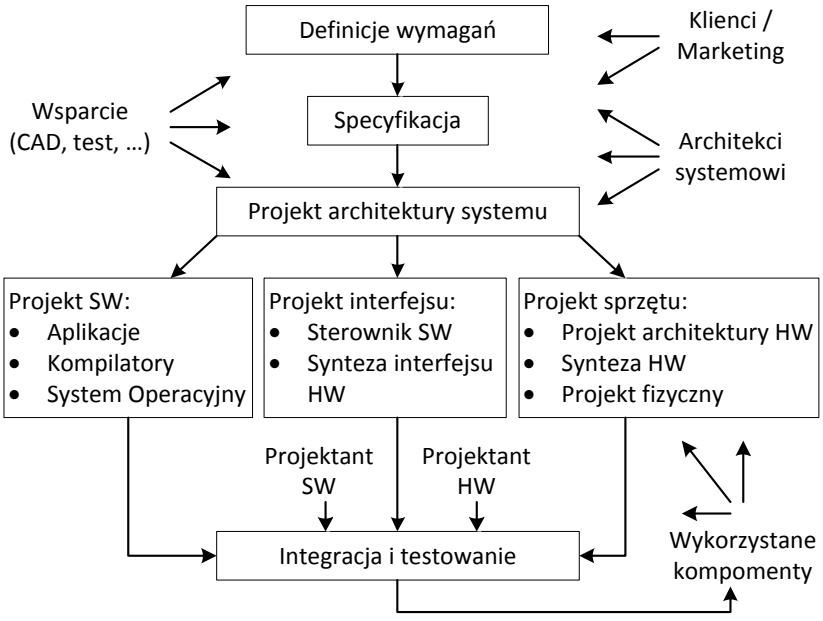


Rysunek 1. Architektura CUDA [185] oraz model platformy w środowisku OpenCL [9]

Aby zapewnić spełnienie wszystkich wymagań systemowych w coraz krótszym czasie, tradycyjny niezależny proces projektowania, który przedstawia Rysunek 2 jest zastępowany przez współbieżny proces projektowania sprzętu i oprogramowania (ang. *Hardware/Software Codesign*), co obrazuje Rysunek 3.



Rysunek 2. Schemat niezależnego procesu projektowania



Rysunek 3. Proces współbieżnego projektowania sprzętu i oprogramowania [89]

Definicja 1. Współbieżny proces projektowania Codesign [58][111][175]

Współbieżny proces projektowania Codesign jest metodologią projektowania systemu specjalnego przeznaczenia, wspierającą wspólny i współbieżny proces rozwoju sprzętu i oprogramowania na każdym jego etapie (wspólna specyfikacja, realizacja i weryfikacja), w celu wykorzystania elastyczności oraz wydajnej alokacji funkcji i osiągnięcia dzielonej funkcjonalności oraz celów wydajnościowych dla łączonego systemu.

Jak określa Definicja 1, proces współbieżnego projektowania rozważa kompromisy pomiędzy alternatywnymi rozwiązaniami oraz interakcje pomiędzy komponentami różnych typów [75]. Proces umożliwia wczesną weryfikację, czy proponowana implementacja spełni założone wymagania (np. czasowe) i nie przekroczy ograniczeń projektowych (np. zużycie zasobów sprzętowych). Jednocześnie rozwiązuje problemy ograniczenia zasobów i podziału funkcjonalności pomiędzy sprzęt i oprogramowanie, co czyni proces projektowania bardziej przewidywalnym [19].

Etapy procesu współbieżnego projektowania przedstawia Rysunek 3 [89][192]:

- Specyfikacja systemu – pozyskiwanie od klienta wymagań funkcjonalnych i poza funkcjonalnych w języku naturalnym.
- Modelowanie systemu – wykonywane jest przy użyciu języka formalnego na poziomie systemowym, zapewnia niezależność od architektury, podziału i implementacji [16][19][58], umożliwia ponowne wykorzystanie komponentów z innych projektów:
  - Homogeniczne – przy użyciu jednej notacji (np. CFSM [58], modeli przepływu sterowania, modeli zorientowanych na zadania [20], modeli przepływu danych) lub języków niższego poziomu (np. HardwareC, SystemC),
  - Heterogeniczne – przy użyciu kilku notacji lub dziedzin przy wsparciu narzędzi (np. Ptolemy [192]), co nierzadko narzuca inicjalny podział funkcjonalności pomiędzy sprzęt i oprogramowanie [73].
- Weryfikacja modelu – symulacja modelu wyspecyfikowanego w języku modelowania, może być przeprowadzona jako weryfikacja formalna.
- Eksploracja architektury – wybór komponentów oraz sposobu ich połączeń i organizacji [19][112] lub wybór architektury z dostępnych rozwiązań.
- Estymacja parametrów – używane są metryki statyczne i dynamiczne do określania kosztów i parametrów implementacji; dla problemu rozszerzonego podziału generowane są krzywe kompromisu (ang. *tradeoff curve*) dla każdego modułu.
- Podział modelu [89]:
  - Podział (ang. *partitioning*) - podział funkcjonalności na elementy lub procesy wykonywany na poziomie modelowania aplikacji (ang. *what*).
  - Alokacja (ang. *allocation*) - przydział zadania lub procesu do jednostki obliczeniowej lub elementu przetwarzającego PE, określenie elementu wykonującego obliczenia (ang. *who*). Alokacja determinuje ilość elementów przetwarzających na platformach bez określania ich typów.
  - Odwzorowanie (ang. *mapping, binding*) - wybór ze zbioru możliwych implementacji, wybór komponentów dla elementu, przypisanie elementu przetwarzającego do poszczególnych typów komponentów (ang. *how*). Odwzorowanie podejmuje szczegółowe decyzje o typach elementów przetwarzających i wymaga konkretnego modelu architektury.

  - Kolejkowanie / szeregowanie (ang. *scheduling*) – wyznaczenie czasu rozpoczęcia i zakończenia zadania i zależności obliczeń na elementach przetwarzających PE (ang. *when*). Szeregowanie rozwiązuje także problemy współdzielenia zasobów sprzętowych [194].
- Implementacja modelu – automatyczne uszczegółowienie abstrakcyjnego projektu wysokiego poziomu przez dodanie szczegółów implementacji [19] i generację opisu dla sprzętu i oprogramowania.
- Weryfikacja i symulacja implementacji – obejmuje przygotowanie testów, testowanie zaimplementowanego układu za pomocą współbieżnej symulacji elementów układowych i programowych bez potrzeby tworzenia pełnej implementacji lub emulacji z użyciem FPGA [246][247].

W praktyce projektowej zaznacza się tendencja skracania czasu dostarczenia produktu na rynek i zmniejszania kosztów. Realizuje się to najczęściej poprzez:

- wykorzystanie gotowych reprogramowalnych architektur SoC; proces projektowy nie zawiera wówczas etapu eksploracji architektury, gdyż jest ona narzucona przez układ SoC i modyfikowalna w niewielkim zakresie;
- wykorzystanie gotowych komponentów [207]; proces projektowy nie zawiera wówczas etapu modelowania, weryfikacji, estymacji parametrów i implementacji komponentów, gdyż zakłada się, że są one odpowiednio zamodelowane, zweryfikowane, dostępne są ich implementacje i parametry.

Układy SoC powstały jako odpowiedź na potrzeby projektowania systemów wbudowanych. W latach ’80 układy ASIC traktowano jako akceleratory pewnych funkcji, działające jako koprocesory dla CPU i realizujące określone algorytmy – stąd nazwa AoC [238]. Projektanci zakładali przy tym, że CPU i ASIC znajdują się w oddzielnych chipach – systemy jednomodułowe [274]. Obecnie postęp technologii umożliwia integrację w jednym układzie scalonym logiki konfigurowalnej, programowalnego CPU i pamięci. Projektowanie z użyciem SoC zapewnia łatwość specyfikacji i weryfikacji systemu, elastyczność w rozwiązywaniu przeciwstawnych kryteriów projektowych, szybkie prototypowanie układów. Duża pojemność logiczna układów pozwala na realizację skomplikowanych aplikacji przy dużym zbiorze różnych konfiguracji architektury. Przepustowość komunikacji wewnątrz układu SoC jest znacznie większa niż pomiędzy układami dyskretnymi, co redukuje narzut komunikacyjny między modułami. Architektura SoC może zostać skonfigurowana do potrzeb aplikacji (np. zbiór instrukcji procesora, rozmiar pamięci cache). Dodatkowo użycie RTOS ułatwia zarządzanie ograniczeniami czasu rzeczywistego [20]. Układy SoC posiadają także wystarczającą elastyczność do zapewnienia rozwoju produktu i tworzenia całej rodziny produktów rozróżnialnych za pomocą oprogramowania [73].

Projektowanie z wykorzystaniem gotowych komponentów jest postrzegane jako kluczowy czynnik wzrostu wydajności projektowania systemów wbudowanych [134]. Producenci systemów wbudowanych rzadko rozpoczynają projektowanie od zera i wykorzystują komponenty z innych wykonanych uprzednio projektów. Poszerzyła się także oferta rynkowa w zakresie IPCores oraz bibliotek i jest dostępne całe spektrum realizacji poszczególnych komponentów, zarówno sprzętowych jak i programowych. Ponowne wykorzystanie IPCores zakłada, że zaprojektowane, zsyntezowane i zweryfikowane komponenty mogą być łatwo zintegrowane w nowo tworzonych projektach redukując ich złożoność i czasochłonność [134]. Przy wykorzystaniu gotowych komponentów pojawia się problem wyboru instancji implementacji danego komponentu. Projektant na podstawie parametrów implementacji musi podjąć decyzję o instancji użytej w realizacji całego systemu. W przypadku optymalizacji wielokryterialnej takiej decyzji nie da się podjąć bez analizy całego systemu.

Najważniejszym krokiem w procesie współbieżnego projektowania sprzętu i oprogramowania jest podział funkcjonalności na sprzęt i oprogramowanie. Z punktu widzenia projektanta i cyklu projektowania, pożądana metoda rozwiązania problemu podziału powinna charakteryzować się następującymi właściwościami:

- przewidywalny czas obliczeń – metoda powinna produkować rozwiązanie w zadanym czasie lub umożliwiać estymację czasu obliczeń w trakcie działania;
- wczesne uzyskanie rozwiązania dopuszczalnego – metoda powinna produkować inicjalne rozwiązanie dopuszczalne w możliwie krótkim czasie, co ułatwia przerwanie obliczeń po upływie zadanego czasu lub przez użytkownika;
- określenie alfa-optymalności – metoda powinna określać jak odległe od optimum jest obecnie wyprodukowane rozwiązanie dopuszczalne, co ułatwia przerwanie obliczeń po osiągnięciu określonego poziomu alfa-optymalności, co określa Definicja 2;

Definicja 2. Alfa-optymalność

Alfa-optymalność jest metryką różnicy wartości funkcji celu dla znalezionego rozwiązania i wartości funkcji celu rozwiązania optymalnego w odniesieniu do wartości funkcji celu dla rozwiązania optymalnego zgodnie z zależnością (1).

$$\alpha_{opt} = \frac{f(X) - f(X_{opt})}{f(X_{opt})} \tag{1}$$

gdzie:

α_{opt} - metryka alfa-optymalności,
$f(X)$ - wartość funkcji celu dla rozwiązania X,
$f(X_{opt})$ - wartość funkcji celu dla rozwiązania optymalnego X_{opt}.

- możliwość uzyskania rozwiązania optymalnego – metoda powinna być zbieżna do rozwiązania optymalnego, wykorzystywanego, gdy istnieje odpowiedni budżet czasu na optymalizację w cyklu projektowania;
- rozwiązanie rozszerzonego problemu podziału – metoda powinna ułatwiać projektowanie z wykorzystaniem wielu implementacji komponentów, co indukuje rozszerzony problem podziału;
- wielokryterialna funkcja celu [50] – metoda powinna mieć możliwość definiowania dowolnej funkcji celu wyrażonej jako liniowa kombinacja parametrów rozwiązania, a także nieliniowej funkcji celu, co pokrywa obecnie stosowane kryteria np. wydajność / moc;
- ograniczenia na wiele parametrów – metoda powinna umożliwić nakładanie sztywnych ograniczeń na wiele parametrów, ale także kombinacji parametrów.

1.2. Cel, zakres i tezy rozprawy

We wprowadzeniu opisano specyfikę systemów wbudowanych i ich implementacji w systemach SoC przy użyciu współbieżnego procesu projektowania sprzętu i oprogramowania. Przedstawiono obecne praktyki projektowe i wynikające stąd modyfikacje procesu projektowania. Podkreślono znaczenie procesu podziału funkcjonalności na sprzęt i oprogramowania jako najważniejszego etapu projektowania i wyszczególniono pożądane właściwości metody rozwiązania problemu podziału.

W związku z tym cele niniejszej rozprawy sformułowano następująco:

- **analiza i porównanie metod optymalizacji wielokryterialnej stosowanych do rozwiązania rozszerzonego problemu podziału;**
- **opracowanie narzędzi do automatycznego optymalnego rozszerzonego podziału przy zadanych ograniczeniach i wielokryterialnej funkcji celu;**
- **zastosowanie narzędzi do automatycznego podziału do heterogenicznej implementacji algorytmów przetwarzania danych w układach SoC.**

Zakres rozprawy ograniczono do optymalizacji rozwiązania problemu podziału na sprzęt i oprogramowanie. Dokonano taksonomii metod optymalizacji, ich analizy pod kątem pożądanych właściwości, przeglądu rozwiązań opublikowanych w literaturze i porównania ich właściwości. Z teoretycznego porównania wynika, że metoda B&B (ang. *Branch and Bound*) wykazuje wiele zalet, które predestynują ją do zastosowania w rozwiązaniu problemu podziału. Szczególny nacisk położono na analizę metody B&B, a także możliwość odpowiedniego doboru jej parametrów za pomocą autokalibracji, przez co można uzyskać przyspieszenie obliczeń. Implementację metody B&B zastosowano w procesie podziału funkcjonalności na sprzęt i oprogramowanie do realizacji algorytmów kryptograficznych (jako przykładu algorytmów przetwarzania danych) w heterogenicznych systemach SoC.

Tezy rozprawy sformułowano następująco:

- **Metoda podziału i ograniczeń umożliwia wielokryterialną optymalizację rozwiązania rozszerzonego problemu podziału funkcjonalności na sprzęt i oprogramowanie przy zadanych ograniczeniach.**
- **Automatyczny dobór parametrów metody podziału i ograniczeń umożliwia przyspieszenie optymalizacji rozwiązania problemu podziału funkcjonalności na sprzęt i oprogramowanie.**

1.3. Układ pracy

W rozdziale 2 sformułowano problem podziału modelu systemu na sprzęt i oprogramowanie i przedstawiono definicję binarnego i rozszerzonego problemu podziału. Zdefiniowano parametry heterogenicznej implementacji systemu wbudowanego: zużycie pamięci ROM, zużycie pamięci RAM, zużycie zasobów sprzętowych, czas wykonania oraz zużycie mocy. Zaprezentowano metody wyznaczania i estymacji parametrów poszczególnych modułów oraz metody agregacji poszczególnych parametrów dla całego systemu.

W rozdziale 3 dokonano taksonomii metod optymalizacji stosowanych przy problemie podziału. Opisano każdą metodę i dla każdej z nich zaprezentowano przegląd opublikowanych zastosowań do problemu podziału. Następnie porównano cechy poszczególnych metod i dokonano oceny ich możliwości zastosowania do rozwiązania problemu podziału.

W rozdziale 4 zaprezentowano szczegóły działania metody podziału i ograniczeń oraz przedstawiono parametry metody: funkcję celu, ograniczenia, rozwiązanie początkowe, strategię iteracji, strategię wyboru podproblemu, strategię rozgałęzień i funkcję ograniczenia dolnego.

Rozdział 5 przedstawia przeprowadzone badania i uzyskane wyniki. W podrozdziale 5.1 przedstawiono zrealizowane środowisko heterogenicznej symulacji sprzętowo-programowej, które posłużyło do weryfikacji poszczególnych modułów, do ekstrakcji niektórych parametrów, a także do walidacji całego systemu. W podrozdziale 5.2 opisano aplikację Partitioner używaną do dalszych eksperymentów podziału funkcjonalności na sprzęt i oprogramowanie, reprezentację DAG, generator

DAG oraz zdefiniowano funkcję ograniczenia dolnego. Podrozdział 5.3 przedstawia analizę wrażliwości parametrów metody podziału i ograniczeń oraz automatyczną kalibrację parametrów metody w celu przyspieszenia obliczeń. Podrozdział 5.4 prezentuje wyniki 2 eksperymentów implementacji algorytmu kryptograficznego na platformie SoC.

Rozdział 6 przedstawia wnioski z przeprowadzonych badań i podsumowuje udowodnione tezy.

W rozdziale 7 zaprezentowano obszerny wykaz cytowanej literatury używanej do opracowania niniejszej rozprawie doktorskiej. Przedstawiono 281 pozycji, głównie z konferencji i czasopism IEEE.

W dodatku A przedstawiono algorytmy kryptograficzne jako dziedzinę aplikacyjną użytą w niniejszej rozprawie doktorskiej. Opisano wprowadzenie do algorytmów kryptograficznych. Dokonano klasyfikacji algorytmów na symetryczne i asymetryczne, przedstawiono struktury algorytmów symetrycznych, zaprezentowano współczesne algorytmy symetryczne. W kolejnych podrozdziałach opisano algorytmy finałowej rundy konkursu AES: MARS, RC6, Rijndael, Serpent i Twofish. W podrozdziale 8.8 opisano badania implementacji algorytmów kryptograficznych w FPGA. W podrozdziale 8.9 przedstawiono kryptoanalizę poprzez ataki w kanale bocznym, ze szczególnym uwzględnieniem analizy DPA i metodom jej zapobiegania. Wyrównywanie mocy implementacji FPGA algorytmu Rijdnael zaprezentowano w podrozdziale 8.10.

W dodatku B przedstawiono architektury heterogenicznych implementacji algorytmów kryptograficznych i zaprezentowano atrybuty kryptograficznego systemu wbudowanego. W kolejnych podrozdziałach opisano poszczególne paradygmaty przetwarzania: w podrozdziale 9.2 – procesor z dedykowanym akceleratorem, w podrozdziale 9.3 – procesor z koprocesorem rozszerzającym listę rozkazów, a w podrozdziale 9.4 – procesor z rekonfigurowalnym koprocesorem.

2. Podział modelu systemu na sprzęt i oprogramowanie

Kluczowym krokiem procesu projektowania heterogenicznego systemu wbudowanego jest podział modelu systemu pomiędzy sprzęt i oprogramowanie, czyli decyzja, które moduły (części modelu) powinny zostać zrealizowane sprzętowo, a które programowo [10][264]. W ogólności podział systemu może następować na N komponentów sprzętowych lub programowych, przy czym każdy może posiadać własne ograniczenia. Decyzja podjęta na etapie podziału jest krytyczna, ponieważ bezpośrednio wpływa na charakterystyki wydajności i koszt finalnego rozwiązania [73]. Każdy moduł jest przydzielany do określonego komponentu w architekturze w taki sposób, żeby moduły miały dostęp do danych i możliwa była komunikacja pomiędzy modułami. Każdy moduł może zostać zaimplementowany na wiele sposobów (np. na poziomie algorytmu, transformacji lub zasobów) zarówno w sprzęcie lub jako oprogramowanie dając w rezultacie rodzinę punktów projektowych, zwanych krzywą Pareto [54][139]. Tradycyjnie punkty projektowe przedstawia się na krzywej kompromisu (ang. *tradeoff curve*) powierzchni krzemu względem czasu obliczeń. W przypadku optymalizacji wielokryterialnej krzywa jest zdefiniowana w przestrzeni wielowymiarowej.

2.1. Sformułowanie problemu podziału modelu systemu

W systemach przetwarzania danych zadania wchodzą ze sobą w interakcje przez przekazywanie danych i są najczęściej modelowane grafem SDF [174], DFG, CDFG [201] albo za pomocą opisu behawioralnego [256]. Opis systemu jest następnie konwertowany do acyklicznego grafu skierowanego DAG:

$$\mathrm{DAG} = (\mathbf{V}, \mathbf{E}) \tag{2}$$

gdzie:

V = {v_1, ..., v_n} jest zbiorem wierzchołków, |**V**| = n,
E = {e_1, ..., e_m} jest zbiorem skierowanych krawędzi, |**E**| = m.

W zależności (2) wierzchołek v_i reprezentuje zadanie (lub ogólnie część systemu) v_i, a jego waga $w(v_i)$ reprezentuje rozmiar zadania. Krawędź $e_{ij} = (v_i, v_j)$ reprezentuje komunikację (lub ogólnie zależność albo przepływ danych) pomiędzy zadaniem v_i oraz v_j, a jego waga $w(e_{ij})$ reprezentuje koszt komunikacji. Każdy wierzchołek i każda krawędź jest etykietowana atrybutami pochodzącymi z procesu estymacji [170]. Graf skierowany posiada co najmniej jeden wierzchołek początkowy (źródłowy) i co najmniej jeden wierzchołek końcowy [131].

DAG jest daną wejściową dla algorytmów podziału [174]. Innymi modelami są graf TG [20] oraz ITPG, używany w przypadkach, gdy istnieje co najmniej jedna wychodząca krawędź dla każdego wierzchołka – tzw. graf zstępujący. DAG może być modelem fragmentu algorytmu powtarzanego iteracyjnie [52]. Ma to szczególne znaczenie w przypadku algorytmów obróbki obrazu (np. JPEG) oraz algorytmów kryptograficznych symetrycznych. DAG może być wygenerowany automatycznie z innej specyfikacji [139] lub może zostać wprowadzony ręcznie przez projektanta. Rozmiary grafu zależą od modelowanego problemu i mogą mieć rozmiar od kilku wierzchołków [139] aż do kilkuset [10].

Definicja 3 poniżej określa binarny problem podziału, gdzie dla każdego modułu dostępna jest tylko jedna realizacja sprzętowa i jedna programowa. Definicja 4 formułuje rozszerzony problem podziału, gdzie dla każdego modułu istnieje wiele możliwych realizacji sprzętowych i programowych.

Definicja 3. Binarny problemu podziału [139]

Mając dany DAG, ograniczenia, koszt implementacji sprzętowej i programowej dla każdego wierzchołka DAG i koszt komunikacji dla każdej krawędzi DAG należy określić dla każdego wierzchołka alokację do sprzętu lub oprogramowania (wybór binarny) i czas rozpoczęcia wykonywania zadania tak, aby zminimalizować koszt przy spełnieniu ograniczeń.

Definicja 4. Rozszerzony problemu podziału [139]

Mając dany DAG, ograniczenia, opcje i koszt implementacji sprzętowych i programowych (ang. *implementation bin*) dla danego wierzchołka DAG i koszt komunikacji dla każdej krawędzi DAG należy określić dla każdego wierzchołka alokację do odpowiedniego komponentu architektury, implementację ze zbioru możliwych implementacji (wybór rozszerzony) i czas rozpoczęcia wykonywania zadania tak, aby zminimalizować koszt przy spełnieniu ograniczeń.

Rozszerzony problem podziału został po raz pierwszy sformułowany i rozwiązany w pracy [139]. W pracach [51][56] rozpatrywane są krzywe kompromisu zużycia zasobów (powierzchni krzemu) względem czasu obliczeń.

Wynikiem procesu podziału jest alokacja – przydział zadań do określonych komponentów architektury, odwzorowanie – wybór implementacji ze zbioru możliwych, szeregowanie zadań - wyznaczenie czasów rozpoczęcia i zakończenia każdego zadania i informacja o komunikacji potrzebnej pomiędzy zadaniami [139][169][170].

W trakcie procesu podziału określa się współdzielenie zasobów: czy zadanie może wykorzystać istniejący zasób sprzętowy wykorzystywany już przez inne zadanie. Najczęściej nie dopuszcza się współdzielenia zasobów [174].

Proces podziału powinien być rozwiązywany wspólnie z problemem wyznaczenia szeregowania zadań, tzn. ustalenia czasu ich rozpoczęcia i zakończenia [54]. Dla systemów przetwarzania danych stosuje się z reguły szeregowanie statyczne. Dla systemów kontroli stosuje się z reguły szeregowanie dynamiczne z użyciem RTOS. Szeregowanie zadań powinno być rozwiązywane równolegle do procesu podziału [73][139][231]. Częściej spotyka się szeregowanie wykonywane sekwencyjnie w stosunku do podziału, co prowadzi najczęściej do nieoptymalnych rozwiązań.

Proces podziału definiuje format reprezentacji danych wyjściowych i ich potencjalne użycie. Każda część modelu jest następnie implementowana za pomocą specyficznych narzędzi syntezy: oddzielnie jest kompilowane oprogramowanie i syntezowane moduły sprzętowe [89].

W kategoriach złożoności obliczeniowej problem podziału modelu systemu w zależności od dodatkowych założeń jest problemem:

- NP-zupełnym (ang. *NP-complete*) [5][18][84][91][136][201],
- NP-trudnym (ang. *NP-hard*) [10][39][121][122][139].

Rozszerzony problem podziału ma złożoność $(2B)^N$, gdzie B jest rozmiarem zbioru implementacji a N liczbą zadań [139]. Przy ustalonej architekturze problem podziału funkcjonalności jest redukowalny do problemu podziału grafu na dwie składowe [75][84][200] i jest równoważny problemowi kolejkowania instrukcji [276]. W pracy [10] wykazano, że problem podziału grafu na dwa podgrafy przy pustym zbiorze krawędzi jest redukowalny do problemu plecakowego. Jeśli zaniedba się koszt komunikacji, wówczas można skorzystać z istniejących podoptymalnych algorytmów wielomianowych dla rozwiązania problemu plecakowego. Dodatkowa optymalizacja funkcji kosztu przy ograniczeniach powoduje, że problem staje się NP-trudny.

Ponieważ przestrzeń projektowa rośnie eksponencjalnie ze wzrostem rozmiaru problemu (liczby modułów), jednym z ważniejszych parametrów określonych przed problemem podziału jest granularność. Określa ona najmniejszą niepodzielną funkcjonalność używaną podczas podziału taką jak zadania, procesy, procedury, pętle, bloki, wyrażenia, wyrażenia arytmetyczne, wyrażenia logiczne [165]. Aby problem był rozwiązywalny w określonym czasie, należy odpowiednio zdefiniować i dobrać poziom szczegółowości opisu [88][89][264]:

- Podział gruboziarnisty (ang. *coarse-grain*) typowy dla podziału ręcznego, jest definiowany jako współbieżna synteza [122] na poziomie zadania lub funkcji i używa luźno powiązanej architektury procesora z koprocesorem. Dla podziału gruboziarnistego komunikacja jest generowana, jeśli zadania są przydzielone do różnych komponentów i w modelu występują jawnie operacje komunikacji.
- Podział drobnoziarnisty (ang. *fine-grain*) typowy dla podziału automatycznego, jest definiowany jako podział właściwy [122] na poziomie bloku podstawowego (ang. *basic block*) lub wyrażenia i używa ściśle związanej architektury procesora z koprocesorem lub ASIP. Dla podziału drobnoziarnistego wymagana jest analiza przepływu danych do znalezienia wymaganej komunikacji. Podział drobnoziarnisty indukuje większą przestrzeń poszukiwań.

Podział drobnoziarnisty dokonuje najczęściej ekstrakcji bloków podstawowych i może przesunąć logicznie spójne bloki do różnych komponentów lub umieścić niezwiązane bloki w jednym komponencie. Podział może ponadto utworzyć implementację, która jest zupełnie odmienna od inicjalnego modelu [51]. Podział drobnoziarnisty jest opisany m.in. w pracach [1][88][91][111][112][129][147]. Podział gruboziarnisty nie dekomponuje inicjalnego modelu [51]. Podział gruboziarnisty jest opisany m.in. w pracach [85][138][139][240]. W pracy [119] zaproponowano podejście heterogenicznej granulacji, w której najpierw wykonuje się podziału gruboziarnistego, a następnie podziału drobnoziarnistego. W niniejszej pracy skupiono się głównie na problemach podziału gruboziarnistego, choć większość rozważań można przenieść również do podziału drobnoziarnistego.

Klasyczny problem podziału funkcjonalnego polega na odwzorowaniu modelu systemu na daną architekturę (ang. *platform-based design approach*) [89]. Inne problemy podziału:

- Synteza architektury modułów, pamięci i magistral dla danego modelu systemu,
- Minimalizacja komunikacji (liczby magistral) – konieczność zapewnienia komunikacji pomiędzy modułami jest jednym z problemów współbieżnego projektowania sprzętu i oprogramowania. Jedynie nieliczne prace biorą pod uwagę czas komunikacji [37][88][136]. Konieczność rozważania komunikacji komplikuje proces podziału, z uwagi na dodatkowy czas komunikacji, zasoby do komunikacji i potrzebę synchronizacji [88]. Z tego też powodu niektóre techniki podziału dokonują minimalizacji komunikacji redukując w ten sposób całkowity czas przetwarzania [111]. W pracy [147] dokonano opisu podejść do problemu podziału ze względu na traktowanie komunikacji.
- Podział na architekturę rekonfigurowalną dynamicznie - jest rozpatrywany w dwóch wymiarach: przestrzennym i czasowym [166]. Tradycyjne metody podziału zajmują się tylko wymiarem przestrzennym, w którym różna funkcjonalność jest fizycznie realizowana jednocześnie w różnych zasobach sprzętowych. Podział czasowy oznacza, że układ może być rekonfigurowany w różnych fazach wykonania programu. Algorytm podziału czasowego musi uwzględniać koszt dynamicznej rekonfiguracji, integrować optymalizacje

kompilatora i przeszukiwanie przestrzeni projektowej oraz kierować się informacjami z profilowania.

2.2. Parametry implementacji i ich estymacja

W celu optymalności decyzji podejmowanych podczas podziału funkcjonalności na część sprzętową i programową, należy zapewnić, że dane do procesu podziału są dokładne [57]. Niedokładność danych wejściowych prowadzi do przeszacowania parametrów i bardziej kosztownej implementacji, lub do niedoszacowania parametrów i implementacji niespełniającej narzuconych wymagań.

Właściwości i parametry systemu są wyznaczane na podstawie [161]:

- analizę modeli formalnych – analiza statyczna zakłada, że system jest kolejkowany w sposób statyczny i ignorowane są zjawiska dynamiczne, takie jak czasy oczekiwania na dostęp do magistrali; analiza statyczna znajduje zastosowanie w projektowaniu aplikacji czasu rzeczywistego [18]
- estymacji parametrów – projektowanie za pomocą modułów IPCore zezwala na uzyskanie parametrów bezpośrednio od producenta lub z ponownie użytej części projektu [89]; zgrubne estymaty uzyskuje się przez analizę kodu źródłowego;
- symulację odpowiedzi systemu na konkretny rodzaj pobudzenia - symulacja systemu jest wykonywana przy użyciu modeli komponentów i ich komunikacji na różnych poziomach abstrakcji, użycie różnych poziomów abstrakcji umożliwia kompromis pomiędzy szybkością symulacji i jej dokładnością [246][247].

W praktyce projektowej stosuje się zazwyczaj wyznaczanie wartości metryk przez estymację z wykorzystaniem bibliotek [147] lub symulację. W pracy [139] estymaty powierzchni i czasu przetwarzania są uzyskane ze środowiska Ptolemy. Dokładniejsze wartości metryk można uzyskać przez generację implementacji każdego modułu na każdej potencjalnej platformie wchodzącej w skład przestrzeni projektowej [89], przed wykonaniem podziału na części sprzętowe i programowe. Wartości te są generowane przez narzędzia syntezy i kompilacji [192] i stanowią dane wejściowe do procesu podziału. W niniejszej pracy estymaty parametrów implementacji uzyskano na drodze ekstrakcji z narzędzi kompilacji i syntezy oraz heterogenicznej symulacji [246][247].

Taksonomia metryk:

- liniowa agregacja lokalnych wartości (addytywne) lub globalne;
- stałe w czasie (statyczne) lub zmienne w czasie (dynamiczne).

2.2.1. Zużycie pamięci ROM

W systemach wbudowanych pamięć ROM jest używana do przechowywania programu oraz stałych danych. Może być również używana przez moduły sprzętowe, choć wymaga to dedykowanych modułów pamięci ROM. Metryka zużycia pamięci ROM jest addytywna i statyczna. Moduły programowe mogą współdzielić kod programu, wówczas w celu prawidłowej agregacji wartości zużycia pamięci ROM, moduły należy podzielić na mniejsze. Programy samomodyfikujące się nie korzystają z pamięci ROM i klasyfikują się jako dynamiczne struktury danych.

Estymacja pamięci programu daje stosunkowo zgrubne wyniki. W pracy [256] do estymacji zaproponowano metryki LoC. W pracy [230] oprócz metryki LoC dodano liczbę równań matematycznych oraz gęstość linii w diagramie blokowym.

Najdokładniejsze dane zużycia pamięci programu uzyskuje się przy zastosowaniu kompilatorów dla procesora docelowego [192]. Kompilatory generują pliki z dokładną mapą pamięci przeznaczonej dla każdego modułu lub funkcji.

2.2.2. Zużycie pamięci RAM

W systemach wbudowanych pamięć RAM jest używana do przechowywania zmiennych globalnych, zmiennych lokalnych, danych dynamicznych oraz do zachowywania stanu przy wywołaniu podprogramu lub zmianie kontekstu. Może być również używana przez moduły sprzętowe, choć wymaga to dedykowanych modułów pamięci RAM. Metryka zużycia pamięci RAM jest addytywna i dynamiczna, gdyż zajętość pamięci RAM zależy od danych wejściowych i ścieżek wykonania programu (np. dynamiczne struktury danych, rekursja). Zużycie pamięci RAM jest zmienne w czasie i głównym celem projektanta jest dostarczyć właściwą estymatę ograniczenia górnego zużycia pamięci.

Estymacja pamięci danych daje bardzo zgrubne wyniki. Używany jest przy tym graf TG [20]. Do wyznaczenia ograniczenia górnego należy zsumować rozmiar wszystkich zmiennych globalnych i statycznych, a następnie przy użyciu grafu TG przeanalizować w stosie wywołań funkcji (ang. *call stack*) zmienne lokalne i dane potrzebne do zachowania stanu.

Dokładniejsze dane można uzyskać na drodze symulacji całego systemu przy zastosowaniu reprezentatywnego zbioru wektorów pobudzeń wejściowych.

2.2.3. Zużycie zasobów sprzętowych

W systemach wbudowanych zasoby sprzętowe służą do realizacji CPU oraz dowolnych funkcji jako akceleratory. Metryka zużycia zasobów sprzętowych jest addytywna i statyczna, co oznacza pełną niezależność modułów od siebie [147][168] i możliwość obliczenia zużycia zasobów metodą wstępującą (ang. *bottom-up*) [111]. W przypadku możliwości współdzielenia zasobów lub rekonfiguracji sprzętu metryka zużycia zasobów sprzętowych staje się odpowiednio nieliniowa i dynamiczna, lecz wartość uzyskana z metryki addytywnej statycznej stanowi górne oszacowanie zużycia zasobów sprzętowych. Współdzielenie zasobów można realizować statycznie na podstawie informacji z kolejkowania statycznego lub dynamicznie za pomocą dodatkowego protokołu arbitrażu.

Estymacja zużycia zasobów sprzętowych daje bardzo zgrubne wyniki. Narzędzia estymacji korzystają z bibliotek jedynie w odniesieniu do standardowych modułów [146].

Dokładne dane zużycia zasobów sprzętowych otrzymuje się za pomocą narzędzi syntezy i implementacji [192].

2.2.4. Czas wykonania

W systemach wbudowanych metryka czasu wykonania jest globalna i statyczna. W przypadku dynamicznego współdzielenia zasobów (np. układów dedykowanych, pamięci cache, potoku procesora) metryka czasu wykonania staje się dynamiczna, co powoduje, że wartość uzyskana z metryki statycznej nie stanowi górnego ograniczenia czasu wykonania.

Estymacja czasu wykonania modułów programowych jest zagadnieniem trudnym. Estymacja czasu wykonania bez kompilacji kodu wymaga sortowania topologicznego węzłów DFG i ich translacji na generyczny zbiór instrukcji, a następnie określenia przez zbiór technologiczny dla docelowego procesora [107][147]. Czas wykonania DFG jest otrzymywany przez sumowanie czasów wykonania generycznych instrukcji

zgodnie z otrzymanym profilem. Estymacja czasu wykonania modułów sprzętowych wymaga określenia ścieżki krytycznej i obliczenia liczby cykli zegara potrzebnych do wykonania danej operacji [146]. W pracy [192] czas wykonania jest szacowany za pomocą analizy statycznej skompilowanego kodu lub syntezowanego sprzętu. Uwzględnianie zjawisk dynamicznych czyni estymację jeszcze trudniejszą [88].

Dokładne dane czasu wykonania modułów programowych realizuje się przez symulację w symulatorze ISS [146][215]. Czas wykonania modułów sprzętowych może zostać pomierzony za pomocą symulacji HDL lub w układzie rzeczywistym za pomocą wbudowanych liczników [110]. Największym problemem jest efektywne wykonanie heterogenicznej symulacji sprzętowo-programowej [246][247].

2.2.5. Zużycie mocy

W systemach wbudowanych moc jest zużywana przez CPU, pamięci, moduły sprzętowe i magistrale. Metryka zużycia mocy jest addytywna [75] i statyczna. Układy SoC posiadają różne tryby niskiego poboru mocy w trakcie bezczynności procesora [33], co pozwala na stosowanie liniowego modelu poboru mocy. Efektem posiadania różnych stanów poboru mocy jest opóźnienie przejścia między stanami, co wprowadza zjawiska dynamiczne utrudniające analizę czasu wykonania.

Estymacja poboru mocy operuje często stosowanym przybliżeniem [108]:

$$\mathrm{P_{total}} = \alpha * \mathrm{C} * \mathrm{V_{dd}^2} * \mathrm{f} \qquad (3)$$

gdzie:

- α – współczynnik aktywności,
- C – całkowita pojemność przełączana,
- V_{dd} – napięcie zasilania,
- f – częstotliwość przełączania.

Głównym problemem zależności (3) jest wyznaczenie współczynnika aktywności oraz całkowitej pojemności przełączania. W estymacji poboru mocy stosuje się różne metryki: EPI [268], EPC – odzwierciedla specyfikę procesorów CISC [145], TCPC – nie wymaga syntezy modelu [215]. W pracy [259] przedstawiono metodę uzyskania danych o poborze mocy dla każdej instrukcji CPU, a w rezultacie dla danego programu wykonywanego na CPU.

Technikę estymacji poboru mocy komponentów i całego systemu przedstawiono w pracy [162]. Danymi wejściowymi są specyfikacja systemu, pobudzenie wejściowe i architektura systemu. Danymi wyjściowymi są statystyka i profil zużycia mocy na poziomie komponentów i całego systemu.

Metoda działa w 3 fazach:

- informacje uzyskane z heterogenicznej symulacji sprzętowo-programowej całego systemu są abstrahowane do grafu BSE,
- moduły składowe i komunikacja są modelowane prostym modelem mocy, bez modelowania zjawisk dynamicznych,
- graf BSE jest przetwarzany przez profiler generując profil mocy dla całego systemu z uwzględnieniem zjawisk dynamicznych.

Dokładniejsze dane zużycia mocy otrzymuje się z symulatorów poboru mocy. W pracach [110][250][252][253] do oszacowania poboru mocy użyto symulatora mocy Xilinx XPower. Aby uzyskać dokładne wskaźniki poboru mocy należy użyć pliku VCD uzyskanego po procesie syntezy fizycznej. Istnieje niewiele narzędzi do analizy mocy, a istniejące narzędzia projektowania nie wspomagają symulacji poboru mocy z wystarczającą dokładnością [39], są bardzo wolne i niepraktyczne w użyciu [259].

3. Metody optymalizacji stosowane przy problemie podziału

Najlepsze rozwiązanie problemu podziału można znaleźć przeglądając wszystkie możliwe rozwiązania. Metoda taka jest jednak niepraktyczna z uwagi na wykładniczą liczbę możliwych rozwiązań. Istotą metod optymalizacji stosowanych przy problemie podziału jest sposób, według którego wybierają podzbiór możliwych rozwiązań do poszukiwań. Algorytmy optymalizacyjne stosowane przy problemie podziału można podzielić na [176]:

- dokładne / enumeracyjne, które przeszukują całą przestrzeń projektową, wyłączając z eksploracji obszary z uwagi na ich właściwości lub założenia i kolejno znajdując rozwiązania częściowe (np. ILP, DP, B&B);
- metaheurystyczne / iteracyjne, które przeglądają przestrzeń projektową w wybranych punktach wraz z otoczeniem, modyfikują rozwiązanie przez przesunięcie w poszukiwaniu lepszego rozwiązania, używana jest funkcja celu do oceny każdego rozwiązania (np. GA, SA, TS);
- heurystyczne / konstruktywne, które grupują obiekty w partycje używając metryk bliskości, lub konstruują rozwiązanie na podstawie metryk lokalnych, przeszukiwana jest relatywnie niewielka przestrzeń projektowa (np. algorytm zachłanny, FL, BBN, KLFM, klastrowanie, GCLP).

W algorytmach dokładnych / enumeracyjnych przeszukiwana jest cała przestrzeń projektowa, co daje gwarancję znalezienia rozwiązania optymalnego globalnie. W celu redukcji złożoności przeszukiwania algorytmy dokonują wyłączania z eksploracji obszarów poprzez:

- przesuwanie się zgodnie z kierunkiem najsilniejszego wzrostu funkcji celu (metoda gradientowa) do rozwiązań o większej wartości funkcji celu wzdłuż płaszczyzn odcinających – LP;
- odcinanie płaszczyznami rozwiązania niecałkowitoliczbowe – ILP;
- wykazanie ich nieoptymalności lub niedopuszczalności – B&B, DP.

W algorytmach metaheurystycznych / iteracyjnych przeszukiwana jest przestrzeń projektowa w wybranych punktach wraz z otoczeniem. Algorytmy rozpoczynają od partycji startowej i dokonują przeszukiwania sąsiedztwa, po czym kontynuują przeszukiwania przechodząc do sąsiedniego punku. Akceptowane są rozwiązania polepszające jak i pogarszające koszt, co umożliwia wyjście z lokalnego minimum i uzyskanie lepszych rozwiązań, choć brak jest gwarancji osiągnięcia minimum globalnego. Algorytmy iteracyjne różnią się od siebie głównie [17]:

- metodą generacji rozwiązania: losowe – GA, SA; deterministyczne – TS;
- ilością jednoczesnych przeszukiwań: pojedyncze – SA; zbiorowe – GA, TS;
- ilością utrzymywanych rozwiązań: pojedyncze – SA, TS; populacja – GA;
- bliskością przeszukiwania: bliskie sąsiedztwo – SA, TS; dalekie sąsiedztwo – GA.

W algorytmach heurystycznych / konstruktywnych przeszukiwana jest relatywnie niewielka przestrzeń projektowa. Rozwiązania całkowite są konstruowane z optymalnych rozwiązań (lub poprawianych) wyznaczanych niezależnie za pomocą określonych metryk. Algorytmy różnią się od siebie:

- metrykami: przystosowania pojedynczego zadania – algorytm zachłanny, FL; prawdopodobieństwa hipotez – BBN; zmiany kosztu po zamianie pomiędzy partycjami – KLFM; bliskości pomiędzy klastrami – klastrowanie; koszt zadania w oparciu o globalną krytyczność – GCLP;

- ilością jednocześnie podejmowanych zmian: przydział 1 zadania – algorytm zachłanny, FL, klastrowanie, GCLP; wymiana lub przeniesienie k zadań – algorytm KLFM; zmiana prawdopodobieństwa dla wszystkich zadań – BBN;
- możliwością wyjścia z lokalnego minimum: brak – algorytm zachłanny, FL, klastrowanie, GCLP; przez jednoczesną wymianę lub przeniesienie więcej niż 1 zadania – KLFM; przez zmianę prawdopodobieństwa dla wszystkich zadań – BBN.

Logika rozmyta i sieci Bayesa są różnymi metodami określenia niepewności stosowanych metryk. Logika rozmyta określa stopień przynależności do zbioru rozmytego, sieć Bayesa określa subiektywne prawdopodobieństwo binarnej przynależności do zbioru.

Właściwości opisanych klas metod optymalizacyjnych podsumowuje Tabela 1.

Tabela 1. Taksonomia metod optymalizacyjnych

	Przeszukiwana przestrzeń	Rozwiązanie	Przykład
Dokładne / Enumeracyjne	Przeszukiwana jest cała przestrzeń, z wyłączeniem nierokujących regionów	Znajdowane są rozwiązania częściowe, z których konstruowane jest rozwiązanie całkowite	1) Programowanie liniowe całkowitoliczbowe ILP 2) Programowanie dynamiczne DP 3) Metoda podziału i ograniczeń B&B
Metaheurystyczne / Iteracyjne	Przeszukiwana jest przestrzeń w wybranych punktach, wraz z otoczeniem	Znajdowane są rozwiązania całkowite, które są zmieniane i polepszane w kolejnych iteracjach	1) Algorytm genetyczny GA 2) Symulowane wyżarzanie SA 3) Poszukiwanie tabu TS
Heurystyczne / Konstruktywne	Przeszukiwana jest niewielka przestrzeń projektowa	Rozwiązania całkowite konstruowane optymalnych rozwiązań wyznaczanych niezależnie za pomocą określonych metryk	1) Algorytm zachłanny 2) Logika rozmyta FL 3) Sieci BBN 4) Heurystyka KLFM 5) Klastrowanie 6) Heurystyka GCLP

3.1. Programowanie Liniowe Całkowitoliczbowe ILP

Programowanie liniowe LP jest najszerzej używaną techniką dla problemów z ograniczeniami [40][57]. Sformułowanie programowania liniowego składa się ze zbioru zmiennych nieujemnych (4), zbioru nierówności liniowych (5), pojedynczej funkcji liniowej wielu zmiennych, która stanowi funkcję celu (6). Programowanie liniowe całkowitoliczbowe ILP dodaje warunek (7), aby zmienne przyjmowały tylko wartości całkowite. ILP jest matematycznym modelowaniem zmiennych wpływających na problem podziału [73].

Programowanie liniowe całkowitoliczbowe jest zdefiniowane [40][124] jako:

$$x_j \geq 0, \mathrm{j} = 1,2,\ldots,\mathrm{n} \tag{4}$$

$$\sum_{\mathrm{j}=1}^{\mathrm{n}} a_{ij} * x_j \leq b_i, \mathrm{i} = 1,2,\ldots,\mathrm{m} \tag{5}$$

$$\max Z = \sum_{j=1}^{n} c_j * x_j \tag{6}$$

$$x_j \in \mathbf{Z}, \mathrm{j} = 1,2,\ldots,\mathrm{n} \tag{7}$$

gdzie:

Z – funkcja celu,
x_j – zmienna decyzyjna,
a_{ij} – współczynnik wagowy ograniczenia,

b_i – wartość ograniczenia,
c_j – współczynnik wagowy funkcji celu,
m – liczba ograniczeń,
n – liczba zmiennych,
Z – zbiór liczb całkowitych.

Problem programowania liniowego LP może zostać rozwiązany:

- metodą simplex (George Danzig 1947 rok). Rozwiązanie optymalne znajduje się w jednym z dopuszczalnych punktów przecięcia krańcowych ograniczeń (ang. *feasible cornerpoint*). Metoda przegląda skończoną ilość wierzchołków przesuwając się do sąsiednich wierzchołków o większej wartości funkcji celu i osiąga maksimum w punkcie, dla którego wartość funkcji celu jest większa niż dla wszystkich punktów sąsiednich [57]. W praktyce metoda operuje na tablicy sympleksów dokonując jej przekształceń. Metoda ma złożoność obliczeniową wykładniczą w najgorszym przypadku ($O(2^n)$), natomiast średnia złożoność jest wielomianowa względem liczby ograniczeń i liczby zmiennych ($O(m^2*n)$) [40].
- metodą punktu wewnętrznego (ang. *interior point*):
  - algorytmem elipsoidy (Leonid Chaczijan 1979 rok) [40]. Metoda umieszcza wielowymiarowe elipsoidy wewnątrz regionów dopuszczalnych, osie elipsoid determinują kierunek kolejnych iteracji, a rozmiary elipsoid asymptotycznie zmniejszają się do punktu w miarę dochodzenia do rozwiązania optymalnego. Algorytm ma wielomianową złożoność obliczeniową ($O(m+n)n^3$)), lecz jego wadą jest zależność od zakresu danych, niestabilność numeryczna i długi czas wykonania [40].
  - algorytmem Karmarkara (Narendra Karmarkar 1984 rok). Metoda używa transformacji z geometrii w celu oszacowania kierunku przesunięć przez wnętrze regionu dopuszczalnego. Algorytm ma wielomianową złożoność obliczeniową. Metoda jest bardzo wydajna dla bardzo dużych problemów LP.

Algorytm 1 przedstawia działanie metoda sympleksów.

Algorytm 1. Schemat metody sympleksów rozwiązania programowania liniowego [57]

1. Inicjalizacja: znalezienie dowolnego dopuszczalnego punktu przecięcia krańcowych ograniczeń. Standardowa postać LP umożliwia przyjęcie punktu (0,0,…,0). Jeśli nie znaleziono żadnego dopuszczalnego punktu, równania są sprzeczne.
2. Sprawdzenie optymalności: sprawdzenie bieżącej wersji funkcji celu, czy jest dostępna do wprowadzenia nowa zmienna bazowa (ang. *entering basic variable*). Jeśli żadna nie jest dostępna, bieżące rozwiązanie jest rozwiązaniem optymalnym.
3. Wprowadzenie nowej zmiennej bazowej: spośród zmiennych niestanowiących bazy, wybór tej, która daje najszybszy wzrost wartości funkcji celu.
4. Usunięcie starej zmiennej bazowej (ang. *leaving basic variable*): spośród zmiennych stanowiących starą bazę wybór tej, która daje najszybsze ograniczenie wzrostu wartości nowej zmiennej bazowej.
5. Uaktualnienie równań w tablicy sympleksów metodą Gaussa.

W metodzie sympleksów kroki 2-5 wykonywane są iteracyjnie aż do kryterium stopu zdefiniowanego w kroku 2. W praktyce stosuje się zrewidowaną metodę

simplex (ang. *revised simplex method*), efektywniejszą do implementacji w postaci programu komputerowego, używającą macierzy rzadkich, zawierającą wiele uproszczeń i eliminującą zbędne obliczenia. W celu dalszego przyspieszenia obliczeń stosuje się także transformację do reprezentacji dualnej (ang. *dual simplex method*) i rozwiązanie problemu dualnego, jeśli problem dualny posiada mniej ograniczeń, a przez to mniej punktów przecięcia krańcowych ograniczeń.

Programowanie liniowe całkowitoliczbowe ILP nakłada warunek (7) na wartość zmiennych. W pierwszym kroku rozwiązywany jest zrelaksowany problem bez ograniczenia na całkowitoliczbowe rozwiązania. Jeśli otrzymane rozwiązanie nie spełnia warunków całkowitoliczbowości, konstruowane są dodatkowe ograniczenia:

- Algorytm Dakina [57] bazuje na metodzie podziału i ograniczeń B&B. W kolejnych krokach budowane jest binarne drzewo metody B&B dodające ograniczenia na zmienne niecałkowite (jedno ograniczające w dół, drugie ograniczające w górę), po czym dokonuje się ponownego rozwiązania dwóch podproblemów. Algorytm kończy się po rozwinięciu wszystkich gałęzi w rozwiązania całkowitoliczbowe lub ich wyeliminowaniu.
- Algorytm Gomory'ego [40] bazuje na dodawaniu płaszczyzn odcinających. W kolejnych krokach odcina się od wielościanu rozwiązań zadania poprzedni punkt optymalny o współrzędnych ułamkowych, ale nie odcina żadnego z punktów o współrzędnych całkowitych tego wielościanu, po czym dokonuje się ponownego rozwiązania problemu. Algorytm kończy po osiągnięciu rozwiązania całkowitoliczbowego.

W metodzie programowania liniowego nie stosuje się reguł heurystycznych, cała wiedza o zadaniu jest dostarczana za pomocą równań i nierówności. Wszystkie zależności muszą być liniowe – algorytm kolejkowania musi być także dostarczony w postaci programowania liniowego, co znacznie zwiększa liczbę zmiennych oraz ograniczeń. Rozwiązanie rozszerzonego problemu podziału jest jeszcze bardziej skomplikowane obliczeniowo. W pracy [139] podano przykład dla N=15 i B=5, co wymaga 718 ograniczeń oraz 396 zmiennych. W praktyce komercyjne narzędzia (ang. *solvers*) rozwiązują zadania o rozmiarze tysięcy zmiennych. Aby problem został rozwiązany w rozsądnym czasie wymagana jest właściwa granulacja.

W pracy [208] zastosowano model MILP do rozwiązania binarnego problemu podziału dla heterogenicznych architektur wieloprocesorowych. Funkcją celu składa się z liniowej kombinacji wydajności systemu i kosztu systemu. Ograniczeniom podlegają odwzorowania, przydział zadań do procesora, typy transferu, gotowości danych, czasy rozpoczęcia i zakończenia wykonania zadania oraz transferu danych, współdzielenie zasobów. Niektóre ograniczenia są relacjami nieliniowymi i muszą podlegać linearyzacji. Przykładowy system z 4 zadaniami indukuje 21 zmiennych czasowych, 72 zmienne binarne i 174 ograniczenia, i jest obliczany poniżej 1 minuty. System z 9 zadaniami indukuje 47 zmiennych czasowych, 225 zmiennych binarnych i 1081 ograniczeń, i jest obliczany około 1 dnia.

W pracach [177][190][191][192] przedstawiono bardzo wyszukany model ILP do rozwiązania wspólnego problemu podziału funkcjonalności na sprzęt i oprogramowanie i szeregowania zadań w systemie COOL. Na podstawie opisu systemu tworzony jest graf DAG służący do podziału, definiowana jest docelowa technologia, definiowane są dostępne implementacje sprzętowe i programowe, a następnie tworzony jest model kosztu. Modelowane są koszty i ograniczenia zasobów, koszty i ograniczenia projektu, odwzorowanie instancji w elementy przetwarzające. Dokonano również porównania zastosowanego modelu MILP z

podziałem za pomocą algorytmów genetycznych i heurystycznego kolejkowania zadań.

Metoda ILP w problemie podziału funkcjonalności na sprzęt i oprogramowanie napotyka na następujące ograniczenia:

- wymaga, aby wszystkie dane do zadania były sformułowane jako zbiór równań i nierówności, co jest trudne dla algorytmów szeregowania zadań, powoduje to zazwyczaj gwałtowny wzrost liczby zmiennych i ograniczeń;
- nie ma możliwości dostarczania dodatkowej wiedzy o zadaniu w postaci heurystyk;
- możliwe rozwiązanie zadania programowania liniowego całkowitoliczbowego otrzymuje się dopiero na końcu (w odróżnieniu od problemu programowania liniowego, gdzie każdy punkt pośredni jest rozwiązaniem możliwym);
- złożoność obliczeniowa jest wykładnicza, nie można przewidzieć dokładnego czasu obliczeń.

3.2. Programowanie Dynamiczne DP

W programowaniu dynamicznym [57] podejmowane są powiązane decyzje w sposób optymalny. Metoda polega na zdefiniowaniu podproblemu i znalezieniu jego optymalnego rozwiązania. Następnie podproblem jest powiększany i jego optymalne rozwiązanie jest znajdowane przy użyciu poprzednio znalezionego rozwiązania. Postępując w ten sposób dochodzi się do problemu oryginalnego i otrzymuje jego rozwiązanie na podstawie kolejnych rozwiązań. Metoda działa w sposób rekursywny dodając kolejne informacje na stosie wywołań, przy czym może wykorzystywać uzyskane uprzednio informacje z rozwiązania wspólnych podproblemów (ang. *overlapping problems*). Po zakończeniu działania rozwiązanie jest dostępne na stosie w odpowiedniej sekwencji. Metoda programowania dynamicznego może być traktowana jako szczególny przypadek metody podziału i ograniczeń [124]. Schemat metody przedstawia Algorytm 2.

Algorytm 2. Schemat metody programowania dynamicznego [57]

1. Zdefiniuj małą część całego problemu i znajdź optymalne rozwiązanie tej małej części.
2. Zwiększ nieznacznie małą część i znajdź optymalne rozwiązanie nowego problemu używając uprzednio znalezione rozwiązanie optymalne.
3. Kontynuuj krok 2 aż powiększenie spowoduje, że bieżący problem zawiera problem oryginalny. Jeśli problem oryginalny jest rozwiązany, kryteria stopu są spełnione.
4. Skonstruuj rozwiązanie całego problemu z optymalnych rozwiązań małych problemów, rozwiązanych w trakcie powiększania problemu.

Metoda programowania dynamicznego zakłada, że problem może być podzielony na etapy (ang. *stages*), gdzie każdy etap stanowi nowy podproblem do rozwiązania. Każdy etap zawiera pewną liczbę stanów, gdzie każdy stan oznacza informację niezbędną do rozwiązania problemu na danym etapie. Decyzja podejmowana na każdym etapie uaktualnia stan na danym etapie do stanu na następnym etapie. W bieżącym etapie optymalna decyzja dla pozostałych etapów jest niezależna od decyzji podjętych na poprzednich etapach. Definiuje się rekursywną zależność między decyzją na danym etapie a optymalną decyzją na etapie poprzednim.

$$f_n(d_n) = F\left(x_n, f_{n-1}(d_{n-1})\right) \tag{8}$$

gdzie:

- n – numer etapu,
- f_n – wartość funkcji celu na etapie n-tym,
- d_n – stan na etapie n-tym,
- F – funkcja celu,
- x_n – decyzja podjęta na etapie n-tym.

W pracach [129][130] problem podziału jest zdefiniowany jako znalezienie podzbioru kandydatów, które umożliwią największe przyspieszenie przy jednoczesnym ograniczeniu zasobów. Problem jest transformowany do problemu plecakowego i rozwiązywany metodą programowania dynamicznego. Zastosowano hierarchiczny wybór kandydatów, po którym dokonuje się profilowania i estymacji. Badane jest przyspieszenie poszczególnych operacji, bez analizy czasów komunikacji i zależności pomiędzy węzłami grafu DAG.

W pracy [51] opisano algorytm podziału Codex-dp stosowany do systemów komunikacyjnych, minimalizujący powierzchnię przy ograniczeniach opóźnienia. Podział jest gruboziarnisty, jeśli jest dokonywana komunikacja w trakcie procesu, wówczas dokonuje się dekompozycji procesów na podprocesy i nie są uwzględniane czasy komunikacji. Szeregowanie zadań jest wykonywane metodą programowania dynamicznego rozszerzonego o łańcuchy powiązań (ang. *binning strings*).

W pracy [147] zaprezentowano algorytm podziału PACE używany w systemie projektowania LYCOS. Rozwiązywany jest problem minimalizacji czasu obliczeń przy ograniczeniach zasobów i dualny problem minimalizacji zasobów przy ograniczeniu czasu obliczeń. Problem jest transformowany do zmodyfikowanego problemu plecakowego, rozszerzonego o analizę współdzielenia zasobów.

Praca [136] stanowi rozwinięcie pracy [147]. Problem podziału jest zdefiniowany identycznie i analogicznie redukowany do problemu plecakowego. W przeciwieństwie do algorytmu PACE, który bazuje na przydziale sekwencji bloków, proponowany algorytm SPACE bazuje na przydziale tylko bieżącego bloku w danym momencie. Powoduje to spadek złożoności obliczeniowej do $O(n*A)$ względem $O(n^2*A)$ w algorytmie PACE i w rezultacie znaczne przyspieszenie obliczeń.

W pracy [37] zaprezentowano podział drobnoziarnisty i kolejkowanie metodą programowania dynamicznego. Problem minimalizacji opóźnienia systemu przy ograniczeniach zasobów sprzętowych bez ich współdzielenia jest rozwiązywany w czasie wielomianowym. W przypadku współdzielenia zasobów problem jest NP-trudny. Rozważany jest graf DAG w podejściu „z dołu do góry”, gdzie etapami są węzły rozpatrywane w porządku topologicznym. Rozwiązania częściowe są przechowywane w postaci list. Rozwiązanie jest otrzymywane przez wybór jednego z rozwiązań z węzła poprzedzającego oraz dodanie parametrów bieżącego węzła.

W pracy [231] zdefiniowano problem podziału zakładający minimalizację całkowitego czasu wykonania. Architektura systemu składa się z procesora i wielu modułów FPGA pracujących współbieżnie i komunikujących się z procesorem za pomocą przerwań. Programowanie dynamiczne definiuje funkcję rekurencyjną, która jest opisywana przez dwa parametry: bieżąca konfiguracja FPGA i kolejka funkcji oczekujących na wykonanie.

Programowanie dynamiczne nadaje się dobrze dla problemów dyskretnych z niewielką liczbą etapów, stanów i decyzji podejmowanych na każdym etapie, które wykazują optymalne podstruktury (ang. *optimal substructure*). Przy ograniczeniach na wartość każdego parametru oraz problemie szeregowania zadań następuje olbrzymia ilość etapów, stanów i decyzji, a zależność rekursywna nie jest zależnością ani addytywną, ani multiplikatywną. Etapami są zazwyczaj kolejne

zadania do implementacji. Stan jest definiowany jako dostępna kombinacja zasobów, co daje eksplozję kombinatoryczną stanów. Aby ograniczyć liczbę stanów ograniczeniom podlega tylko jeden zasób, którego ciągłe wartości są dodatkowo zgrubnie kwantowane do wartości dyskretnych, np. zasoby sprzętowe [129][136][147] lub czas wykonania [231]. Decyzjami do podjęcia na danym etapie jest wybór implementacji, więc programowanie dynamiczne dobrze nadaje się do rozwiązania rozszerzonego problemu podziału. Z uwagi na konieczność sformułowania prostej zależności rekurencyjnej, nie stosuje się współbieżnego wykonywania zadań, a łączny czas wykonania jest sumą czasów wykonania zadań i potrzebnej komunikacji. Nie jest potrzebny więc algorytm szeregowania zadań.

Metoda programowania dynamicznego daje optymalny podział przy założeniu sekwencyjności wykonywanych operacji sprzętowych i programowych (brak algorytmu szeregowania zadań i prosta zależność rekurencyjna), braku współdzielenia zasobów i nałożeniu ograniczeń tylko na jeden z parametrów poddany dodatkowo zgrubnemu kwantowaniu. Metoda ma złożoność wielomianową względem ilości etapów (węzłów DAG), stanów (skwantowanych wartości parametrów) i decyzji (liczby implementacji).

3.3. Metoda Podziału i Ograniczeń B&B

Metoda jest algorytmem enumeracyjnym [124] i rozwiązuje problem n-wymiarowy przeszukując przestrzeń rozwiązań oraz trawersując drzewo poszukiwań w celu znalezienia dopuszczalnego rozwiązania o najmniejszej wartości funkcji celu [32]. Inicjalnie problem n-wymiarowy odpowiada całej nieprzebadanej przestrzeni rozwiązań i wierzchołkowi korzeniowi drzewa poszukiwań. W każdym kroku metody dokonywana jest redukcja problemu do podproblemów o wymiarze mniejszym o 1, odpowiadająca dekompozycji przestrzeni na rozłączne podprzestrzenie i generacji wierzchołków potomnych w drzewie. Każdy podproblem, podprzestrzeń i wierzchołek potomny powstają przez ustalenie kolejnego elementu rozwiązania [78]. W każdej iteracji metody, status rozwiązania jest opisany przez listę nieprzeszukanych podprzestrzeni tzw. listę *live* (wierzchołki drzewa poszukiwań) i aktualne rozwiązanie optymalne [62].

Algorytm 3. Schemat metody podziału i ograniczeń

1. Inicjalizacja (ang. *initialization*): dodanie całej przestrzeni do listy *live* i znalezienie wartości rozwiązania początkowego jako górnego ograniczenia funkcji celu.
2. Rozgałęzienie (ang. *branching, growing, expanding*): wybór elementu z listy *live* według określonej strategii wyboru i podział zbioru na rozłączne podzbiory – węzły w drzewie poszukiwań.
3. Ograniczenie (ang. *bounding*): obliczenie ograniczenia dolnego dla podzbioru.
4. Test: jeśli ograniczenie dolne jest większe od aktualnego minimum, wówczas podzbiór można odrzucić (ang. *prune*), gdyż nie prowadzi do poprawy. w przeciwnym wypadku podzbiór jest dodawany do listy *live*. Jeśli z bieżącego rozwiązania częściowego nie można wyprowadzić żadnego rozwiązania dopuszczalnego, wówczas podzbiór można również odrzucić. Jeśli ograniczenie górne jest mniejsze od aktualnego minimum, wówczas należy przypisać nowe minimum i jeśli wynikające z niego rozwiązanie jest dopuszczalne, to podzbioru nie trzeba dodawać do listy *live* (ang. *fathom*).

Algorytm 3 pokazuje, że kroki podziału, ograniczenia i testu są wykonywane iteracyjnie aż do wyczerpania węzłów w drzewie poszukiwań (opróżnienia listy *live*)

[62] lub do momentu, gdy wartość aktualnego minimum będzie mniejsza od najmniejszej wartości ograniczenia dolnego dla wszystkich nieprzebadanych gałęzi na liście *live* [57].

Metoda może zostać opisana formalnie następująco [62][124]:

$$\min \{ f(x) \mid x \in S \} \tag{9}$$

gdzie:

- f – funkcja celu (ang. *objective function*), f: S→**R**,
- S – dyskretny zbiór możliwych rozwiązań (ang. *feasible region*), S⊆P, spełniający narzucone ograniczenia,
- P – dyskretny zbiór potencjalnych rozwiązań.

W przypadku binarnego problemu podziału HW/SW, wykonuje się podział dychotomiczny (na dwa podzbiory), w przypadku rozszerzonego problemu podziału HW/SW wykonuje się podział politomiczny (na więcej niż dwa podzbiory) [62]. Wygenerowane podprzestrzenie są zazwyczaj rozłączne, choć nie jest to warunek konieczny. Jeśli moc podzbioru po podziale jest mniejsza niż zbioru przed podziałem, wówczas metoda B&B jest zbieżna w przestrzeniach dyskretnych [62].

W pracach [38][39] optymalizowany jest czas wykonania (maksymalizacja przepustowości) przy ograniczeniu zasobów sprzętowych (liczba bramek) i pobieranej mocy. W metodzie B&B zadania są wybierane zgodnie z kolejnością w grafie DAG (Branch First). Funkcja ograniczenia dolnego estymuje czas wykonania zakładając, że każde następne zadanie zostanie zrealizowane w sprzęcie.

W pracy [18] minimalizowane są zasoby sprzętowe przy ograniczeniu czasu wykonania. W metodzie B&B analizowane są zadania zgodnie z kolejnością w grafie DAG (Branch First). Szukanie odbywa się zgodnie ze strategią BeFS, co pozwala wyeliminować więcej odgałęzień.

W pracy [52] wprowadzone są ograniczenia czasu wykonania i zasobów sprzętowych. W metodzie B&B rozwiązanie początkowe generowane jest metodą zachłanną w oparciu o metrykę odpowiedniości zadania do implementacji w HW (ang. *suitability*). Użyto strategii rozgałęzień Branch Suitable i strategii wyboru podproblemu DFS. Zastosowano potokowe szeregowanie zadań z niezerowym czasem komunikacji i możliwością współdzielenia zasobów. Zrealizowano podział dla DAG z 12 węzłami i DAG z 30 węzłami, dla którego czas obliczeń nie przekraczał 30 minut.

W pracy [20] optymalizowane jest zużycie zasobów sprzętowych przy spełnieniu ograniczeń czasowych. Dla każdego rozgałęzienia zadania są porządkowane według priorytetów, dla danego poziomu implementacje są porządkowane według kosztów tak, że lewa krawędź to rozwiązania SW. Stosowane jest strategia trawersowania wszerz BFS z lewa na prawo. Jeśli znajdowane jest rozwiązanie, które da się kolejkować, prawa strona rozwiązań może być wyeliminowana i algorytm schodzi na dół do kolejnego poziomu – następnego zadania. Główny problem pojawia się, gdy podczas eksploracji drzewa rozwiązanie HW jest wybrane po rozwiązaniu SW. W tym przypadku algorytm musi wrócić do góry w celu dodania czasu komunikacji SW/HW i ponownego sprawdzenia kolejkowalności.

W pracy [135] optymalizacji podlega czas wykonania przy ograniczeniu zużycia zasobów sprzętowych. Do znajdowania rozwiązania początkowego używany jest algorytm zachłanny. Bloki są sortowane według efektywności, a następnie w tej kolejności są przydzielane do HW. Drzewo trawersowane jest wszerz BFS z lewa na prawo, rozpoczynając od korzenia drzewa i schodząc o jeden poziom niżej, gdzie znajdują się 2 węzły: HW i SW. Przy sprawdzaniu ograniczenia każde częściowe

rozwiązanie zostaje rozszerzone do rozwiązania całkowitego przez przydział pozostałych węzłów. Dolne ograniczenie oblicza się jako suma czasu wykonania rozwiązania częściowego oraz dolnego ograniczenia czasu wykonania pozostałych węzłów. Przy przydziale nieprzydzielonych węzłów są one sortowane według efektywności, a następnie węzły są przydzielane do HW aż do osiągnięcia ograniczenia powierzchni. Na tej podstawie jest obliczany czas wykonania rozwiązania częściowego.

W pracy [10] rozpatruje się dwa warianty optymalizacji: minimalizacja kosztu HW przy ograniczaniu kosztu SW oraz minimalizacji kosztu SW przy ograniczaniu kosztu HW. Architekturą docelową jest procesor z jednym kontekstem SW oraz koprocesor z jednym kontekstem HW. Kosztem SW jest czas działania, koszt HW to powierzchnia, energia cieplna, pobór mocy. Przyjęto założenia, że czas HW jest równy 0, a czas komunikacji pomiędzy modułami w tym samym kontekście jest pomijalny. Brak jest kolejkowania, bo czas HW jest równy 0, a czas SW jest sumą czasów komponentów SW.

W pracy [276] problem kolejkowania instrukcji jest sformułowany metodą ILP a rozwiązywany przy pomocy metody B&B. Zastosowano rozbudowany model formalny, minimalizacji podlega suma dewiacji poboru mocy przy ograniczeniach zasobów i czasu wykonania. Szczegółowo opisano algorytm wyznaczania funkcji ograniczenia dolnego oraz strategii rozgałęzienia.

Metoda podziału i ograniczeń w naturalny sposób rozwiązuje rozszerzony problem podziału przy wielokryterialnej optymalizacji. Metoda daje rozwiązanie optymalne i umożliwia wcześniejsze przerwanie obliczeń. Metoda umożliwia zastosowanie modelu matematycznego przeszukiwania binarnego, a także heurystyk przy obliczaniu funkcji celu i funkcji ograniczenia dolnego (np. kolejkowanie), traktowaniu ograniczeń (np. funkcja kary). Metoda zawiera wiele parametrów (rozwiązanie początkowe, strategie iteracji, strategie rozgałęzień, strategie wyboru podproblemu), które odpowiednio dobrane mogą znacząco skrócić czas obliczeń.

3.4. Algorytm Genetyczny GA

Algorytm genetyczny GA jest heurystyką poszukiwań, która naśladuje proces naturalnej ewolucji [57][132][213]. Populacja osobników, których chromosomy stanowią zakodowane rozwiązania problemu optymalizacji, ewoluuje w kierunku lepszych rozwiązań. Rozwiązania mogą być kodowane binarnie 0/1 lub w inny dowolny sposób, algorytm nie przetwarza bezpośrednio parametrów (fenotypów), lecz jedynie ich zakodowaną postać (chromosomy). Ewolucja rozpoczyna się od losowego wygenerowania populacji osobników, przy czym liczność populacji może zależeć od rozmiaru zadania (z reguły 100-1000). W każdej generacji obliczane jest przystosowanie każdego osobnika, na podstawie ich przystosowania osobniki są losowo wybierane z bieżącej populacji, a ich chromosomy podlegają modyfikacjom (reprodukcji, krzyżowaniu i mutacji), tworząc nową populację. Nowa populacja jest następnie używana w nowej iteracji algorytmu. Zakończenie algorytmu może nastąpić:

- po osiągnięciu pożądanego przystosowania,
- po wygenerowaniu określonej liczby generacji,
- po upływie określonego czasu,
- jeśli zadana liczba kolejnych iteracji nie poprawia wartości średniego (lub najgorszego) przystosowania dla populacji – wartość najlepszego przystosowania może nie oddawać wewnętrznych zmian populacji [57].

Algorytm 4. Schemat algorytmu genetycznego [57][213]

0. Projekt algorytmu: wybór rozmiaru populacji *n* i prawdopodobieństwa mutacji; wybór operatorów i warunków stopu.
1. Losowa generacja populacji inicjalnej i obliczenie wartości dopasowania dla każdego osobnika. Najlepsze rozwiązanie jest determinowane przez osobnika o najlepszej wartości funkcji dopasowania w inicjalnej populacji.
2. Zastosowanie operatora reprodukcji do bieżącej populacji i wygenerowanie puli krzyżujących się osobników o rozmiarze *n*.
3. Zastosowanie operatora krzyżowania do puli krzyżujących się osobników i wygenerowanie tymczasowej nowej populacji.
4. Zastosowanie operatora mutacji do tymczasowej nowej populacji i utworzenie finalnej nowej populacji. Obliczenie wartości dopasowania dla każdego osobnika w nowej populacji i uaktualnienie wartości najlepszego rozwiązania, jeśli znaleziono lepsze rozwiązanie w nowej populacji
5. Jeśli spełnione są warunki stopu, wartość najlepszego rozwiązania jest rozwiązaniem końcowym. W przeciwnym przypadku kontynuacja od punku 2.

Schemat działania przedstawia Algorytm 4, zaś szczegółowy opis działania, a także dowód zbieżności algorytmu można znaleźć w pracach [132][213]. GA jest chętnie stosowany do problemów optymalizacyjnych, np. projektowanie filtrów cyfrowych IIR o nietypowych charakterystykach [236], minimalizacja liczby tranzystorów w kombinacyjnych układów cyfrowych [237], podział systemu na układy VLSI [153], oraz podział funkcjonalności na sprzęt i oprogramowanie.

GA umożliwia proste modelowanie problemu [10][213]:

- populacja = zbiór wektorów,
- osobniki = wektor będący zbiorem parametrów rozwiązania zadania,
- chromosomy = zakodowana w postaci łańcucha postać zbioru parametrów rozwiązania zadania, będąca ciągiem genów,
- gen = zakodowany wybór implementacji zadania,
- allel = wartość danego genu, czyli wybór implementacji zadania,
- generacja = iteracja algorytmu,
- funkcja przystosowania = maksymalizowana funkcja celu.

Stosowane są następujące operatory w bazowym algorytmie genetycznym [57]:

- reprodukcja – determinuje ilość egzemplarzy poszczególnych osobników, prawdopodobieństwo przetrwania osobnika jest proporcjonalne do jego przystosowania, najczęściej stosowana jest tu metoda ruletki [213],
- krzyżowanie – dwa losowo wybrane osobniki rodzicielskie tworzą dwa osobniki potomne, przez losowy wybór pozycji krzyżowania (ang. *locus*) i zamianę części genów w chromosomach,
- częściowe krzyżowanie – krzyżowanie zapewniające uzyskanie osobnika o niezerowym dopasowaniu (rozwiązania dopuszczalnego),
- inwersja – w jednym osobniku dokonuje się odwrócenia sekwencji genów chromosomu pomiędzy dwoma losowo wybranymi pozycjami chromosomu,
- mutacja – używany do losowej zmiany allela (wartości jednego genu) w chromosomie na inną losowo wybraną wartość.

W celu efektywnej pracy i osiągania dobrych rozwiązań bliskich optimum, algorytm genetyczny wymaga strojenia parametrów takich jak sposób generacji

inicjalnej populacji, rozmiar populacji, prawdopodobieństwo mutacji i krzyżowania, schematy krzyżowania, ograniczanie dominujących osobników podczas reprodukcji.

W pracach [15][56] opisano system EPICURE wykorzystujący algorytm genetyczny do przeszukania przestrzeni projektowej. Kodowanie chromosomu odpowiada przydziałowi zadania do punktu na krzywej kompromisu powierzchni i czasu przetwarzania. Chromosom składa się z N genów, gdzie N odpowiada ilości wierzchołków. Każdy gen jest kodowany wartością 0-100, która określa procentowe zużycie zasobów sprzętowych względem najbardziej kosztownej implementacji HW. Operator mutacji wybiera losowo gen i dokonuje zmiany jego wartości, co może zmienić punkt implementacji na krzywej kompromisu. Zastosowano 5 operacji mutacji: 2-Opt, 3-Opt (operatory przeszukiwania sąsiedztwa), DoubleBridge (operator wyskakuje z obecnej przestrzeni poszukiwań), CutAndPaste (operator zamienia dwie części chromosomu), Scramble (operator skrambluje geny pomiędzy dwoma losowo wybranymi punktami). Zastosowano 2 operatory krzyżowania: 1p-Cross oraz 2p-Cross. Algorytm kończy działanie, jeśli w ciągu określonej liczbie populacji N_{gen}=100 nie nastąpiła poprawa. Alokacja i szeregowanie zadań są rozłączne: alokacja jest wykonywana w fazie przeszukiwania przestrzeni projektowej, podczas gdy szeregowanie zadań jest wykonywane podczas ewaluacji każdego rozwiązania.

W pracy [50] zaproponowano algorytm genetyczny z wielokryterialną funkcją dopasowania, który przeszukuje przestrzeń projektową z uwzględnieniem ograniczeń i stosuje metodę hybrydową z użyciem B&B do przekształcenia rozwiązania niedopuszczalnego w dopuszczalne. Z powodu istnienia wzajemnie wykluczających ograniczeń operacje mutacji i krzyżowania mogą wygenerować rozwiązanie niedopuszczalne, niespełniające podanych wcześniej ograniczeń. Metodę zastosowano do drobnoziarnistego podziału funkcjonalności przy projektowaniu rozszerzenia listy rozkazów procesora.

Rozwiązania otrzymane przy użyciu GA nie mają żadnego limitu optymalności. W pracy [10] pokazano, że optymalizacja przy użyciu GA może dawać w wyniku rozwiązania, których wartość funkcji celu przewyższa optimum o 100%. GA umożliwia akceptowanie rozwiązań niedopuszczalnych, które mogą być następnie karane przez funkcję dopasowania [10]. Zaletą algorytmów genetycznych zastosowanych do problemu podziału HW/SW jest możliwość wygenerowania akceptowalnych rozwiązań w relatywnie krótkim czasie. Algorytm utrzymuje zbiór rozwiązań uaktualnianych podczas każdej iteracji a dzięki reprodukcji wybierane są najlepsze rozwiązania. Operator krzyżowania umożliwia uzyskanie lepszych rozwiązań, natomiast operator mutacji umożliwia eksplorację nowych obszarów poszukiwań, dzięki czemu algorytm nie utyka w lokalnym minimum.

3.5. Symulowane Wyżarzanie SA

Metoda symulowanego wyżarzania SA [57][132][144] jest metodą optymalizacji globalnej, która emuluje proces fizycznego wyżarzania metali. Metoda rozpoczyna się z inicjalnym rozwiązaniem oraz inicjalną wysoką temperaturą T_{max}. Gdy temperatura jest wysoka, eksplorowana jest przestrzeń rozwiązań i akceptowane są niemal wszystkie przypadkowe zmiany. W każdym kroku temperatura jest stopniowo zmniejszana T=rT. W miarę zmniejszania się temperatury, wykonywanych jest coraz mniej przypadkowych zmian, aż stan osiąga zamrożenie. Przeszukiwanie przestrzeni jest niedeterministyczne, dla każdej temperatury generowane są sąsiednie rozwiązania w sposób przypadkowy. Kryterium stopu jest określone przez zamrożenie stanu, czyli brak zmian przez ustaloną liczbę iteracji. Algorytm 5 prezentuje schemat metody symulowanego wyżarzania.

Algorytm 5. Schemat metody symulowanego wyżarzania [57]

0. Inicjalizacja: znalezienie inicjalnego rozwiązania ***S*** przez jego losową generację; wybór inicjalnej (wysokiej) temperatury *T>0*; wybór wartości współczynnika chłodzenia *r*.
1. Wybór losowego rozwiązania ***S'***, sąsiedniego w stosunku do rozwiązania ***S***.
2. Obliczenie różnicy kosztów: $\Delta E=cost(\boldsymbol{S'})-cost(\boldsymbol{S})$.
3. Decyzja o akceptacji nowego rozwiązania: jeśli $\Delta E<0$ (***S'*** jest lepsze niż ***S***), wówczas rozwiązanie jest akceptowane zawsze ***S***=***S'***, w przeciwnym przypadku (***S'*** jest gorsze niż ***S***) rozwiązanie jest akceptowane z prawdopodobieństwem określonym zależnością (10) lub (11).
4. Jeśli spełnione są warunki stopu, wartość najlepszego rozwiązania jest rozwiązaniem końcowym. W przeciwnym przypadku redukowana jest temperatura *T=rT* i następuje kontynuacja od punku 1.

Rozwiązanie sąsiednie w przestrzeniach dyskretnych jest generowane poprzez operacje prostego przesunięcia (ang. *simple* move), zamiany pomiędzy dwoma parametrami (ang. *pairwise exchange*) lub innych, które indukują dużą przestrzeń poszukiwań w jak najmniejszej ilości ruchów.

Prawdopodobieństwo P akceptacji rozwiązania **S'** przy rozwiązaniu **S** jest dodatnie nawet, gdy cost(**S'**) > cost(**S**). Zabezpiecza to przed utknięciem w lokalnych minimach. Gdy T→0, wówczas w przypadku cost(**S'**) > cost(**S**) prawdopodobieństwo P→0, a dla przypadku przeciwnego P→1. W oryginalnej metodzie SA rozwiązanie polepszone jest zawsze akceptowane. Prawdopodobieństwo P zostało zdefiniowane:

$$\mathrm{P}(\Delta\mathrm{E},\mathrm{T}) = e^{-\left(\frac{\Delta E}{T}\right)} \tag{10}$$

gdzie:

P – prawdopodobieństwo akceptacji rozwiązania,
ΔE – różnica wartości funkcji celu, ΔE = cost(**S**')-cost(**S**)
T – aktualna temperatura.

W innych opracowaniach prawdopodobieństwo zostało definiowane rozkładem Boltzmanna [132][144]:

$$\mathrm{P}(\Delta\mathrm{E},\mathrm{T}) = \frac{1}{1+e^{\left(\frac{\Delta E}{cT}\right)}} \tag{11}$$

gdzie:

P – prawdopodobieństwo akceptacji rozwiązania,
ΔE – różnica wartości funkcji celu, ΔE = cost(**S**')-cost(**S**)
T – aktualna temperatura,
c – stała Boltzmanna.

Algorytm przesuwa się ku lepszym rozwiązaniom, okazjonalnie akceptując także gorsze rozwiązania [18]. Unika się w ten sposób utknięcia w lokalnym minimum i umożliwia uzyskanie globalnego minimum (ang. *hill climbing*). Prawdopodobieństwo globalnego minimum zmierza do 1 wraz ze wzrostem liczby iteracji algorytmu (temperatury i tempa chłodzenia), nie ma jednak gwarancji jego osiągnięcia.

W systemie Cosyma [88] używany jest algorytm symulowanego wyżarzania, gdzie wszystkie zadania (instrukcje) inicjalnie są przypisane do SW. W każdej iteracji algorytmu następuje przenoszenie zadań (instrukcji) do HW przy minimalizacji czasu przetwarzania i spełnieniu ograniczeń zużycia zasobów sprzętowych. Algorytm

rozpoczyna od rozwiązania leżącego poza przestrzenią dopuszczalnych rozwiązań. Funkcja celu jest poszerzona o funkcję kary dla rozwiązań niedopuszczalnych.

W pracach [84][85] zaprezentowano metodę SA używaną w systemie Cadlab. Optymalizowano funkcję celu będącą liniową kombinacją 3 składników przy ograniczeniach pamięci programowej oraz zużycia zasobów sprzętowych. Dla każdej wielkości grafu zachodzi potrzeba strojenia parametrów. Stosowane są dwa rodzaje operacji: proste przesunięcie (ang. *simple move*) – losowy wybór węzła i przydział do przeciwnego zbioru, oraz ulepszone przesunięcie (ang. *improved move*) – losowy wybór węzła i przydział węzła wraz z sąsiedztwem do przeciwnego zbioru. Przy akceptacji gorszych rozwiązań stosowany jest rozkład prawdopodobieństwa zdefiniowany zależnością (10).

W pracach [25][26] zastosowano dynamiczną funkcję kosztu, która umożliwia częstsze osiąganie lepszych rozwiązań przez eksplorację regionów nieprzeszukiwanych przy użyciu tradycyjnej funkcji kosztu. Algorytm SA operuje na grafie wywołań (ang. *call graph*) i oblicza przyrost metryki czasu wykonania. Funkcja kosztu jest liniową kombinacją czasu wykonania i zużycia zasobów. Rozwiązania niedopuszczalne przekraczające ograniczenia są karane przez szybki wzrost funkcji celu. Kryterium stopu jest zdefiniowane przez brak poprawy w kolejnych iteracjach, a ich ilość jest uzależniona od rozmiaru problemu. Stosowany jest rozkład prawdopodobieństwa zdefiniowany zależnością (10).

W pracy [119] w systemie HiPART zastosowano gruboziarnisty algorytm SA do podziału klastrów na HW i SW, które są następnie optymalizowane algorytmem KLFM. Przekroczenie ograniczeń jest karane przez wysoki wzrost funkcji kosztu.

Symulowane wyżarzanie SA może zostać szybko zaimplementowane i ma zastosowanie do szerokiego spektrum problemów [1][84][86][88]. Główną zaletą metody SA jest jej ogólność (SA jest meta-heurystyką). Wadami algorytmu są długi czas wykonania i potrzeba zestrojenia parametrów: schematu chłodzenia oraz funkcji celu / kosztu. W pracach [115][170] zastosowano dynamiczne chłodzenie, dzięki któremu nie jest potrzebne strojenie wielu parametrów algorytmu i zapewniono większą zbieżność. Funkcja kosztu musi uwzględniać przekroczenie parametrów za pomocą technik minimalizacji błędu średniokwadratowego, funkcji bariery lub kary. Techniki minimalizacji błędu średniokwadratowego umożliwiają uzyskanie dokładnej wartości parametru, zamiast jego minimalizacji (np. zajętości zasobów FPGA). Funkcja bariery stosowana jest przy sztywnych ograniczeniach na wartość parametru, natomiast w przypadku, gdy ograniczenia nie są sztywne, stosuje się funkcję kary.

3.6. Poszukiwanie Tabu TS

Poszukiwanie tabu TS jest metaheurystyczną metodą optymalizacji dyskretnej należącej do klasy iteracyjnych technik lokalnego przeszukiwania dynamicznego sąsiedztwa. Metoda TS została zaproponowaną przez Freda Glovera w latach 1986-89 [103][104] i bazuje na systematycznym wykorzystaniu adaptacyjnych lub ewolucyjnych form pamięci krótkookresowej (ang. *recency-based memory*), średniookresowej i długookresowej (ang. *frequency-based memory*), która umożliwia modyfikację stanu w czasie. Metoda TS używa procedur przeszukiwania sąsiedztwa oraz przeszukiwania lokalnego, modyfikując struktury sąsiedztwa w miarę postępu poszukiwań w sposób ekonomiczny i efektywny dzięki długookresowej pamięci adaptacyjnej. Metoda TS utrzymuje także w pamięci krótkookresowej listę tabu, czyli listę ostatnio odwiedzanych rozwiązań, dzięki czemu zapobiega się ruchom powrotnym i cyklom. Pamięć średniookresowa służy do wpływania na poszukiwania

w kierunku najbardziej rokujących obszarów. Wszystkie lokalne wybory korzystają z ilościowego modelu decyzyjnego uzupełnionego o reguły heurystyczne i są kierowane przez informację zbieraną podczas poszukiwań, co kontrastuje z bezstanowymi metodami poszukiwań (np. GA, SA) i ze ścisłymi pamięciowymi metodami (np. B&B) [116][160].

Algorytm 6. Schemat metody poszukiwania tabu z pamięcią krótkookresową

1. Inicjalizacja: lista tabu jest początkowo pusta, inicjalne rozwiązanie może być generowane losowo, najlepsze rozwiązanie jest równe inicjalnemu.
2. Przeszukiwanie sąsiedztwa: jeśli kandydat z sąsiedztwa nie znajduje się na liście tabu, wówczas dodawany jest do listy kandydatów.
3. Wybór najlepszego kandydata: lista kandydatów jest ewaluowana zgodnie z funkcją celu i wybierany jest kandydat o najlepszym dopasowaniu.
4. Akceptacja kandydata: jeśli dopasowanie kandydata jest lepsze od najlepszego rozwiązania, to jest on przyjmowany jako najlepsze rozwiązanie, a lista tabu jest uaktualniania o cechy nowego rozwiązania, usuwając najstarsze elementy.

Algorytm 6 wykonuje kroki 2-4 w sposób iteracyjny. Kryteriami stopu jest zazwyczaj limit czasu przeznaczony na poszukiwania, osiągnięcie określonej wartości funkcji celu lub brak zmian przez ustaloną liczbę iteracji.

W pracach [84][85] przedstawiono podział z wykorzystaniem metody TS w systemie Cadlab. Adaptacja metody TS do natury problemu podziału HW/SW okazała się trudniejsza niż w przypadku metody SA. Jednak determinizm metody TS ułatwił zestrojenie parametrów. Optymalizowano funkcję kosztu będącą liniową kombinacją 3 składników przy ograniczeniach zasobów sprzętowych i pamięci programowej. W implementacji metody TS użyto wiele ponownych ścieżek (ang. *restarting tours*), dodano także strategie dywersyfikacji poszukiwań. Szczegółowy algorytm naszkicowano w [86]. Czas działania TS jest około 20 razy krótszy niż czas działania SA. Pokazano, że metoda TS posiada znacznie lepsze właściwości niż SA.

W pracy [235] zaprezentowano system projektowania Corsair, w którym podział jest wykonywany metodą TS. Optymalizowane jest zużycie zasobów sprzętowych przy ograniczeniach na czas wykonania. Aby zredukować złożoność problemu, użyto trzystopniowej heurystyki dla optymalizacji architektury systemu. Krok pierwszy optymalizuje priorytet grupy węzłów w poszukiwaniu możliwej strategii szeregowania zadań. Krok drugi zmienia odwzorowanie grupy węzłów do elementów przetwarzających, podczas gdy krok trzeci umożliwia zmianę odwzorowania każdego węzła niezależnie. Każdy krok heurystyki posiada własną listę tabu w celu wyeliminowania cykli podczas poszukiwań.

W pracy [272] zaprezentowano metodę TS ze strategią intensyfikacji oraz dywersyfikacji poszukiwań. Minimalizowany jest czas przetwarzania przy ograniczeniu zasobów sprzętowych i uwzględnieniu czasu komunikacji, rekonfiguracji i oczekiwania na zasoby współdzielone. Dywersyfikacja poszukiwań jest oparta o pamięć długookresową, która przechowuje dane o intensywności poszukiwań dla każdego regionu. Intensyfikacja poszukiwań promuje region, w którym znaleziono najlepsze rozwiązanie. Wyniki pokazują, że metoda TS lepiej się skaluje i daje lepsze rozwiązania w krótszym czasie w porównaniu z metodą SA oraz GA.

W pracach [91][92] zastosowano metodę TS, dla której rozwiązanie inicjalne uzyskiwane jest metodą klastrowania CG. Minimalizowany jest czas przetwarzania przy ograniczeniu zużycia zasobów sprzętowych. Względem prac [84][86] zaimplementowana metoda TS przeszukuje jedynie częściowe sąsiedztwo

bieżącego rozwiązania, posiada więcej strategii ewolucyjnych w sytuacji, gdy nie ma dopuszczalnych rozwiązań i stosuje wydajniejsze metody poprawy w przypadku, gdy żadne pojedyncze przesunięcie nie powoduje dalszej optymalizacji funkcji kosztu.

Wydajność metody TS zależy głównie od modelowania. Dostrajanie parametrów nie zrównoważy złego wyboru struktur sąsiedztwa ani funkcji celu. W każdym kroku metody TS jest obliczanych wiele rozwiązań, ważne jest zatem wykonanie obliczeń w sposób wydajny. Często stosowanym sposobem jest wyznaczanie różnicy wartości funkcji celu zamiast bezpośredniej wartości funkcji celu. Rozmiar listy tabu ma również istotne znaczenie: zbyt mała lista nie zapobiegnie cyklom, zbyt duża wprowadza nadmierną liczbę ograniczeń. Za pomocą pamięci średniookresowej intensyfikuje się poszukiwania w najbardziej obiecującym regionie i dywersyfikuje poszukiwań w nieprzebadanych jeszcze regionach.

3.7. Algorytm Zachłanny

Algorytm zachłanny (ang. *greedy*) jest algorytmem heurystycznym, który dokonuje lokalnie optymalnych wyborów na każdym etapie w nadziei znalezienia globalnego optimum. Algorytm wymaga zdefiniowania 5 podstawowych składników:

- zbioru kandydatów, z których zbudowane jest rozwiązanie,
- funkcja selekcji, która wybiera najlepszego kandydata,
- funkcja dopuszczalności, która określa, czy kandydat może być dodany do rozwiązania,
- funkcja celu, która przydziela wartość danemu rozwiązaniu,
- funkcja rozwiązania, która określa, kiedy zostało znalezione kompletne rozwiązanie.

Algorytm 7. Schemat algorytmu zachłannego

1. Nadanie wartości wszystkim implementacjom wszystkich zadań zgodnie z funkcją selekcji. Funkcja selekcji zazwyczaj uwzględnia wartość funkcji celu dla pojedynczej implementacji zadania oraz przekraczanie ograniczeń.
2. Dla wszystkich implementacji ze zbioru dokonuje się wyboru implementacji za pomocą funkcji selekcji i ocena dopuszczalności rozwiązania. Jeśli rozwiązanie jest niedopuszczalne z uwagi na przekroczenie ograniczeń, wówczas jest odrzucane i usuwane ze zbioru. Jeśli rozwiązanie jest dopuszczalne, wówczas jest akceptowane i pozostałe implementacje danego zadania są usuwane ze zbioru.
3. Jeśli otrzymano niekompletne rozwiązanie (funkcja rozwiązania), wówczas dodawane są implementacje brakujących zadań bez oceny dopuszczalności rozwiązania. W kolejnym kroku dokonuje się zamiany implementacji zadań, aby spełnić ograniczenia przy jak najmniejszym pogorszeniu funkcji celu.
4. Po otrzymaniu kompletnego rozwiązania dopuszczalnego można zastosować krok iteracyjnej poprawy, akceptując tylko rozwiązania dopuszczalne zmniejszające funkcję celu.

Podejmowany jest wybór, który wydaje się być najlepszy w danym kroku, a który może zależeć od dotychczas wykonanych wyborów (ale nie od przyszłych). Algorytm iteracyjnie dokonuje zachłannych wyborów redukując każdy dany problem do mniejszego. Algorytm zachłanny nadaje się dobrze dla problemów dyskretnych, które wykazują optymalne podstruktury (ang. *optimal substructure*). Jednak z reguły nie znajduje globalnego optimum, gdyż nie operuje na wyczerpującej ilości danych i zbyt

wcześnie podejmuje wybory, które uniemożliwiają znalezienie najlepszego rozwiązania w przyszłości. W niektórych przypadkach następuje proces iteracyjnej poprawy, akceptujący tylko modyfikacje zmniejszające koszt. Algorytm zatrzymuje się w pierwszym lokalnym minimum. Schemat metody przedstawia Algorytm 7.

Przykładem algorytmu zachłannego jest przydzielenie wszystkich zadań do HW lub SW i iteracyjne ich przenoszenie do SW lub HW w kolejności zdeterminowanej określoną metryką [111][260][261].

W pracy [24] zaprezentowano prosty algorytm zachłanny, który przydziela zadania do HW w kolejności najkrótszych czasów wykonania aż do wyczerpania zasobów sprzętowych. W pracach [53][54] opisano algorytm heurystyczny, który przydziela moduły według ich metryk i dokonuje iteracyjnej poprawy z przesuwaniem HW → SW lub SW → HW. W pracy [109] przedstawiono metodę alokacji zasobów sprzętowych do optymalnego podziału. Metoda analizuje moduł po module i przydziela do HW lub SW. W pracy [111] stosuje się iteracyjną poprawę i ekstrahuje SW bloki z inicjalnego rozwiązania, gdzie wszystkie moduły są zaimplementowane w HW. Ograniczeniem jest czas wykonania, funkcja celu minimalizuje czas wykonania CPU, czas komunikacji i zużycie zasobów sprzętowych. W pracach [22][23] opisano metodę podziału z możliwością ręcznej interwencji użytkownika.

W pracy [280] zaprezentowano gradientową metodę poszukiwań. W każdej iteracji wykonuje się próbę relokacji jednego procesu z jednego CPU do innego, przeniesienie komunikacji z jednej magistrali do innej i obliczenie zmiany kosztów. Jeśli wykrywa się przekroczenie ograniczeń, dodaje się zasoby. Algorytm jednak z powodu swojej zachłanności nie jest optymalny.

W pracy [56] opisano algorytm zachłanny, w którym użyto metody szeregowania zadań ALAP i wsteczną ewaluację długości krytycznej ścieżki. Metoda ALAP operuje wstecz grafu, rozpoczynając od zadań terminalnych i wybiera najbardziej krytyczne czasowo zadanie. Zadanie jest alokowane i szeregowane do kontekstu, który minimalizuje różnicę pomiędzy czasem zakończenia zadania i najwcześniejszym czasem rozpoczęcia inicjalnego zadania w grafie. Odległości od każdego zadania do inicjalnych zadań są obliczane przy użyciu algorytmu rekursywnej ewaluacji długości ścieżki krytycznej. Iteracyjny proces podziału optymalizuje lokalnie alokację i szeregowanie zadań.

Algorytm zachłanny jest metodą dającą stosunkowo dobre wyniki przy niezbyt ścisłych ograniczeniach i funkcji celu składającej się tylko z lokalnych metryk. Jeśli narzucone zostaną ścisłe ograniczenia na wiele parametrów, to algorytm zachłanny po osiągnięciu pierwszego z ograniczeń może podejmować bardzo nieefektywne decyzje i nadmiernie zwiększając wartość funkcji celu. Jeśli funkcja celu będzie zdeterminowana przez czas wykonania, co wymaga wyznaczenia czasu rozpoczęcia i zakończenia dla wszystkich zadań, wówczas algorytm zachłanny nie jest w stanie przewidzieć wpływu pojedynczej decyzji na wartość globalnej metryki. W rezultacie może wygenerować rozwiązanie składające się z bardzo efektywnej realizacji poszczególnych zadań, ale niezsynchronizowanych ze sobą i nieefektywnie się komunikujących. Algorytm zachłanny w sposób naturalny modeluje rozszerzony problem podziału.

3.8. Logika rozmyta FL

Logika rozmyta jest rozszerzeniem logiki Boole'a o logikę wielowartościową, włączającym pojęcie prawdy częściowej, znajdującej się pomiędzy zupełną prawdą i zupełnym fałszem. Została wprowadzona przez Lotfi Zadeha [281] jako narzędzie modelowania nieokreśloności, wieloznaczności i braku precyzji występujących w

języku naturalnym [132], naśladując wnioskowanie przeprowadzane przez człowieka. W teorii logiki rozmytej definiuje się podzbiory rozmyte, operacje na zbiorach rozmytych oraz wnioskowanie rozmyte, które przedstawia Algorytm 8.

Algorytm 8. Schemat wnioskowania rozmytego

1. Rozmywanie (fuzyfikacja) – proces przyporządkowania liczbom wejściowym wartości funkcji przynależności (np. klasy s, π, γ, t, L) i normalizacji [213].
2. Zastosowanie operatorów rozmytych – proces przeprowadzenia operacji logicznych na przesłankach zgodnie z przyjętą definicją operatorów rozmytych.
3. Zastosowanie implikacji *modus ponens* (rzadziej *modus tollens*) – obliczanie funkcji przynależności wniosku w każdej z reguł, np. metodą minimum (Mamdaniego), iloczynowej (Larsena), Łukasiewicza, max-min (Zadeha), lub innej [213].
4. Kompozycja – obliczanie funkcji przynależności wniosku, np. metodą kompozycji MAX lub kompozycji sumacyjnej.
5. Precyzowanie (defuzyfikacja) – proces otrzymywania jednej wartości liczbowej, np. przez wyznaczenie środka ciężkości funkcji przynależności według zależności (12).

Wyznaczenie środka ciężkości funkcji przynależności można wyznaczyć następująco [132][213]:

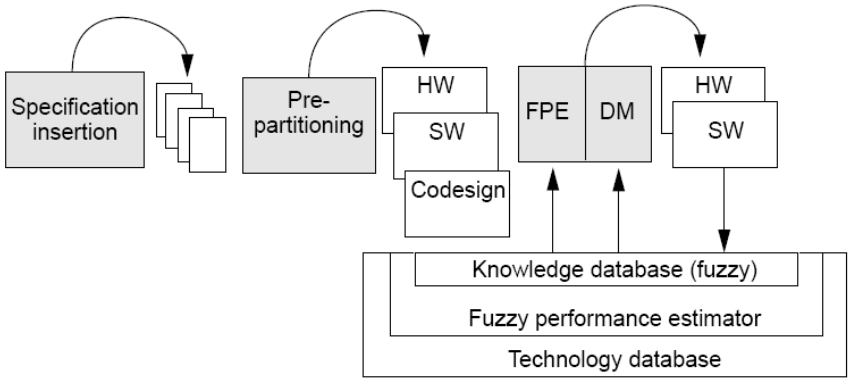
$$s_c = \frac{\int_a^b xs(x)dx}{\int_a^b s(x)dx} \tag{12}$$

gdzie:

s_c – środek ciężkości,
a,b – wartości graniczne dziedziny funkcji przynależności,
s(x) – funkcja przynależności.

Zastosowanie logiki rozmytej do procesu podziału wymaga utworzenia systemu regułowego, w którym działa logika rozmyta. System regułowy może być wyspecyfikowany przez projektanta, może być także pozyskany w procesie zdobywania wiedzy przez system ekspertowy [189] lub sieć neuronową [213].

W pracy [49] zastosowano logikę rozmytą do estymacji kosztów i do problemu podziału wraz z systemem ekspertowym do ekstrakcji parametrów, co przedstawia Rysunek 4. Podział wstępny dzieli moduły na grupy sprzętową, programową i codesign. Moduły codesign są następnie charakteryzowane przez rozmyty estymator wydajności FPE, który korzysta z bazy wiedzy systemu ekspertowego utworzonej w fazie nauki. System ekspertowy przechowuje wiedzę o zależności kosztu implementacji od wybranych atrybutów specyfikacji, które projektant uważa za najważniejsze podczas ewaluacji kodu. Kluczowym problemem rozwiązywanym przez logikę rozmytą jest wyrażenie relacji pomiędzy przekazanymi danymi i parametrami niezbędnymi do oszacowania kosztu implementacji sprzętowej lub programowej i ustalenie wagi każdego parametru w tej relacji. Wymaga to sformułowania rozmytych reguł wnioskowania. Koszt każdego modułu jest również wyrażony w sposób rozmyty. Innym zadaniem realizowanym przez logikę rozmytą jest podejmowanie decyzji o implementacjach i poszukiwanie nowych rozwiązań. Jeśli dane rozwiązanie nie spełnia wymagań, wówczas znajdowane jest nowe.

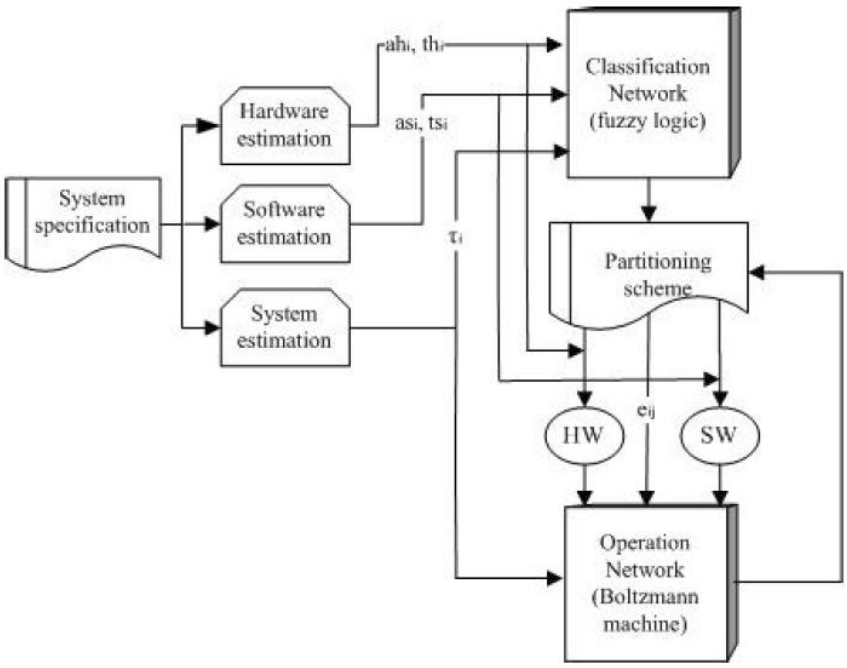


Rysunek 4. Schemat działania systemu rozmytego w problemie podziału [49]

W pracach [168][170] zaprezentowano system ekspertowy SHAPES, który używa logiki rozmytej do heurystycznej klasyfikacji węzłów. SHAPES przeprowadza podział na sprzęt i oprogramowanie w 4 krokach:

- dopasowanie – generuje rozwiązanie inicjalne na podstawie analizy poszczególnych zadań, wykonywane przez moduł heurystycznej klasyfikacji;
- przydział – uwzględnia ograniczenia (zużycie zasobów i czas wykonania) i generuje wartość progu określającą granicę sprzętowo-programową;
- ewaluacja – ocenia jakość wygenerowanego rozwiązania: zużycie zasobów, czas wykonania wyznaczając kolejkowanie oraz koszt komunikacji;
- wybór – analizuje wygenerowane rozwiązanie w oparciu o diagnostykę symptomów przy użyciu metryk i proponuje nowe rozwiązanie przez wymianę zadań pomiędzy sprzętem i oprogramowaniem.

Estymaty parametrów zużycia zasobów, różnicy czasów wykonania pomiędzy realizacjami programowymi i sprzętowymi oraz liczba wykonań są konwertowane do zmiennych rozmytych, a następnie stosowane są rozmyte reguły wnioskowania. Reguły wnioskowania pochodzą z bazy wiedzy uaktualnianej interakcyjnie przez projektanta.



Rysunek 5. Architektura hybrydowego systemu neuronowo-rozmytego [121][122]

W pracach [121][122] zaproponowano hybrydowy system neuronowo-rozmyty zawierający maszynę Boltzmanna do problemu podziału. Rysunek 5 przedstawia system składający się z sieci klasyfikującej wykorzystującej logikę rozmytą i sieci operacyjnej generującej optymalny podział dla zadanych ograniczeń. Klasyfikator zrealizowano w MATLABie przy użyciu sigmoidalnych funkcji przynależności. Rozmyte reguły wnioskowania nie są sformułowane przez projektanta, lecz są implementowane przez 5-warstwową sieć neuronową, która posiada możliwość uczenia się. Uczenie się sieci neuronowej jest przyspieszone dzięki zainicjalizowaniu wag sieci na podstawie bazy wiedzy.

Zaletą logiki rozmytej jest odwzorowanie subiektywności, nieprecyzyjności i niepewności wyborów dokonywanych przez inżynierów. Kluczem do logiki rozmytej jest odwzorowanie przestrzeni wejściowej na przestrzeń wyjściową pod kontrolą listy reguł warunkowych [122]. Dane wejściowe są z reguły estymatami, które mogą być łatwo modelowane za pomocą rozmytych zmiennych. Reguły pochodzące z domeny ekspertowej są lingwistyczne i niedokładne, stąd definiuje się funkcje odwzorowujące reguły na teorię zbiorów rozmytych. Ograniczenia również są modelowane za pomocą zmiennych rozmytych, co utrudnia modelowanie sztywnych ograniczeń. Obliczenia związane z logiką rozmytą są wykonywane niskim kosztem i w krótkim czasie. Rozwiązywany jest zazwyczaj binarny problem podziału, a logika rozmyta jest klasyfikatorem dla systemów opartych o sieci neuronowe i systemy ekspertowe.

3.9. Sieci Bayesa BBN

Sieci Bayesa reprezentują za pomocą grafu DAG zmienne losowe (wnioski w systemie) i warunkowe zależności pomiędzy nimi [132]. Z każdym wierzchołkiem są stowarzyszone funkcje prawdopodobieństwa zależne od zmiennych losowych wierzchołków poprzedzających (przyczynowych). Gęstość prawdopodobieństwa jest zdefiniowana przez:

$$p(x) = \prod_{v \in V} p\left(x_v | x_{pa(v)}\right) \tag{13}$$

gdzie:

- p(x) – łączna gęstość prawdopodobieństwa,
- v – wierzchołek,
- V – zbiór wierzchołków,
- x_v – zmienna losowa indeksowana przez v,
- pa(v) – zbiór wierzchołków poprzedzających wierzchołek v.

Podczas propagacji dowodów (ang. *belief propagation*) przez sieć BBN używa się dwa rodzaje wnioskowań [132]:

- diagnostyczne – od skutków do przyczyn – przekazywanie komunikatu λ;
- przyczynowe – od przyczyn do skutków – przekazywanie komunikatu π.

Prawdopodobieństwo brzegowe hipotez dla wierzchołków nieobserwowanych jest obliczane z prawdopodobieństwa warunkowego dla wierzchołków obserwowanych.

Algorytm 9. Schemat propagacji dowodów w sieciach Bayesa

1. Inicjalizacja: generacja sieci BBN, ustalenie początkowego prawdopodobieństwa hipotez według zależności (13).
2. Pozyskanie dowodów na drodze obserwacji lub symulacji.
3. Propagacja dowodów dla wierzchołków obserwowanych przez sieć BBN i wyznaczanie prawdopodobieństwa brzegowego hipotez dla nieobserwowanych wierzchołków.

Algorytm 9 powtarza kroki 2-3 w sposób iteracyjny, aż do osiągnięcia wartości pożądanych parametrów i prawdopodobieństwa hipotez.

W pracach [195][196] użyto sieci Bayesa [132] do klasyfikacji komponentów na realizacje sprzętowe i programowe. Sieci Bayesa reprezentują probabilistyczny model opisu funkcjonalnego i umożliwiają propagację lokalnych wyborów oraz ich wpływ na całą sieć. Metoda podziału przebiega w 4 krokach: generacja sieci Bayesa, transformacja wyników symulacji systemu na dowody (złożoność, przepustowość, częstotliwość wykonania), propagacja dowodów przez sieć Bayesa i klasyfikacja komponentów funkcjonalnych na realizacje sprzętowe i programowe. Inicjalnie do każdego węzła sieci Bayesa jest przypisane prawdopodobieństwo 50% realizacji sprzętowej i 50% realizacji programowej. W miarę propagacji dowodów przez sieć Bayesa uaktualniane są wartości prawdopodobieństwa, a po stabilizacji sieci dokonywana jest klasyfikacja.

3.10. Heurystyka KLFM

Heurystyka KL została zaproponowana przez Kernighana i Lina [141] do rozwiązania problemu wyznaczenia wartości minimalnego rozcięcia (ang. *min-cut*). Heurystyka jest klasycznym podejściem iteracyjnej poprawy i umożliwia procesowi optymalizacji na wyjście z minimum lokalnego. Algorytm dąży do znalezienia optymalnego podziału zbioru wierzchołków ***V*** na 2 równoliczne rozłączne partycje ***A*** i ***B***, minimalizując sumę wag ***T*** krawędzi pomiędzy partycjami. Algorytm rozpoczyna z przypadkowym przydziałem zadań do równolicznych partycji i migruje *k* spośród |***V***|*/2* par zadań pomiędzy partycjami. Umożliwia w ten sposób wyjście z lokalnego minimum, gdzie pojedyncza wymiana nie poprawia już kosztu, ale ich sekwencja znajduje lepsze rozwiązanie. Algorytm dokonuje wymiany według zmian funkcji kosztu (ang. *gain*), $\boldsymbol{T_{old}}$-$\boldsymbol{T_{new}}$, zdefiniowanej jako suma wag krawędzi zbioru rozcięcia.

$$D_a = E_a - I_a \tag{14}$$

$$g = T_{old} - T_{new} = D_a + D_b - 2c_{a,b} \tag{15}$$

gdzie:

- D_a – różnica kosztów zewnętrznych i wewnętrznych dla zbioru A,
- E_a – koszty zewnętrzne,
- I_a – koszty wewnętrzne,
- g – zmiana funkcji kosztu (ang. *gain*),
- T_{old} – funkcja kosztu przed zmianą,
- T_{new} – funkcja kosztu po zmianie,
- $C_{a,b}$ – koszt możliwych krawędzi pomiędzy partycjami.

Algorytm 10. Schemat heurystyki Kernighana-Lina

1. Inicjalizacja: znajdowany jest zbalansowany inicjalny podział na dwa rozłączne podzbiory ***A*** i ***B***.
2. Obliczenie różnicy D pomiędzy kosztami zewnętrznymi i wewnętrznymi dla każdego wierzchołka w zbiorze ***A*** i ***B*** według zależności (14).
3. Znajdowanie |***V***|*/2* par wymian: a) znajdowanie pary wierzchołków a_i i b_i, maksymalizujących wartość g_i; b) zamiana wierzchołków a_i i b_i; c) usunięcie wierzchołków a_i i b_i z rozważanego zbioru; d) aktualizacja wartości *D* dla zbiorów $\boldsymbol{A=A\backslash a_i}$ oraz $\boldsymbol{B=B\backslash b_i}$.
4. Akceptacja sekwencji k wymian, które poprawiają koszt według zależności (15).

Rozwiązanie uzyskane w danej iteracji jest używane jako punkt startowy w kolejnej iteracji. Algorytm 10 wykonuje kroki 2-4 w sposób iteracyjny i kończy się, gdy w danej iteracji nie uzyskano już żadnej poprawy.

Heurystyka FM została zaproponowana w 1982 przez Fiduccia i Mattheysesa [94] i dodała rozszerzenia do metody KL:

- rozszerzono metrykę rozcięć dla hipergrafów, umożliwiając podział na więcej niż 2 partycje (również niezbalansowane);
- dodano wagi wierzchołków;
- umożliwiono pojedyncze przesunięcia (ang. *move*) pomiędzy partycjami, zamiast tylko zamiany (ang. *swap*), co zredukowało złożoność obliczeniową;
- dodano tablicę zmian funkcji kosztu (ang. *gain bucket array*), grupującą węzły o takich samych wartościach zmiany funkcji, zapewniającą liniowy czas dostępu przy określaniu najlepszego przesunięcia.

Algorytm dokonuje priorytetyzowania przesunięć według zmian funkcji kosztu. Algorytm kończy się, gdy kolejna iteracja nie dokonała już poprawy wartości funkcji kosztu zdefiniowanego zależnością (15). FM nie jest metodą zachłanną i nie zatrzymuje się w lokalnych minimach, gdyż zmiana funkcji kosztu może być ujemna.

Heurystyka KL oraz FM zapoczątkowała projektowanie układów VLSI. Obecnie w projektowaniu VLSI większość istniejących algorytmów optymalizacji bazuje na iteracyjnych przesunięciach używając podejścia FM [200].

W pracach [262][263] zastosowano heurystykę KLFM do rozwiązania problemu podziału HW/SW oraz HW/HW, dokonując modyfikacji klasycznego algorytmu minimalizacji rozcięcia. Koszt rozcięcia zastąpiono przez koszt czasu wykonania zadania, wyznaczany przez sumę czasu obliczeń i czasu komunikacji z innymi zadaniami. Zamiany parami, które zapewniały utrzymanie zbalansowanego digrafu, zastąpiono definicją pojedynczych przesunięć, które lepiej odpowiadają specyfice podziału funkcjonalności. Dokonano efektywnej implementacji tablicy zmian kosztu, propagując zmiany wynikające z pojedynczego przesunięcia, uzyskując złożoność O(logn). Zaproponowano rozszerzenie funkcji celu na inne metryki oraz umożliwiono modelowanie ograniczeń za pomocą funkcji celu. Zmodyfikowano kryteria stopu, które umożliwiają otrzymanie dobrego rozwiązania w relatywnie krótkim czasie.

W pracy [146] użyto metody KLFM do rozwiązania problemu podziału na dwie partycje sprzętową i programową, minimalizując zużycie zasobów przy spełnieniu ograniczeń czasowych. Największy nacisk położono na modelowanie na wysokim poziomie i użycie wysokopoziomowych metryk bezpośrednio do problemu podziału.

W pracy [200] do rozwiązania problemu podziału HW/SW użyto wielopoziomowej heurystyki MLFM, która składa się z trzech komponentów: klastrowania, podziału i poprawy. Podział jest wykonywany heurystyką FM, która w każdym przebiegu przeszukuje sąsiedztwo danego podziału, znajduje najlepsze rozwiązanie i akceptuje je, uaktualniając informacje o koszcie danego rozwiązania. Również informacje o koszcie sąsiednich wierzchołków muszą zostać uaktualnione. Heurystyka MLFM została zaimplementowana w narzędziach MLPart oraz hMETIS.

Praca [176] stanowi rozszerzenie prac [262][263] o analizę funkcji zmiany kosztu, generalizację funkcji celu o metryki addytywne i aspekty implementacyjnych tablicy zmiany kosztu. Uzyskano przyspieszenie realizacji heurystyki KLFM, wskazano jednak szereg ograniczeń metody – m.in. trudności w uwzględnieniu kolejkowania i współdzielenia zasobów.

Heurystyka KL jest bardzo wrażliwa na inicjalną konfigurację, wskutek czego dla ścisłych ograniczeń parametrów nie otrzymuje się dopuszczalnych rozwiązań. Funkcja celu algorytmu podziału funkcjonalności jest bardziej złożona niż

jednoargumentowa moc zbioru rozcięcia lub suma wag krawędzi zbioru rozcięcia. Obliczanie zmiany funkcji kosztu nie zawsze jest możliwe z powodu złożonych zależności czasowych i często sprowadza się do wyznaczenia wartości funkcji kosztu dla nowej konfiguracji podziału. W wielu implementacjach obliczanie zmiany funkcji kosztu prowadzi do długiego czasu działania algorytmu podziału. W celu przyspieszenia operacji stosuje się tablicę zmian funkcji kosztu, która zapewnia liniowy czas uaktualniania. Aspekty implementacyjne opisują prace [200][262][263].

3.11. Klastrowanie

Klastrowanie jest głównym zadaniem wydobywania wiedzy (ang. *data mining*) i jest procesem przekształcenia grafu DAG w graf o mniejszej ilości wierzchołków poprzez łączenie wierzchołków na podstawie ich kryterium podobieństwa lub bliskości w większe grupy wierzchołków zwane klastrami. Istnieje wiele modeli klastrowania: model połączeń, model centroidu, model dystrybucji, model gęstości, model podprzestrzeni, modele grupowe. Problem podziału grafu na N partycji (ang. *N-way partitioning*) minimalizacji sumy wag przeciętych krawędzi przy ograniczeniu sumy wag wierzchołków w każdej partycji (ang. *balance constraints*) odpowiada problemowi podziału funkcjonalności na N modułów i minimalizacji komunikacji pomiędzy modułami przy ograniczeniach kosztu każdego modułu [200]. Hierarchiczne klastrowanie bazujące na modelu połączeń jest konstruktywnym podejściem do rozwiązania problemu podziału [73], grupującym obiekty na podstawie metryki bliskości pomiędzy obiektami. Popularnymi metrykami są:

- minimum odległości pomiędzy obiektami (ang. *single-linkage clustering*);

$$min\{d(a,b): a \in A, b \in B\} \tag{16}$$

- maximum odległości pomiędzy obiektami (ang. *complete linkage clustering*);

$$max\{d(a,b): a \in A, b \in B\} \tag{17}$$

- średnia odległość pomiędzy obiektami (ang. *average linkage clustering*).

$$\frac{1}{|A||B|}\sum_{a\in A}\sum_{b\in B} d(a,b) \tag{18}$$

gdzie:

d – metryka odległości pomiędzy dwoma elementami,
a,b – elementy,
A,B – klastry.

Algorytm 11. Schemat klastrowania z metryką minimum odległości pomiędzy obiektami

1. Inicjalizacja: klastry są rozłączne, poziom klastra L(0)=0; numer sekwencji m=0.
2. Znajdowanie pary klastrów *a* i *b*, dla których odległość d(a,b) jest minimalna pośród wszystkich klastrów według zależności (16).
3. Inkrementacja numeru sekwencji m=m+1. Połączenie klastrów *a* i *b* w pojedynczy klaster *ab*, poziom klastra L(m)=d(a,b).
4. Uaktualnienie metryk bliskości dla pozostałych klastrów przez wyznaczenie nowych wartości odległości: d(ab,c)=min{d(a,c),d(b,c)}.
5. Zakończenie, jeśli wszystkie obiekty znajdują się w jednym klastrze. W przeciwnym przypadku kontynuacja od punku 2.

W pracy [159] zaprezentowano automatyczny podział architektury w systemie APARTY oparty na wieloetapowym klastrowaniu. W kolejnych iteracjach dokonuje

się klastrowania w oparciu o inne metryki. Zdefiniowano kryteria bliskości stosowane w klastrowaniu:

- kryterium bliskości kontroli – umożliwia grupowanie operatorów w celu redukcji liczby przypadków, w których kontrola jest przekazywana pomiędzy klastrami;
- kryterium bliskości danych – umożliwia grupowanie danych w celu redukcji liczby przypadków, w których dane są przekazywane pomiędzy klastrami;
- kryterium bliskości kolejkowania – umożliwia współużywanie zasobów w celu optymalnego kolejkowania zadań.

W pracy [28] zaprezentowano system UNITY, który realizuje wieloetapowe klastrowanie. Optymalizowany jest czas wykonania, przy zadanych ograniczeniach na wiele parametrów i nacisku na współdzielenie zasobów sprzętowych. Program wyspecyfikowany w systemie UNITY jest klasyfikowany w zależności od alternatyw implementacyjnych i zależności danych (np. relacje asynchroniczne, synchronizm wewnętrzny/zewnętrzny, wzajemna wyłączność), a następnie budowane jest drzewo klastrowania w oparciu o metryki podobieństwa i metryki współdzielenia zasobów. Klastry są alokowane do określonej architektury i wyznaczany jest koszt bieżącego podziału przy jednoczesnej analizie spełnienia ograniczeń.

W pracy [278] zaprezentowano podział w systemie PARAGON, bazowany na systemie UNITY, ale na wyższym poziomie granularności. Proces podziału rozpoczyna się od identyfikacji fragmentów, charakteryzowanych jako intensywnie wykorzystujące CPU, IO lub przepływu kontroli. W kolejnym kroku określane są podobieństwa pomiędzy jednostkami podziału, przez sprawdzenie wspólnych lub powiązanych właściwości i wyznaczana jest metryka odległości. Zadania z podobną charakterystyką, wspólnymi zmiennymi globalnymi, podobnym poziomem współbieżności, podobną hierarchią klas lub na podobnym poziomie drzewa syntaktyki zostaną przydzielone do tego samego procesora. Podobnie jak w systemie UNITY, klastry są alokowane do architektury, wyznaczany jest koszt podziału i analizowane są spełnienia ograniczeń.

W pracy [119] zaprezentowano hierarchiczny algorytm podziału stosowanego w systemie HiPART. Minimalizowany jest czas wykonania systemu przy ograniczeniach na wiele parametrów. Algorytm składa się z trzech etapów metod konstrukcyjnych i iteracyjnych. W pierwszym etapie grupowane są w klastry ściśle powiązane obiekty, minimalizując komunikację pomiędzy partycjami na podstawie metryk czasu wykonania. W drugim kroku w każdym z analizowanych klastrów przeprowadzany jest drobnoziarnisty podział za pomocą metody SA. W ostatnim kroku dokonuje się dostrajania na poziomie instrukcji przy użyciu heurystyki KLFM.

W pracach [169][170] opisano klastrowanie hierarchiczne grupujące obiekty na podstawie metryki bliskości pomiędzy obiektami, zaimplementowaną odmiennie dla implementacji sprzętowych i programowych. Inicjalnie wszystkie obiekty są realizowane programowo i są iteracyjnie przenoszone do sprzętu. Metryka bliskości została zdefiniowana jako ważona suma krawędzi i przyspieszenia spowodowanego przesunięciem zadania do sprzętu. W każdej iteracji algorytm wybiera dwa obiekty, dla których uzyskuje się największą wartość funkcji bliskości. Podczas wzrostu klastra sprawdzane są zużycie zasobów sprzętowych i czas wykonania. Jeśli jeden z klastrów wzrośnie ponad aktywne ograniczenie zużycia zasobów sprzętowych, jest zaznaczany jako skończony.

W pracy [207] opisano klastrowanie hierarchiczne do problemu podziału funkcjonalności dla architektury wieloprocesorowej, minimalizujące czas wykonania. Podział jest dokonywany na dwóch poziomach: systemowym i funkcjonalnym. Podział na poziomie systemowym dzieli zadania w homogeniczne klastry

minimalizując komunikację pomiędzy klastrami, przenosi współpracujące zadania pomiędzy klastrami i dostarcza informacji o koniecznym przyspieszeniu do kolejnego etapu. Podział na poziomie funkcjonalnym jest wykonywany metaheurystyką GA, SA lub TS.

W pracy [24] drobnoziarniste klastrowanie jest używane do budowy bloków podstawowych do późniejszego podziału. Optymalizowany jest czas wykonania przy ograniczeniach zasobów sprzętowych oraz pamięci RAM i ROM. Klastrowanie polega na konstrukcji maksymalnych regionów (ang. *clubs*), które mogą być implementowane jako pojedyncza instrukcja bez naruszania zależności danych. Klaster jest blokiem funkcjonalności do potencjalnej implementacji sprzętowej. Węzeł może być dodany do istniejącego klastra, jeśli nie zawiera pierwotnych wejść, zawiera węzły kontrolne lub danych i nie wprowadza cykli kombinacyjnych. Jeśli któryś z warunków nie jest spełniony, wówczas tworzony jest nowy klaster.

W pracy [56] opisywany jest dynamiczny podział funkcjonalności na sprzęt i oprogramowanie z przełączaniem kontekstów sprzętowych. Podział statyczny optymalizuje czas wykonania, przez zmniejszanie długości najdłuższej ścieżki i jest wyznaczany algorytmem GA. W dla każdego osobnika w algorytmie GA jest wykonywane klastrowanie w celu budowy kontekstów dla dynamicznej zmiany zasobów w systemach rekonfigurowalnych. Wszystkim zadaniom nadawane są priorytety, równe najdłuższej ścieżce od danego zadania do ostatniego zadania. Następnie przydziela się zadania o najwyższym priorytecie do klastra. Jeśli klaster przekracza dostępną ilość zasobów sprzętowych, wówczas tworzony jest nowy klaster.

W pracy [273] opisano dwustopniowe klastrowanie gruboziarniste, które jest używane jako krok wstępny przed właściwym podziałem wykonywanym metodą TS. Minimalizowany jest czas komunikacji, czas wykonania wszystkich zadań i ścieżka krytyczna, przy ograniczeniach innych parametrów. W pierwszej fazie minimalizowana jest ścieżka krytyczna za pomocą algorytmu TS, łącząc zadania w pary dla najlepszej funkcji dopasowania, zezwalając na akceptację gorszych rozwiązań omijając lokalne minima. W drugiej fazie redukowany jest koszt systemu przez analizę węzłów leżących poza ścieżką krytyczną.

W pracy [202] przedstawiono użycie algorytmu hierarchicznego klastrowania do syntezy pamięci i architektury magistral. Dokonano minimalizacji liczby magistral przy spełnieniu założeń wydajnościowych i minimalizacji zużytych zasobów.

Klastrowanie hierarchiczne jest stosowane do szybkiej eksploracji przestrzeni projektowej i jest zazwyczaj pierwszym etapem zwiększającym granularność zadań i zmniejszającym ilość komunikacji pomiędzy klastrami. Podział gruboziarnisty może zwiększyć wydajność implementacji przez wzrost równoległości zadań. Po pierwszym etapie klastrowania rozmiar grafu zadań jest znacznie zredukowany, co przyspiesza proces dalszego podziału, kolejkowania zadań i syntezy. Wyniki w pracach [169][170] pokazały, że metoda klastrowania może być skutecznie stosowana do problemów o rozmiarze 500 węzłów.

3.12. Heurystyka GCLP

Heurystyka GCLP jest używana w systemie Ptolemy [138][139][174]. GCLP rozwiązuje binarny i rozszerzony problem podziału, do którego używa heurystyki wyboru implementacji IBS. GCLP minimalizuje zużycie zasobów sprzętowych przy ograniczeniu czasu przetwarzania. Heurystyka GCLP zastosowana do rozwiązania podziału binarnego ma złożoność $O(N^2)$, natomiast heurystyka MIBS w przypadku rozwiązania podziału rozszerzonego ma złożoność $O(N^3 + N^2B)$. GCLP bazuje na

algorytmach szeregowania list i redukuje zachłanność algorytmów szeregowania zadań przez wprowadzenie metryki GC.

W każdym kroku algorytmu jest obliczana metryka globalnej krytyczności czasu GC (prawdopodobieństwa implementacji sprzętowej), zapobiegająca przekroczeniom ograniczeń czasowych:

$$GC = \frac{\sum_{N_H} area_i}{\sum_{N_U} area_i} \tag{19}$$

gdzie:

- GC – wartość metryki krytyczności,
- N_H – zbiór węzłów przeznaczonych do realizacji sprzętowej,
- N_U – zbiór węzłów nieprzydzielonych,
- $area_i$ – zużycie zasobów sprzętowych przez węzeł i.

Zbiór N_H jest wybierany na podstawie funkcji selekcji według malejącej wartości czasu wykonania SW, przyspieszenia czasu wykonania SW względem HW lub w kolejności rosnącej powierzchni HW. Na podstawie kryterium pilności przydzielany jest jeden węzeł, którego poprzedniki w grafie DAG zostały zaalokowane. Następnie wybierana jest adaptacyjnie odpowiednia funkcja celu podziału, porównując globalną estymatę krytyczności czasu z lokalnie modyfikowaną wartością progu GC. Funkcja celu Obj1 minimalizuje czas wykonania, wybierając implementację sprzętową, jeśli w danym momencie czas jest krytyczny. Funkcja celu Obj2 minimalizuje zużycie zasobów, wybierając realizację programową, jeśli w danym momencie krytyczne staje się zużycie zasobów.

Algorytm 12. Schemat heurystyki GCLP

1. Inicjalizacja: N_U=N; N_M=0; N_U – zbiór węzłów nieprzydzielonych, N_M – zbiór węzłów przydzielonych, N – rozmiar grafu DAG.
2. Obliczenie globalnej krytyczności GC: znalezienie w zbiorze N_U minimalnego podzbioru N_H węzłów przeznaczonych do realizacji sprzętowej, wyznaczenie wartości metryki GC zgodnie z zależnością (19) i czasu zakończenia.
3. Wybór wierzchołka: wyznaczenie zbioru N_R wierzchołków gotowych (których poprzedniki w DAG zostały już zaalokowane), dla każdego wierzchołka gotowego wyznaczenie czasu jego wykonania (uwzględniając wartość GC dla wierzchołków ze zbioru N_U) i obliczenie najdłuższej ścieżki w grafie DAG, wybierany jest wierzchołek {i} o największej wartości najdłuższej ścieżki w grafie DAG.
4. Faza lokalna LP: klasyfikacja wierzchołka do jednej z trzech klas i wyznaczenie wartości progu Th=0.5+Δ.
5. Wybór funkcji celu: porównanie wartości GC z wyznaczoną wartością progu Th i wybór funkcji Obj1 lub Obj2.
6. Wybór implementacji ze zbioru: zgodnie z funkcją celu dokonuje się wyboru implementacji sprzętowej lub programowej i wyznacza czas rozpoczęcia zadania.
7. Aktualizacja stanu: N_M=N_MU{i}; N_U=N_U\{i}; aktualizacja wartości pozostałego czasu.

Algorytm 12 wykonuje kroki 2-7 w sposób iteracyjny i kończy się, gdy przydzielono już wszystkie węzły: N_U=0 i N_M=N.

Faza lokalna LP jest metodą klasyfikacji węzłów opartą o ich heterogeniczność i wbudowane właściwości. Każdy węzeł może być sklasyfikowany jako:

- węzeł zużywający nieproporcjonalnie więcej zasobów w jednej realizacji w porównaniu do drugiej (ang. *extremity*), wartość metryki nieproporcjonalności Δ modyfikuje wartość progu GC;
- węzeł odpychający (ang. *repeller*), używany do wymiany pomiędzy HW i SW w trakcie działania algorytmu, wartość metryki odpychania Δ modyfikuje wartość progu GC;
- węzeł normalny (ang. *normal*) – nie modyfikuje wartości progu GC.

Metryka LP kwantyfikuje preferencje lokalnego przydziału i modyfikuje wartość progu GC.

W rozszerzonym problemie każdy węzeł jest charakteryzowany przez 3 atrybuty: odwzorowanie, zbiór implementacji, kolejkowanie, może zatem znajdować się w jednym z 3 stanów:

- wolnym (ang. *free*) – niewybrany do implementacji;
- etykietowanym (ang. *tagged*) – wybrany do implementacji;
- ustalonym (ang. *fixed*) – ustalona implementacja.

Kluczową ideą jest użycie estymat do korelacji zbioru implementacji dla węzła zaetykietowanego z powierzchnią HW wymaganą dla węzła wolnego. Wybierany jest najbardziej odpowiedni zbiór pod tym względem jako zbiór implementacji dla danego węzła. Definiuje się przy tym krzywą BFC jako zbiór wszystkich wartości częściowych dla zbioru implementacji dla danego zaetykietowanego węzła. Definiowana jest również wrażliwość zbioru implementacji jako gradient BFC, co odzwierciedla zmianę wartości częściowych dla zbioru implementacji na zmiany zbioru w węźle. Algorytm 13 przedstawia schemat heurystyki MIBS.

Algorytm 13. Schemat heurystyki MIBS

1. Inicjalizacja: $N_{FREE}=N$; $N_{FIXED}=0$; N_{FREE} – zbiór węzłów wolnych, N_{FIXED} – zbiór węzłów ustalonych, N – rozmiar grafu DAG.
2. Wyznaczenie odwzorowania i kolejkowania dla węzłów wolnych N_{FREE} za pomocą GCLP, przy założeniu median wartości parametrów zasobów i czasu.
3. Wybór wierzchołka {i} z odwzorowaniem M_I.
4. Wyznaczenie implementacji dla wierzchołka {i} z odwzorowaniem M_I.
5. Aktualizacja stanu: $N_{FIXED}=N_{FIXED}U\{i\}$; $N_{FREE}=N_{FREE}\backslash\{i\}$; aktualizacja kolejkowania.

Zaletą metody GCLP jest jej stosunkowo szybkie działanie, co sprawia, że może rozwiązywać problemy modelowane grafami DAG z 500 wierzchołkami. Autorzy podawali [139], że generowane rozwiązanie dla małych grafów było nie gorsze niż 30% od optimum, choć jednocześnie nie podali dowodu alfa-optymalności, co dopuszcza uzyskanie dowolnie oddalonego rozwiązania. Sporym ograniczeniem metody jest brak możliwości stosowania dowolnych funkcji celu oraz ograniczeń większej ilości parametrów. Heurystyka GCLP jest bardzo skomplikowana do implementacji, a jej aspekty implementacyjne opisuje szczegółowo praca [139].

3.13. Inne metody

W pracy [22] opisano system projektowania Tosca, który używa modelu obliczeń opartego o algebrę procesów i rozszerzoną składnię Occam II. Metoda podziału polega na wykonywaniu formalnych transformacji reprezentacji algebry procesów.

W pracy [267] opisano algorytm podziału metodą kolonii mrówek (ang. *ant colony*) [132]. Metoda kolonii mrówek bazuje na użyciu wielu agentów do przeszukiwania przestrzeni rozwiązań i znalezieniu lokalnie produktywnych

obszarów. Metoda stosowana w przypadkach, gdy nie ma globalnej perspektywy rozwiązania problemu.

W pracy [264] przedstawiono algorytm podziału oparty o analizę wymagań niezawodności i bezpieczeństwa systemu. Niezawodność została zdefiniowana jako „możliwość wykonania przez system wymaganych funkcji w określonych warunkach, w danym zakresie, w określonym okresie czasu, i jeśli system nie może wykonać funkcji w sposób prawidłowy, przechodzi do ściśle określonego bezpiecznego stanu".

W pracach [3][165] opisano algorytm podziału oparty o koszt jego testowania. W rezultacie otrzymuje się system o minimalnym koszcie testowania – najmniejszej liczbie wektorów testowych niezbędnych do wyczerpującego przetestowania implementacji sprzętowych i programowych, lub też najmniejszej liczbie przypadków potrzebnych do przetestowania systemu. Koszty testowania poszczególnych modułów są estymowane za pomocą analizy mutacji. Mutację określa się jako niedziałającą wersję różniącą się pojedynczą prostą zmianą.

3.14. Podsumowanie

W literaturze spotyka się wiele prób porównań metod stosowanych do rozwiązania problemu podziału. Najczęściej skupiają się one na określeniu jakości generowanych rozwiązań i czasie obliczeń. W pracy [17] porównano metody TS, SA i GA, wykazując, że niezależnie od stosowanego schematu kolejkowania, najlepszą metodą jest TS, potem SA, na końcu zaś GA. W pracy [272] porównano metodę GA z TS i SA. Wyniki pokazują, że najlepszą metodą jest TS, nieco gorsza jest SA, najgorsze wyniki osiągnięto dla GA. W pracy [10] wykazano, że optymalizacja przy użyciu GA może w wyniku dawać rozwiązania o wartości funkcji celu 100% przewyższającej optimum. Inne prace [139] podają stopień optymalności, ale brak jest dowodu alfa-optymalności. W pracy [85] porównano działanie heurystyki KL z metodą TS oraz SA. Dla grafów o rozmiarze mniejszym niż 100, heurystyka KL jest szybsza niż metoda SA, dla grafów o rozmiarze większym niż 400 rysuje się przewaga metody SA. W pracy [170] porównano działanie heurystyki KL z metodami SA, klastrowaniem i systemem ekspertowym do rozwiązania problemu podziału, dając rekomendację, że heurystyka KL nadaje się rozwiązania problemów o rozmiarze co najwyżej 100 węzłów. W pracy [262] porównano metody zachłanną, KL, klastrowania i SA dla 4 różnych przykładów i 4 różnych architektur. Wykazano, że metoda SA jest zdecydowanie najlepsza przed metodą KL, choć KL ma zdecydowanie krótsze czasy działania i mniejszą złożoność obliczeniową. W pracy [139] porównano metodę GCLP z ILP dla przykładowych grafów o rozmiarach 15 i 27, a także dla grafów losowych o rozmiarach 4-20 i wykazano, że kilka rzędów wielkości dłuższy czas obliczeń ILP przynosi co najwyżej 30% poprawę parametrów. Przy założeniu rozsądnego czasu obliczeń, górną granicą rozmiaru grafu dla komercyjnego narzędzia CPLEX było 20, natomiast metoda GCLP znajdowała rozwiązania dla grafów o rozmiarze 500.

Tabela 2 przedstawia zalecane w pracy [57] metody optymalizacji w zależności od rozmiaru problemu.

Tabela 2. Zalecana metoda optymalizacji w zależności od rozmiaru problemu [57]

Rozmiar problemu	Metoda
Mały	Enumeracja
Średni	Metoda podziału i ograniczeń B&B Programowanie dynamiczne DP Szukanie A*
Duży	Metoda podziału i ograniczeń B&B z wariantami Heurystyki specyficzne dla problemu Algorytmy genetyczne GA Symulowane wyżarzanie SA Poszukiwanie tabu TS

W niniejszej pracy nie jest rozważane projektowanie procesorów ASIP, ani też nie zastosowano architektury sprzętowej ściśle związanej z CPU (ang. *tightly coupled*) (np. koprocesor rozszerzający listę rozkazów). W takich architekturach narzut komunikacji jest znikomy a instrukcje koprocesora są pobierane w podobnych cyklach, co indukuje podział drobnoziarnisty i optymalizację systemu na poziomie pojedynczych instrukcji.

W systemach wbudowanych podział funkcjonalności na sprzęt i oprogramowanie zazwyczaj odbywa się dla architektur luźno związanych (ang. *loosely coupled*), w których narzut komunikacji jest niepomijalny. Powoduje to, że podział drobnoziarnisty jest nieefektywny z uwagi na narzut komunikacyjny i predestynuje podział na poziomie gruboziarnistym. W związku z tym rozmiar wynikowego grafu DAG nie stanowi ograniczenia dla metody podziału i ograniczeń, ani też innych zalecanych do stosowania dla problemów o rozmiarze średnim [57].

We współcześnie realizowanych wbudowanych systemach sprzętowo-programowych dokonuje się redukcji kosztów i próbuje się jednocześnie optymalizować wiele parametrów rozwiązania. Realizowane jest to zazwyczaj przez wykorzystanie gotowych architektur, które wnoszą określone ograniczenia do przestrzeni projektowej i formułowanie wielokryterialnych funkcji celu. Pożądana metoda rozwiązania problemu podziału powinna umożliwiać uzyskanie optymalnego wyniki również bardzo ścisłych ograniczeniach. Powinna również umożliwić stosowanie dowolnej funkcji celu, również dla innych metryk niż tylko czas wykonania i zużycie zasobów sprzętowych.

Inną obserwowaną tendencją jest dostępność wielu możliwych implementacji danego zadania, co indukuje rozszerzony problem podziału, a także podział na dowolną architekturę sprzętową, która może zawierać dowolną ilość partycji (np. wiele procesorów, koprocesorów, lub programowalnych macierzy FPGA). Dostępność implementacji implikuje, że znane są ich parametry, przez co w procesie podziału operuje się wartościami dokładnymi, rzadziej ich estymatami. Wartości dokładne umożliwiają przeprowadzenie podziału metodą dokładną, uzyskując globalnie optymalne rozwiązanie. Jednocześnie rzadziej przeprowadza się analizę wrażliwości wartości rozwiązania na zmianę danych wejściowych.

Jak wyspecyfikowano w rozdziale 1.1, pożądana metoda rozwiązania problemu podziału powinna charakteryzować się określonymi właściwościami. Z analizy metod przeprowadzonych w poprzednich podrozdziałach wynika, że właściwości są spełniane w różnym zakresie:

- przewidywalny czas obliczeń; złożoność obliczeniowa może być:
  - stała - zadana liczba iteracji lub czas obliczeń:
    - GA, TS – zależna od zadanych parametrów, ale niezależna od liczby zadań,

        - SA – zależna od współczynnika chłodzenia i początkowej temperatury, ale niezależna od liczby zadań;
    - wielomianowa:
        - DP - O(liczba zadań * liczba wartości parametru ograniczenia * liczba implementacji);
        - zachłanny, FL – O(liczba zadań);
        - BBN – O((liczba zadań)^3);
        - KL – O((liczba zadań)^2*log(liczba zadań));
        - KLMF – O(liczba zadań * log(liczba zadań));
        - Klastrowanie – O((liczba zadań)^3)
        - GCLP – O((liczba zadań)^2)
    - nie wielomianowa:
        - ILP (Simplex/Gomory) –O(liczba zadań * liczba implementacji)$^{\text{liczba ograniczeń}}$
        - B&B – O(liczba implementacji $^{\text{liczba zadań}}$).
- rozwiązanie dopuszczalne:
    - jest generowane tylko na końcu obliczeń: ILP, DP, zachłanny, FL, klastrowanie, GCLP;
    - jest generowane w trakcie obliczeń:
        - B&B – najszybciej w strategii DFS;
        - GA, SA, TS, BBN, KLFM – w procesie poprawy rozwiązania.
- alfa-optymalność:
    - nie jest definiowana, gdyż rozwiązanie jest produkowane po zakończeniu obliczeń: ILP, DP, zachłanny, FL, klastrowanie, GCLP;
    - nie jest określona, gdyż nie są znane ograniczenia na wartość rozwiązania optymalnego: GA, SA, TS, BBN, KLFM;
    - jest określona przy pomocy funkcji ograniczeń: B&B – określona przez wartość funkcji ograniczenia dolnego i górnego zgodnie z zależnością (25).
- rozwiązanie optymalne:
    - nie jest gwarantowane: GA, SA, TS, zachłanny, BBN, KLFM, klastrowanie, FL, GCLP;
    - może być osiągnięte przy uproszczeniach lub dodatkowych założeniach: DP – przy ograniczeniach jednego parametru i rekursywnej funkcji celu;
    - jest gwarantowane:
        - ILP – dowód optymalności LP wymaga przejrzenia wierzchołków sąsiednich oraz uzyskania rozwiązania całkowitoliczbowego;
        - B&B – może być wygenerowane wcześniej, ale dowód jego optymalności wymaga przejrzenia całego drzewa.
- rozszerzony problem podziału:
    - jest wspierany przez metodę:
        - ILP – modelowany przez dodatkowe zmienne;
        - DP, zachłanny – wybór implementacji jest decyzją podejmowaną na każdym etapie;
        - B&B – podział politomiczny;
        - GA – modelowany przez kodowanie alleli dla genu;
        - SA, TS, BBN, GCLP – modelowany w sposób naturalny;
    - nie jest wspierany przez metodę: klastrowanie, KLFM.
- wielokryterialna funkcja celu jest:
    - ustalona: GCLP – minimalizacja zużycia zasobów sprzętowych (jednokryterialna);

  - liniowa: ILP;
  - rekursywna: DP – typu $f_n(d_n) = F(x_n, f_{n-1}(d_{n-1}))$ – zależność (8);
  - dowolna:
    - zachłanny, FL, klastrowanie – definiowane przez metrykę bliskości;
    - B&B – rekursywna funkcja celu bez metryk globalnych ułatwia obliczanie ograniczenia dolnego;
    - GA – rekursywna funkcja celu bez metryk globalnych ułatwia obliczenie po operacji krzyżowania i mutacji;
    - SA, TS, – rekursywna funkcja celu bez metryk globalnych ułatwia obliczanie ΔE;
    - BBN – modelowane przez propagację dowodów;
    - KLFM – rekursywna funkcja celu bez metryk globalnych ułatwia obliczanie jej zmiany po pojedynczym przesunięciu, ale efektywna implementacji tablicy zmiany kosztów wymaga kwantyzacji zmian.
- ograniczenia są stosowane do:
  - żadnego parametru: FL, KLFM – metoda nie wspiera ograniczeń, muszą być modelowane przez funkcję celu jako ograniczenia twarde lub miękkie;
  - jednego parametru:
    - DP – z uwagi na konieczność ograniczenia liczby stanów po kwantyzacji;
    - GCLP – ograniczeniom podlega czas wykonania;
  - wszystkich parametrów lub ich dowolnej kombinacji liniowej: ILP;
  - wszystkich parametrów lub ich dowolnej zależności:
    - B&B – rekursywna postać zależności ułatwia sprawdzanie ograniczeń;
    - GA, SA. TS, zachłanny, klastrowanie – możliwość stosowania ograniczeń twardych i miękkich, ale metoda może generować rozwiązania niedopuszczalne;
    - BBN – modelowane przez propagację dowodów.

Właściwości wszystkich analizowanych metod przedstawia Tabela 3. Metaheurystyki (np. GA, SA, TS) nie są pożądane, gdy wymagane jest rozwiązanie optymalne lub co najmniej rozwiązanie z kryterium alfa-optymalności. Algorytm zachłanny nie jest pożądany, gdy mamy do czynienia ze ścisłymi ograniczeniami wielu parametrów i funkcję celu, w której dominuje składnik metryki globalnej (np. czas wykonania). Heurystyka KLFM operuje funkcją zmiany kosztu, co jest trudne do wyznaczenia w przypadku metryk globalnych i zwiększa złożoność obliczeniową. Heurystyka nie modeluje także rozszerzonego problemu podziału ani ścisłych ograniczeń, które należy uwzględnić dopiero w funkcji celu.

Z przedstawionych opisów algorytmów wynika, że metoda podziału i ograniczeń B&B spełnia wszystkie kryteria narzucone na pożądane cechy metody rozwiązania problemu rozszerzonego podziału funkcjonalności na sprzęt i oprogramowanie. Metoda zapewnia rozwiązanie optymalne, w trakcie swojego działania produkuje również rozwiązania dopuszczalne z określoną metryką alfa-optymalności, co umożliwia wcześniejsze przerwanie obliczeń. Rozszerzony problem podziału jest modelowany za pomocą politomicznego podziału przeszukiwanej przestrzeni rozwiązań. Wielokryterialna funkcja celu może być dowolnego typu (również nieliniowa) i zawierać metryki addytywne oraz globalne. Jej właściwości mają zasadnicze znaczenie przy konstrukcji funkcji ograniczenia dolnego. Ograniczenia mogą być zadawane dla dowolnych parametrów i dla dowolnej ich zależności, co daje wielką elastyczność w modelowaniu złożonych ograniczeń projektowych. Ścisłe ograniczenia pomagają eliminować więcej rozwiązań z przestrzeni projektowej,

stanowiąc czynnik przyspieszający osiągnięcie rozwiązania optymalnego. Wykładniczy czas obliczeń, wobec zastosowań do podziału gruboziarnistego, nie jest czynnikiem upośledzającym użycie metody. Czas obliczeń może zostać znacznie zredukowany dzięki technikom, które zostały opisane w rozdziale 4 oraz w rozdziale 5.3.

Tabela 3. Porównanie metod optymalizacji stosowanych do rozwiązania problemu podziału

Metoda	Dokładne / Enumeracyjne			Metaheurystyki / Iteracyjne			Heurystyki / Konstruktywne					
	ILP	DP	B&B	GA	SA	TS	Greedy	FL	BBN	KLFM	Cluster	GCLP
Autor	George Danzig (1947) Ralph Gomory (1958)	Richard Bellman (1953)	Alisa Land, Alison Doig (1960)	John Holland (1975)	Scott Kirkpatrick (1983)	Fred Glover (1986)	---	Lotfi Zadeh (1965)	Judea Pearl (1985)	Brian Kernighan, Shan Lin (1970), Charles Fiduccia, Robert Mattheyses (1982)	---	Asawaree Kalavade, Edward Lee (1994)
Czas obliczeń	Wykładniczy względem liczby węzłów i ograniczeń	Wielomianowy	Wykładniczy względem liczby węzłów	Niezależny od liczby węzłów, zależny od kryteriów stopu	Niezależny od liczby węzłów, zależny od kryteriów stopu	Niezależny od liczby węzłów, zależny od kryteriów stopu	Liniowy względem liczby węzłów	Liniowy względem liczby węzłów	Wielomianowy $O(n^3)$	Wielomianowy KL: $O(n^2 \log n)$ KLFM: $O(n \log n)$	Wielomianowy $O(n^3)$	Wielomianowy GCLP: $O(n^2)$ MIBS: $O(n^3+n^2B)$
Wczesne uzyskanie rozwiązania dopuszczalnego	Nie	Nie	Tak (DFS)	Tak	Tak	Tak	Nie	Nie	Tak	Tak	Nie	Nie
Alfa-optymalność	Nie dotyczy	Nie dotyczy	Tak	Nie	Nie	Nie	Nie dotyczy	Nie dotyczy	Nie	Nie	Nie dotyczy	Nie dotyczy
Uzyskanie rozwiązania optymalnego	Tak	Tak, przy dodatkowych założeniach	Tak	Nie	Nie, choć jest asymptotycznie zbieżny	Nie	Nie	Nie	Nie	Nie	Nie	Nie
Rozszerzony problem podziału	Tak – modelowany przez dodatkowe zmienne	Tak	Tak	Tak – modelowany przez kodowanie chromosomu	Tak	Tak	Tak	Nie – z reguły jest binarny	Tak – modelowany przez dodatkowe hipotezy	Nie	Nie	Tak – modelowany przez IBS
Wielokryterialna funkcja celu	Liniowa	Rekursywna	Dowolna	Dowolna	Dowolna	Dowolna	Dowolna z metrykami lokalnymi	Dowolna	Brak, musi być modelowane propagacją dowodów	Dowolna	Dowolna, definiowana przez metryki bliskości	Nie – funkcja jest ustalona
Ograniczenia na wiele parametrów	Liniowe	Jeden parametr	Dowolne	Dowolne	Dowolne	Dowolne	Dowolne	Brak możliwości sztywnych ograniczeń	Brak, musi być modelowane propagacją dowodów	Brak, musi być modelowane funkcją celu	Dowolne	Nie – ograniczenia są ustalone

4. Metoda podziału i ograniczeń B&B

Metoda podziału i ograniczeń B&B jest najszerzej stosowanym narzędziem do rozwiązywania NP-trudnych problemów optymalizacji dyskretnej [57][62]. Metoda może być zastosowana zarówno do maksymalizacji jak i minimalizacji funkcji celu. W procesie podziału rozważany jest tylko problem minimalizacji. Dane wejściowe metody nie muszą zostać opisane jako zbiór równań i nierówności [39].

Metoda znajduje rozwiązanie optymalne, jednak przeszukanie wszystkich przypadków jest efektywne z uwagi na wykładniczą złożoność obliczeniową. Główną ideą metody B&B jest ograniczanie wzrostu drzewa poszukiwań [57]. Aby uzyskać wysoką efektywność algorytmu należy stosować reguły heurystyczne przy obliczaniu funkcji ograniczenia dolnego do odrzucania nieoptymalnych rozwiązań na jak najwcześniejszym etapie [39]. Zastosowanie reguł eliminujących nierokujące gałęzie drzewa poszukiwań znacznie skraca czas wykonania [18].

Metoda B&B znajduje zastosowanie w przestrzeniach dyskretnych oraz ciągłych [42], optymalizacjach deterministycznych oraz stochastycznych [193]. Implementacja metody B&B wymaga dopasowania do natury problemu, co kontrastuje z ogólnym podejściem programowania liniowego [124]. Obszerną dyskusję implementacji (także równoległych i rozproszonych), korzyści i przedstawienie wyników zawierają prace [21][32][45][62][186][209].

Klasycznymi zastosowaniami metody są:

- problem komiwojażera (ang. *Travelling Salesman Problem*) [57][62][78][186],
- podział grafu (ang. *Graph Partitioning Problem*) [62][186],
- przydział kwadratowy (ang. *Quadratic Assignment Problem*) [62][186].

4.1. Funkcja celu

Funkcja celu (ang. *goal / objective function*) jest miarą jakości rozwiązania. Optymalizacja jednokryterialna definiuje funkcję celu jako wartość jednego z parametrów rozwiązania. W problemie podziału HW/SW są to zazwyczaj zasoby sprzętowe lub czas przetwarzania. Optymalizacja jednokryterialna jest stosunkowo łatwo i wydajnie przeprowadzana za pomocą algorytmów heurystycznych. Optymalizacja wielokryterialna zazwyczaj definiuje funkcję celu jako liniową kombinacją wartości parametrów rozwiązania. Funkcja celu może być także nieliniowa, ale jeśli jej obliczanie da się wyrazić w formie rekurencyjnej, nie stanowi to utrudnienia dla obliczania stowarzyszonej funkcji ograniczenia dolnego. W przeciwnym przypadku, funkcja ograniczenia dolnego jest trudniejsza do skonstruowania.

4.2. Ograniczenia

Ograniczenia parametrów rozwiązania (ang. *parameter constraints / variable bounds* [57]) są wprowadzane na skutek limitów zasobów sprzętowych platformy implementacyjnej (np. zasoby sprzętowe, rozmiar pamięci kodu, stosu, danych) i na skutek wymagań funkcjonalnych (np. czas przetwarzania, pobór mocy, złożoność kodu źródłowego itp.). Dodatkowe ograniczenia wynikają z wzajemnej zależności niektórych parametrów (np. łączny rozmiar pamięci) lub wzajemnej wyłączności (ang. *mutual exclusion*) parametrów binarnych [50].

Ograniczenia definiują przestrzeń możliwych rozwiązań w przestrzeni wszystkich potencjalnych rozwiązań (zależność (9)). W problemie podziału HW/SW definiuje się zazwyczaj ograniczenia dla poszczególnych parametrów rozwiązania, co generuje

przestrzeń możliwych rozwiązań w postaci hiperkostki (ang. *hypercube*). Możliwe też są ograniczenia sformułowane w postaci dowolnych funkcji uzależniających od siebie wartości parametrów. Jeśli określanie dopuszczalności bieżącego rozwiązania da się wyrazić w formie rekurencyjnej, przy wykorzystaniu ograniczeń z rozwiązania poprzedniego, to znacznie upraszcza to obliczenia. W przeciwnym przypadku ograniczenia należy sprawdzać dla każdego rozwiązania niezależnie.

4.3. Rozwiązanie początkowe

Rozwiązanie początkowe (ang. *initial solution*) jest wartością górnego ograniczenia funkcji celu. Rozwiązanie początkowe musi leżeć w przestrzeni dopuszczalnych rozwiązań. Dobre rozwiązanie początkowe nie jest niezbędne dla metody B&B, ale pozwala na znaczną redukcję czasu obliczeń, zwłaszcza w strategiach wyboru podproblemu BFS i BeFS, przez odrzucanie nierokujących gałęzi.

Rozwiązanie to jest zazwyczaj znajdowane heurystycznie. Zazwyczaj używane są metaheurystyki: symulowane wyżarzanie SA, algorytmy genetyczne GA oraz poszukiwanie tabu TS [62]. Często stosuje się także metodę zachłanną, w której dla kolejnych zadań wybierane są implementacje według określonego kryterium [52]. Innym rozwiązaniem jest stosowanie strategii wyboru podproblemu DFS do momentu uzyskania pierwszego rozwiązania dopuszczalnego i następnie zastosowanie pożądanej strategii w dalszych poszukiwaniach [57].

4.4. Strategia iteracji

Strategia iteracji definiuje kolejność wykonywania operacji rozmnażania węzłów, obliczania ograniczeń, odrzucania węzłów i dodawania ich do listy *live*.

Algorytm 14. Schemat strategii gorliwej w metodzie B&B [62]

1. Inicjalizacja: Incumbent:=∞; $LB(P_0):=g(P_0)$; Live:={P_0,$LB(P_0)$};
2. Selekcja: wybór węzła P z listy Live do przetwarzania; Live:=Live\{P};
3. Rozgałęzienie & Ograniczenie:
   - Podział problemu P na podproblemy $P_1,\ldots,P_k$;
   - Dla każdego P_i wyznaczenie $LB(P_i):=g(P_i)$;
   - Jeśli $LB(P_i)$==f(X) dla rozwiązania dopuszczalnego X i f(X)<Incumbent → Incumbent:=f(X) i Solution:=X;
   - Jeśli LB(Pi)<Incumbent → Live:=Live U {(Pi,LB(Pi))}
4. Stop: Jeśli Live==Ø → OptimalSolution:=Solution; OptimumValue:=Incumbent;

W strategii gorliwej (ang. *eager*) [62] (Algorytm 14) wybierany jest kolejny węzeł z listy *live*, wykonywany jest podział (rozmnażanie węzłów) i generowane są węzły potomne. Jeśli węzeł potomny jest węzłem terminalnym, następuje porównanie z wartością globalnego minimum i jej ewentualne uaktualnienie. W przeciwnym przypadku obliczania jest wartość funkcji ograniczenia dolnego dla gałęzi i porównywana z wartością globalnego minimum. Jeśli żadne rozwiązanie w gałęzi nie wygeneruje rozwiązania o mniejszej wartości funkcji celu niż wartość globalnego minimum, cała gałąź jest eliminowana z poszukiwań. W przeciwnym przypadku węzeł jest dodawany do listy *live*. W strategii gorliwej porównanie wartości funkcji ograniczenia dolnego z wartością globalnego minimum następuje obliczane natychmiast po wygenerowaniu nowych węzłów w grafie poszukiwań.

Algorytm 15. Schemat strategii leniwej w metodzie B&B [62]

1. Inicjalizacja: Incumbent:=-∞; Live:={P_0,-∞};
2. Selekcja: wybór węzła P z listy Live do przetwarzania; Live:=Live\{P};
3. Ograniczenie & Rozgałęzienie:

- dla P wyznaczenie LB(P):=g(P);
- Jeśli LB(P)==f(X) dla rozwiązania dopuszczalnego X i f(X)<Incumbent → Incumbent:=f(X) i Solution:=X;
- Jeśli LB(P)<Incumbent → podział problemu P na podproblemy $P_1,\ldots,P_k$ i dodanie ich do listy Live: Live:=Live U {(P_i,LB(P))}

4. Stop: Jeśli Live=Ø → OptimalSolution=Solution; OptimumValue=Incumbent;

W strategii leniwej (ang. *lazy*) [62] (Algorytm 15) wybierany jest kolejny węzeł z listy *live*, obliczana jest wartość funkcji ograniczenia dolnego i porównywana z wartością globalnego minimum. Jeśli gałąź nie zostaje odrzucona, wykonuje się podział (rozmnażanie węzłów) a wygenerowane węzły dodaje do listy *live*. W strategii leniwej porównanie wartości funkcji ograniczenia dolnego z wartością lokalnego minimum jest odroczone do momentu usunięcia węzła z listy *live*. Istnieje zatem większe niż w strategii gorliwej prawdopodobieństwo odrzucenia danej gałęzi. Odbywa się to kosztem większej liczby węzłów przebywających jednocześnie na liście live.

4.5. Strategie wyboru podproblemu

Strategie wyboru podproblemu (ang. *selection rule / strategy, next subproblem selection strategy* [62], *node selection policy* [57]) określają kolejności usuwania węzłów z listy *live*. Strategie zwane też są regułami wyboru ze zbioru kolejkowania, regułami wyboru określającymi kolejność rozwiązywania podproblemów [78], lub strategiami trawersowania drzewa (ang. *tree traversal* [32]). W literaturze spotyka się następujące strategie wyboru:

- szukanie wszerz BFS – wybierany jest węzeł najbliższy korzeniowi. Rozwijane są wszystkie możliwe gałęzie poszukiwań na danym poziomie przed przejściem na głębszy poziom drzewa.
- szukanie wgłąb DFS – wybierany jest węzeł najbardziej oddalony od korzenia. W pierwszej kolejności rozwijana jest gałąź na najniższym (najgłębszym) poziomie. W ten sposób uzyskuje się wczesne rozwiązanie początkowe [57]. Jeśli żadna z gałęzi na najniższym poziomie nie może zostać rozwinięta wykonywany jest powrót na poziom wyższy. Strategię DFS najłatwiej zaimplementować w sposób rekurencyjny.
- szukanie ku najlepszemu BeFS – wybierany jest węzeł o najmniejszej wartości ograniczenia dolnego. Gałąź może znajdować się na dowolnym poziomie drzewa poszukiwań.
- strategie dynamiczne – wybierany jest węzeł w oparciu o analizę jego wpływu na inne węzły [102].

Podproblem P nazywamy krytycznym, jeśli dla danej funkcji ograniczenia dolnego g(x) wartość funkcji g(x) dla podproblemu P jest mniejsza niż wartość funkcji celu f(x) dla rozwiązania optymalnego. Oznacza to, że podproblem krytyczny zawsze zostanie dodany do listy *live*.

W problemie podziału HW/SW wszystkie rozwiązania całkowite leżą na tym samym n-tym poziomie w drzewie poszukiwań. W związku z tym strategia szukania wszerz nie jest efektywna, gdyż rozwiązania całkowite będą przeszukiwane dopiero na końcu. Zgodnie z zależnością (22) wartości funkcji ograniczenia dolnego dla węzłów rodzicielskich jest mniejsza niż dla potomnych. Powoduje to, że strategia szukania ku najlepszemu preferuje węzły leżące bliżej korzenia, a rozwiązania całkowite będą przeglądane w następnej kolejności. Dla problemu podziału HW/SW stosuje się zazwyczaj strategię szukania wgłąb lub strategie mieszane.

4.6. Strategie rozgałęzień

Strategie rozgałęzień (ang. *branching rules*) definiują kolejność zmiennych, dla których ustalane są wartości podczas podziału (rozmnażania węzłów). Strategie te zwane są też strategiami rozmnażania wierzchołków, podziału, tworzenia podproblemów [78], wyboru zmiennej (ang. *variable selection policy*) [57] lub regułami rozgałęzienia dla zbioru kolejkowania. W literaturze spotyka się następujące strategie rozgałęzień:

- Branch First – wybierane jest zadanie pierwsze w sensie porządku topologicznego, którego poprzedniki są już zaalokowane [18],
- Branch Last – wybierane jest zadania ostatnie w sensie porządku topologicznego, którego następniki są już zaalokowane,
- Branch Least – wybierane jest zadanie o najmniejszej liczbie implementacji, co pozwala na budowę drzewa o najmniejszej liczbie gałęzi, choć liczba liści pozostaje taka sama,
- Branch Restrictive – wybierane jest zadanie, które jest najbardziej restrykcyjne w stosunku do nakładanych ograniczeń [57], co zapewnia odrzucanie rozwiązań na wczesnym etapie,
- Branch Dominant / Suitable – wybierane jest zadanie o maksymalnej wartości dominacji jednej z implementacji nad pozostałymi implementacjami danego zadania [20][52][55], co zapewnia najmniejszą liczbę powrotów,
- Branch Best – wybierane jest zadanie o minimalnej funkcji celu dla jednej z implementacji [55].

Strategia rozgałęzień pozwala dobrać kolejność zmiennych w celu osiągnięcia pożądanych właściwości drzewa poszukiwań.

4.7. Funkcja ograniczenia dolnego

Funkcja ograniczenia dolnego LB jest optymistycznym niedoszacowującym estymatorem wartości funkcji celu dla najlepszego rozwiązania osiągalnego w danej podprzestrzeni [57]. LB jest kluczowym komponentem metody B&B, w tym sensie, że funkcja kiepskiej jakości nie zostanie skompensowana przez dobre strategie wyboru i podziału [62]. Funkcja powinna działać w czasie wielomianowym i dawać wartości bardzo bliskie optimum dla danej gałęzi. Im lepsza funkcja ograniczenia dolnego tym mniejsze będzie drzewo poszukiwań [57][62]. Głównym problemem metody B&B jest fakt, iż decyzje o podziale są podejmowane w oparciu o ograniczoną informację o prawdopodobieństwie, że dana gałąź doprowadzi do rozwiązania optymalnego [102].

Dla funkcji ograniczenia dolnego g(x): $g(x) \leq f(x)$ dla $x \in S$ zachodzą następujące zależności [62]:

$$g(\mathrm{Pi}) \leq f(\mathrm{Pi}), \text{dla wszystkich wierzchołków w drzewie} \tag{20}$$

$$g(\mathrm{Pi}) = f(\mathrm{Pi}), \text{dla wszystkich wierzchołków terminalnych (liści)} \tag{21}$$

$$g(Pi) \geq g(Pj), \text{dla wszystkich wierzchołków Pi potomnych względem Pj} \quad (22)$$

Zgodnie z zależnością (22) funkcja ograniczenia dolnego generuje coraz lepsze przybliżenie w miarę dostarczania coraz większej ilości informacji i zbiega się z funkcją celu dla wierzchołków terminalnych (liści) zgodnie z zależnością (21). Oznacza to, że w przestrzeniach ciągłych różnica pomiędzy górnym i dolnym ograniczeniem jednostajnie zmierza do zera, jeśli przeszukiwana przestrzeń kurczy się do pojedynczego punktu [42]. Wartość górnego ograniczenia zazwyczaj przyjmuje się jako najmniejszą znalezioną dotąd wartość funkcji celu [55][62].

Dla funkcji g(x) będącej ograniczeniem dolnym dla funkcji celu f(x) zachodzą następujące zależności [62]:

$$\min_{x \in P} g(x) \leq \min_{x \in P} f(x) \leq \min_{x \in S} f(x) \quad (23)$$

$$\min_{x \in P} g(x) \leq \min_{x \in S} g(x) \leq \min_{x \in S} f(x) \quad (24)$$

gdzie:

P – zbiór potencjalnych rozwiązań,
S – zbiór dopuszczalnych rozwiązań, $S \subseteq P$.

W przestrzeniach ciągłych jest spełniony warunek zbieżności [43]:

$$\forall \varepsilon > 0 \; \exists \delta > 0 \; \forall Q \subseteq Q_{init}, \text{size}(Q) \leq \delta \rightarrow UB(Q) - LB(Q) \leq \varepsilon \quad (25)$$

gdzie:

Q – zbiór rozwiązań (n-wymiarowy),
size(Q) – średnica zbioru Q (długość najdłuższej krawędzi w zbiorze Q),
UB – funkcja górnego ograniczenia,
LB – funkcja dolnego ograniczenia.

Stosowane są różne strategie obliczania funkcji ograniczenia dolnego [78]:

- minimalizacja g(x) w zbiorze dopuszczalnych rozwiązań,
- bezpośrednia – przez przejrzenie wszystkich rozwiązań w danym podzbiorze, stosowane, gdy moc podzbioru jest wystarczająco mała,
- pośrednia – przez odrzucenie rozwiązań dla podzbioru gdzie wartość ograniczenia dolnego dla podzbioru jest większa niż aktualne minimum,
- podział – dalsze rozbijanie na podzbiory,
- relaksacja – przez odrzucenie ograniczeń zgodnie z zależnością (24), co może sprowadzić problem do klasy problemów wielomianowych [62] i uczynić go łatwiejszym do rozwiązania. Po znalezieniu rozwiązania sprawdza się, czy rozwiązanie spełnia ograniczenia. Jeśli ograniczenia są spełnione to rozwiązanie zrelaksowane stanowi jednocześnie rozwiązanie podproblemu (ang. *fathom*), w przeciwnym przypadku rozwiązanie zrelaksowane określa dolne ograniczenie dla problemu.

5. Przeprowadzone badania i uzyskane wyniki

W celu udowodnienia tez przedstawionych w rozdziale 1.2 zaprojektowano zestaw badań i eksperymentów. Przeprowadzono badania w ramach prowadzonej pracy dyplomowej dyplomanta [72], projektu dla firmy Alatek [246][247] oraz projektów akademickich, których wyniki prezentowano na konferencjach [249][251].

Proces współbieżnego projektowania wymaga dostarczenia dokładnych danych o parametrach realizacji. Niektóre parametry (np. czas wykonania, czas komunikacji, zajętość pamięci danych) można określić tylko dynamicznie poprzez symulację całego systemu. Z powodu braku otwartych środowisk symulacji heterogenicznej, zaprojektowano i wykonano system symulacyjny. System umożliwia użycie różnych symulatorów HDL, symulatorów ISS, emulatorów FPGA, fizycznych układów CPU. System jest wykorzystywany do symulacji i uruchamiania modułów w różnym cyklu projektowania.

Najistotniejszym krokiem w procesie współbieżnego projektowania jest proces podziału. Zaprojektowano aplikację Partitioner realizującą metodę B&B, zdefiniowano funkcję ograniczenia dolnego i zaproponowano format grafów DAG. Do walidacji aplikacji Partitioner zaprojektowano aplikację testową DAG Generator, generującą losowe grafy DAG według określonych schematów.

Podstawowym problemem procesu podziału przy użyciu metody B&B jest jej długi czas działania. W celu jego skrócenia należy odpowiednio dobrać parametry metody B&B. Zbadano wrażliwość metody na zmianę parametrów i zidentyfikowano parametry, których zmiana może skutkować znaczącym przyspieszeniem. Zaprojektowano także procedurę automatycznego strojenia parametrów.

Na podstawie wykonanych sprzętowych i programowych realizacji algorytmów kryptograficznych, zaimplementowanego systemu do heterogenicznej symulacji oraz zrealizowanej aplikacji Partitioner do podziału funkcjonalności na sprzęt i oprogramowanie, zaprojektowano 2 serie eksperymentów heterogenicznej realizacji algorytmów kryptograficznych. Zastosowano proces współbieżnego projektowania obejmujący specyfikację modułów, ich kompilację i syntezę, współbieżną symulację, ekstrakcję parametrów, proces podziału oraz realizację algorytmu na docelowej platformie SoC.

5.1. Realizacja środowiska heterogenicznej symulacji sprzętowo-programowej

Analizę i realizację środowiska heterogenicznej symulacji sprzętowo-programowej opracowaną dla firmy Alatek opisano w pracach [246][247].

5.1.1. Problemy systemów współbieżnej symulacji

Zintegrowane środowiska heterogenicznej symulacji, takie jak Mentor Graphics Seamless oraz Synopsys DesignWare, używają dedykowanych aplikacji i zazwyczaj nie jest możliwa praca z aplikacjami stosowanym dotychczas przez projektanta. Nie wszystkie środowiska wspomagają dowolne debugery, tryb symulacyjny i metodę synchronizacji. W połączeniu z dużą ceną stanowi to poważną barierę ograniczającą szersze zastosowanie tych środowisk [246][247].

Przeprowadzone badania [114][164] wykazały, że w systemach wbudowanych około 95% cykli symulacyjnych jest związanych z interakcją procesora z pamięcią: pobrania rozkazów, zapisy i odczyty, podczas gdy cykle I/O stanowią przeciętnie 5% wszystkich cykli magistrali [247]. Oznacza to, że układy I/O pozostają bezczynne przez większość czasu, a przeprowadzana jest wielokrotna symulacja identycznych operacji na pamięci, nie dostarczając istotnych informacji do symulatora.

Przyspieszenie można osiągnąć poprzez szczegółową symulację układów I/O w symulatorze HDL tylko w czasie cykli I/O, podczas gdy pozostałe cykle operacji na pamięci są symulowane w debugerze z dużą większą prędkością. Takie rozwiązanie daje wzrost wydajności symulacji bez utraty obserwowalności stanów układów I/O. Synchronizacja symulatorów następuje tylko na czas komunikacji [246][247]. Opisana koncepcja została zastosowana w środowiskach Mentor Graphics Seamless [70][114][164] i ARC MetaSim (obecnie Synopsys DesignWare). Definiowane są regiony adresowe, dla których wykonywana jest symulacja HDL lub ISS. Możliwa jest rekonfiguracja regionów pamięci w trakcie symulacji bez potrzeby jej restartowania, poprzez kopiowanie odpowiednich obszarów pomiędzy symulatorami. Seamless do zarządzania pamięcią stosuje *Memory Image Server* [246].

Wadą opisanego rozwiązania jest brak synchronizacji globalnej, co oznacza, że ilość cykli w symulatorze HDL nie pokrywa się z ilością cykli w symulatorze ISS. Utrudnia to uruchamianie aplikacji pracujących w czasie rzeczywistym. Nie jest możliwa także autonomiczna praca układów I/O, gdyż symulator HDL nie dokonuje ich symulacji w trakcie wykonywania oprogramowania. Mimo opisanych wad, w środowiskach współbieżnej symulacji tryb ten jest stosowany najczęściej.

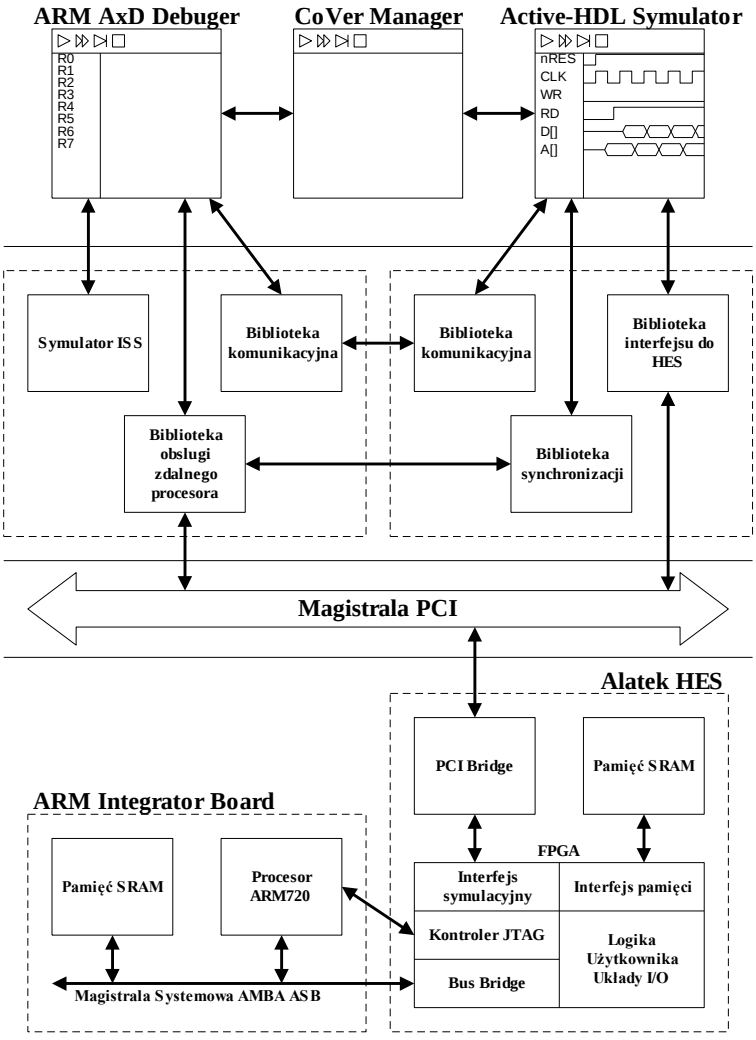
5.1.2. Przyjęte założenia

Projekt środowiska współbieżnej symulacji [247] obejmował specyfikację i implementację aplikacji zarządzającej CoVerification Manager, bibliotek dołączanych do debugera i symulatora oraz modułów sprzętowych umieszczanych na karcie HES.

Rysunek 6 przedstawia architekturę systemu współbieżnej symulacji. Do symulacji procesora w trybie HDL i CPU zaprojektowano dwie biblioteki komunikujące się z debugerem przez interfejs RDI [11]. Pierwsza z nich przekazuje komendy sterujące do modelu RTL procesora i pozwala na odczyt rejestrów bezpośrednio z symulatora HDL. Druga biblioteka przekazuje komendy sterujące do umieszczonego w FPGA na karcie HES kontrolera JTAG, który dokonuje translacji komend i manipuluje bezpośrednio stanem fizycznego układu na karcie procesora. W trybie ISS procesor jest symulowany przez standardowy symulator ISS ARMulator, do którego są dołączane transakcyjne lub puste moduły układów I/O.

Do symulacji układów I/O z symulatorem ISS zaprojektowano biblioteki komunikujące się z symulatorem HDL przez interfejs PLI oraz VHPI. Biblioteki te w połączeniu z transakcyjnymi lub pustymi modułami układów I/O umożliwiają wymianę danych pomiędzy symulatorami ISS i HDL. Podobna biblioteka jest używana do wymiany danych podczas symulacji procesora w trybie HDL. W trybie CPU używany jest moduł synchronizacji symulatorów współpracujący z biblioteką debugera sterującą kontrolerem JTAG. Jeśli jakaś część systemu jest realizowana sprzętowo wykorzystuje się standardową bibliotekę interfejsu do karty HES.

W układzie FPGA na karcie HES umieszczono dodatkowe moduły sprzętowe. Kontroler JTAG generuje komendy JTAG dla układu procesora w oparciu o komendy RDI otrzymywane z biblioteki debugera w trybie CPU. Sprzętowa realizacja tego modułu umożliwia osiągnięcie lepszej wydajności z uwagi na maksymalną częstotliwość zegara równą 8 MHz. Drugim modułem osadzonym w układzie FPGA jest bridge magistrali ASB z konfigurowalnym dekoderem adresowym. Dekoder umożliwia definicję regionów pamięci, określając w ten sposób, które cykle magistrali mają być symulowane przez symulator HDL.



Rysunek 6. Architektura systemu współbieżnej symulacji HW/SW [247]

Przy synchronizacji globalnej dekoder adresowy wymusza pracę z pamięcią symulowaną w symulatorze HDL, umożliwiając wizualizację każdego cyklu magistrali. W takim trybie pracy zegar magistrali jest generowany przez symulator HDL,

synchronizacja z debugerem następuje na granicy wykonania rozkazu i dopiero wtedy są realizowane komendy debugera.

Przy synchronizacji na czas komunikacji symulacja oprogramowania w ISS i CPU przebiega z pełną prędkością z zegarem lokalnym. Odwołanie się do przestrzeni adresowej poza magistralą lokalną powoduje przeprowadzenie symulacji przez symulator HDL i zmianę sygnału zegarowego. W tym trybie pracy uzyskuje się skompresowane przebiegi magistrali, ograniczone tylko do cykli, w których nastąpiło odwołanie poza magistralę lokalną. Wadą takiego rozwiązania jest brak możliwości autonomicznej pracy układów I/O, gdyż symulacja HDL jest przeprowadzana tylko na czas komunikacji.

Praca z pamięcią symulowaną w symulatorze HDL posiada pewne wady. Odczyt zawartości pamięci z aplikacji debugera może być realizowany tylko przez procesor, gdyż wyłącznie ten układ podlega sterowaniu. Powoduje to pojawienie się dodatkowych cykli symulacji niezwiązanych z wykonaniem bieżącego programu. Cykle te są nierozróżnialne dla symulatora HDL.

5.1.3. Zasada działania środowiska heterogenicznej symulacji

W trybie ISS+ISS procesor i pamięć są symulowane przez symulator ISS, gdzie dołączone są transakcyjne modele układów I/O w formie bibliotek aplikacyjnych. Uruchamianie oprogramowania i symulacja układów I/O odbywa się w środowisku debugera, który służy jako magistrala systemowa. Kontrola symulacji układów, wizualizacja stanów wewnętrznych i obsługa sygnałów nie-magistralowych wykonuje się w symulatorze HDL przy pomocy biblioteki symulacyjnej. Umożliwia to uruchamianie złożonego oprogramowania z bardzo dużą szybkością z synchronizacją na czas komunikacji. Wadami są brak wizualizacji przebiegów magistrali i konieczność użycia niesyntezowalnych modeli układów I/O.

W trybie HDL+HDL model TAS procesora, pamięć i układy I/O są połączone magistralą systemową i symulowane w symulatorze HDL, który wizualizuje stany wewnętrzne i daje możliwość sterowania wszystkimi sygnałami. Kontrola wykonania oprogramowania i wizualizacja stanu rejestrów odbywa się w środowisku debugera przy pomocy bibliotek symulacyjnych. Umożliwia to symulację czasową syntezowalnego systemu, śledzenie stanów wewnętrznych i sygnałów na magistrali z jednoczesnym uruchamianiem oprogramowania w środowisku debugera. Wadą jest mała szybkość symulacji modelu RTL procesora i konieczność synchronizacji globalnej.

W trybie ISS+HDL procesor i pamięć są symulowane przez symulator ISS, gdzie dołączony jest pusty moduł układów I/O w formie biblioteki aplikacyjnej. Uruchamianie oprogramowania i wizualizacja stanu rejestrów odbywa się w środowisku debugera. Model BFM magistrali systemowej, układy I/O i dodatkowe bloki pamięci są symulowane w symulatorze HDL, który wizualizuje stany wewnętrzne i daje możliwość sterowania sygnałami. W przypadku, gdy następuje odwołanie do pustego modułu układu I/O, zamiast transakcji do pamięci symulatora ISS wykonywane jest sterowanie modelu BFM za pomocą bibliotek symulacyjnych. Daje to dużą prędkość symulacji oprogramowania z synchronizacją na czas komunikacji, śledzenie sygnałów magistrali i stanów wewnętrznych układów I/O. Możliwa jest także symulacja czasowa, której dokładność jest uwarunkowana szczegółowością zastosowanego modelu BFM.

W trybie ISS+FPGA środowisko programisty działa analogicznie jak w trybie ISS+HDL. Układy I/O są natomiast realizowane w FPGA, a moduły pamięci są odwzorowane w fizyczne układy na karcie akceleratora symulacji. Model BFM

magistrali systemowej, interfejsy układów I/O i pamięci połączone są magistralą systemową w symulatorze HDL, który jedynie wizualizuje stan magistrali. Każdy delta cykl zamiast symulacji powoduje zapis pobudzeń i odczyt odpowiedzi z fizycznych układów na karcie przez bibliotekę symulacyjną. Emulacja pozwala na zwiększenie prędkości symulacji złożonych układów I/O przy uruchamianiu oprogramowania wykonującego dużą ilość zapisów i odczytów.

W trybie CPU+HDL procesor i pamięć jako fizyczne układy są połączone magistralą systemową na karcie procesora. Uruchamianie oprogramowania i wizualizacja rejestrów odbywa się w środowisku debugera w konfiguracji z modelem sprzętowym przy pomocy interfejsu JTAG w układzie FPGA. Pusty model magistrali, układy I/O i dodatkowa pamięć są dołączone do magistrali systemowej w symulatorze HDL, który wizualizuje stany wewnętrzne i daje możliwość sterowania sygnałami. W przypadku, gdy następuje odwołanie do adresu układów I/O, magistrala systemowa przechodzi z trybu pełnej prędkości w tryb zdarzeniowy i każdy cykl magistrali jest symulowany w symulatorze HDL, a wynik podawany do procesora przez bibliotekę symulacyjną i układ FPGA. Zastosowanie fizycznych układów procesora i pamięci oddzielonych bridgem magistrali umożliwia symulację z pełną prędkością z synchronizacją na czas komunikacji oraz śledzenie sygnałów magistrali i stanów wewnętrznych układów I/O.

W trybie CPU+FPGA środowisko programisty działa analogicznie jak w trybie CPU+HDL. Środowisko sprzętowe jest natomiast identyczne jak w trybie ISS+FPGA, umożliwiając jedynie wizualizację stanu magistrali i sterowanie sygnałem zegarowym. W przypadku, gdy następuje odwołanie do adresu układów I/O lub dodatkowej pamięci, następuje synchronizacja na czas komunikacji i symulator HDL wizualizuje stan magistrali, podczas gdy funkcjonalność jest emulowana w FPGA na karcie akceleratora. Innym wariantem jest praca autonomiczna bez wizualizacji w symulatorze, co umożliwia osiągnięcie najwyższej prędkości symulacji przy braku obserwowalności części sprzętowej. Taki tryb pracy jest stosowany przy końcowej weryfikacji całego systemu w czasie rzeczywistym.

Tabela 4 podsumowuje tryby pracy środowiska i podstawowe własności.

Tabela 4. Tryby pracy środowiska współbieżnej symulacji i ich podstawowe własności [247]

	ISS+ISS	HDL+HDL	ISS+HDL	ISS+FPGA	CPU+HDL	CPU+FPGA
Model IO	Transakcje	Czasowy	Czasowy	Cyklowy	Czasowy	Cyklowy
Moduł Symulacji IO	Biblioteka Aplikacyjna	Symulator HDL	Symulator HDL	Emulacja w FPGA	Symulator HDL	Emulacja w FPGA
Model CPU	ISS	TAS	ISS/BFM	ISS/BFM	HW	HW
Moduł Symulacji CPU	Symulator ISS	Symulator HDL	Symulator ISS	Symulator ISS	Zdalny Procesor	Zdalny Procesor
Magistrala ISS	Tak	-	Tak	Tak	-	-
Magistrala HDL	-	Tak	Tak	Tak	Tak	Tak
Magistrala HES	-	-	-	Tak	Tak	Tak
Magistrala CPU	-	-	-	-	Tak	Tak
Pamięć ISS	Tak	-	Tak	Tak	-	-
Pamięć HDL	-	Tak	Tak	-	Tak	-
Pamięć HES	-	-	-	Tak	-	Tak
Pamięć CPU	-	-	-	-	Tak	Tak
Autonomiczna	Tak	-	-	-	-	Tak
Sync. Lokalna	Tak	-	Tak	Tak	Tak	Tak
Sync. Globalna	-	Tak	Teoretyczny	Teoretyczny	Tak	Tak

5.1.4. Wyniki realizacji środowiska heterogenicznej symulacji

Zaprojektowano przykładowy system wbudowany zawierający procesor ARM720, 256kB pamięci i licznik 8253. Oprogramowanie dokonywało odczytów i zapisów do licznika, generując częstą interakcję z układem I/O (1 odczyt i 1 zapis na 10 cykli pobrań instrukcji. W celu uzyskania wiarygodnych wyników wyłączono pamięć podręczną w procesorze oraz układ MMU. W każdym trybie przeprowadzono symulację z synchronizacją lokalną i globalną stosując symulator Active-HDL v5.2 oraz debuger AxD v1.2. Badania przeprowadzono na komputerze z procesorem Athlon 1MHz z 256MB pamięci w systemie Win2000 [247]. Uzyskane wyniki przedstawia Tabela 5.

Tabela 5. Wyniki prędkości symulacji przykładowego systemu wbudowanego [247]

Synchronizacja	ISS+ISS	HDL+HDL	ISS+HDL	ISS+FPGA	CPU+HDL	CPU+FPGA
Autonomiczna	2.570.000	-	-	-	-	16.000.000
Lokalna	950.000	-	156.500	236.800	112.500	295.000
Globalna	-	15	Bez testu	Bez testu	39.215	49.130

Porównanie wyników z innymi środowiskami jest trudne, gdyż nie przeprowadzano badań z wykorzystaniem takiego samego systemu wbudowanego i identycznych symulatorów. Prędkości symulacji podawane w notach aplikacyjnych różnych producentów nie podają architektury systemu wbudowanego, intensywności współpracy z układami I/O, rodzaju synchronizacji, ani używanego sprzętu komputerowego.

5.1.5. Wnioski z realizacji środowiska heterogenicznej symulacji

Tabela 5 pokazuje, że największą prędkość osiąga się w środowisku symulacji homogenicznej w pracy autonomicznej bez potrzeby synchronizacji, używając jednolitego modelu symulacyjnego. Tryb taki jest często stosowany w celu ostatecznej weryfikacji systemu. System wbudowany emulowany na kartach prototypowych jest znacznie szybszy od symulatora ISS, pracuje bowiem z prędkością 16 MHz, a co więcej – w środowisku zbliżonym do docelowego.

Znacznie wolniejsza jest praca z synchronizacją lokalną na czas komunikacji, gdyż badany system intensywnie współpracuje z układem I/O. Symulacja oprogramowania jest szybsza od symulacji układów I/O, która nadto wymaga graficznej wizualizacji przebiegów, stąd też widoczny jest spadek wydajności. Największą prędkość symulacji uzyskuje się w trybie ISS+ISS, gdyż na czas cyklu I/O symulator ISS przekazuje sterowanie do symulatora HDL, który symuluje tylko jedno zdarzenie dyskretne. W trybie ISS+FPGA symulator HDL wykonuje pełen cykl magistrali dla modułu I/O umieszczonego w układzie FPGA. Dla trybu ISS+HDL moduł I/O jest symulowany w symulatorze HDL. Widać zatem, że w miarę wzrostu aktywności symulatora HDL efektywna prędkość symulacji spada. W przypadku modelu fizycznego procesora zmniejszanie się wydajności jest wyraźniejsze. W trybie CPU+FPGA symulator HDL dostarcza jedynie sygnału zegarowego dla cykli I/O i wizualizuje graficznie przebieg magistrali. W trybie CPU+HDL symulator HDL oprócz dostarczania sygnału zegarowego i wizualizacji graficznej musi wykonać jeszcze symulację modułu I/O i przekazać wyniki do CPU. Wskutek tego w badanym systemie prędkość zmniejszyła się prawie trzykrotnie.

Praca z synchronizacją globalną jest najwolniejsza, gdyż każdy cykl magistrali musi zostać synchronizowany przez symulator HDL, a przebiegi magistrali wizualizowane graficznie. W trybie CPU+FPGA symulator HDL dostarcza sygnału

zegarowego dla systemu, odczytuje sygnały magistrali z FPGA i dokonuje graficznej wizualizacji. W trybie CPU+HDL symulator HDL dodatkowo symuluje układy I/O, co w przypadku badanego systemu z prostym układem I/O wiąże się z nieznacznym spadkiem wydajności. Tryby ISS+HDL i ISS+FPGA nie zostały przebadane. W trybie HDL+HDL osiąga się najniższą prędkość symulacji, gdyż model RTL procesora jest symulowany przez symulator HDL. Synchronizacja globalna umożliwia jednak autonomiczną pracę modułów I/O, przez co jest często stosowana w systemach przetwarzania równoległego.

Przedstawiony system współbieżnej symulacji może znaleźć zastosowanie w całym cyklu projektowym [247]. Weryfikację funkcjonalną specyfikacji przeprowadza się w trybie ISS+ISS. Symulację oprogramowania i sprzętu przed syntezą wykonuje się w trybie ISS+HDL. Do walidacji zależności czasowych na magistrali systemowej stosuje się tryb HDL+HDL. Weryfikację sprzętu po syntezie dokonuje się w trybie ISS+FPGA. Weryfikację działania procesora i pamięci wykonuje się w trybie CPU+HDL. Końcowa integracja sprzętu i oprogramowania z pełną prędkością korzysta z trybu CPU+FPGA. Osiągnięte wyniki szybkości symulacji potwierdzają, że opisany system może konkurować z drogimi rozwiązaniami firm komercyjnych [70][164].

5.2. Realizacja aplikacji Partitioner

W ramach realizacji aplikacji służącej do wykonania automatycznego podziału zaprojektowano:

- specyfikację formatu danych wejściowych i wyjściowych,
- przetwarzanie danych wejściowych do formatu wewnętrznego i sformułowanie zadania,
- implementację algorytmu B&B,
- przetwarzanie danych z formatu wewnętrznego do formatu danych wyjściowych.

Zaprojektowano także generator losowych grafów DAG do testowania aplikacji. Aplikacje Partitioner oraz DAG Generator zaimplementowano następnie w języku C++ przy użyciu środowisko VisualStudio 2005.

5.2.1. Przyjęte założenia

Danymi wejściowymi do aplikacji Partitioner są:

- topologia grafu – opisują zadania oraz sposób ich komunikacji,
- atrybuty grafu – zawierają zbiory parametrów uzyskane z implementacji zadań w środowiskach HW i SW, lub z bezpośredniej symulacji,
- ograniczenia parametrów – opisują ograniczenia fizycznej platformy lub wymagania, które system musi spełniać,
- funkcja celu – określa liniową kombinację parametrów optymalizowanego systemu,
- parametry algorytmu B&B – umożliwia wykonanie eksperymentów dotyczących optymalizacji działania samego algorytmu B&B.

Danymi wyjściowymi z aplikacji Partitioner są:

- przydział zadań do konkretnej realizacji i konkretnego zasobu – opisuje alokację (ang. *who*) i odwzorowanie (ang. *how*),
- kolejkowanie – definiuje czasy rozpoczęcia i zakończenia zadań (ang. *when*),
- parametry rozwiązania – wartość parametrów i funkcji celu dla systemu,
- statystykę działania algorytmu B&B – czas obliczeń, ilość przeanalizowanych podproblemów, ilość odrzuconych podproblemów, itp.

Aplikacja Partitioner rozwiązuje problem podziału i generuje dane wyjściowe dla ustalonej architektury sprzętowej.

Aplikacja umożliwia podział systemów wielkości do 100 komponentów, co znajduje zastosowanie w praktycznych projektach. Badania miały na celu określić limit praktycznej przydatności zastosowanego podejścia [10]. W literaturze spotyka się pojęcie „rozsądnego czasu obliczeń” (ang. *reasonable computation time*), które w różnych pracach jest różnie definiowane – od 30 minut [52] do 24 godzin [57]. W praktyce ten czas można wydłużyć nawet do kilku dni, biorąc pod uwagę, że podziału systemu dokonuje się raz na początku procesu projektowania.

5.2.2. Reprezentacja DAG

Graf DAG według zależności (2) jest reprezentowany przez zbiór wierzchołków i zbiór krawędzi. Każdy wierzchołek posiada informację o poprzednikach, następnikach oraz o zbiorze implementacji zadania modelowanego danym wierzchołkiem. Każda krawędź posiada informację o wierzchołkach incydentnych oraz o zbiorze implementacji komunikacji modelowanej daną krawędzią.

Aby umożliwić współpracę projektowanego systemu z zewnętrznymi narzędziami edycji grafów należy wykorzystać jeden ze standardowych tekstowych formatów

plików. Format powinien umożliwiać bezpośrednią edycję grafu, zapis dowolnego grafu, definiowanie wierzchołków i krawędzi oraz nadanie dowolnych atrybutów wierzchołkom i krawędziom. Do tego celu stosuje się powszechnie następujące formaty plików [143]:

- GML [117] – opis strukturalny z atrybutami w formacie 'klucz wartość', specyfikuje tzw. bezpieczne atrybuty,
- Dot [99] – opis z użyciem słów kluczowych, operatorów, podobnych do języków wysokiego poziomu,
- GDL [143] – opis strukturalny z atrybutami w formacie 'klucz wartość' – podobny do GML,
- GraphML [44] – rozszerzenie języka XML.

Po analizie wybrano format GML, którego syntaktyka ułatwia konstrukcję parsera, prosta semantyka umożliwia edycję grafu w edytorze tekstowym, a bezpieczne atrybuty umożliwiają przetwarzanie specyfikacji przez narzędzia edycji grafów.

5.2.3. Generator DAG

Do badania algorytmów podziału HW/SW potrzebna jest duża ilość przypadków testowych. Jak dotąd nie opracowano powszechnie akceptowanego zestawu testów wydajnościowych pozwalających porównać jakość rozwiązań i szybkość obliczeń poszczególnych metod podziału [170]. Dla potrzeb podziału układów VLSI, algorytmy są ewaluowane przez benchmark ISPD1998 opublikowany przez IBM [200]. Do celów testowania aplikacji Partitioner zaimplementowano aplikację DAG Generator, której działanie jest podobne do generatora TGFF [77]. W literaturze opisywane są różne metody generacji losowych grafów DAG dla różnych zastosowań, np. topologia grafów [205], kolejkowanie zadań [5][77][120], wizualizacja grafów [182][183], sieci Bayesa [126][127]. Generację losowych grafów DAG stosowanych do testowania problemu podziału HW/SW opisano w pracach [85][86][121][122][139][231][272]. Działanie algorytmów podziału zależy od topologii i własności generowanych grafów, które powinny być dopasowane do dziedziny aplikacyjnej [131]. Losowo generowane grafy DAG powinny się charakteryzować następującymi własnościami:

- graf jest spójny,
- istnieje dokładnie ściśle określona liczba wierzchołków początkowych (ang. *source vertex*) v_{in} i końcowych (ang. *sink vertex*) v_{out} [131],
- każda krawędź skierowana rozpoczyna się w wierzchołku o niższym numerze i kończy w wierzchołku o wyższym numerze [182] – graf jest topologicznie uporządkowany [205].

W aplikacji DAG Generator zaimplementowano 4 algorytmy: łańcucha Markowa, heurystyczny, minimalnego drzewa spinającego i przestrzeni prawdopodobieństwa.

5.2.3.1. Algorytm łańcucha Markowa

Algorytm łańcucha Markowa oparto na pracach [76][126][127][182][183], gdzie przestrzenią stanów są wszystkie DAG nad zbiorem wierzchołków **V**. Pozycja zawiera losowo wybraną uporządkowaną parę (i, j) różnych wierzchołków V. Aby uzyskać łańcuch Markowa należy zapewnić [126][127]:

- nieredukowalność – jeśli z dowolnego grafu można wygenerować inny dowolny graf,
- nieokresowość – jeśli istnieje prawdopodobieństwo uzyskania tego samego grafu w kolejnym kroku.

W związku z tym funkcja przejścia stanu X(t) do stanu X(t+1) jest zdefiniowana następująco [182]:

- jeśli pozycja (i, j) odpowiada krawędzi e istniejącej w X(t), wówczas krawędź e jest usuwana z X(t):

$$X(t+1) = X(t) \setminus e \tag{26}$$

- jeśli pozycja (i, j) odpowiada krawędzi e nieistniejącej w X(t), wówczas krawędź e jest dodawana do X(t), pod warunkiem, że nie tworzy cyklu:

$$X(t+1) = X(t) + e \tag{27}$$

w przeciwnym przypadku:

$$X(t+1) = X(t) \tag{28}$$

Przyjmuje się, że łańcuch Markowa stabilizuje się po n^2 przejściach stanu [182]. Stanem początkowym może być graf pusty [76][182], lub graf o pożądanych własnościach, np. drzewo [126]. Funkcja przejścia stanu może sprawdzać zachowanie cech generowanego grafu np. planarność po dodaniu krawędzi [76], spójność grafu po usunięciu krawędzi według zależności (26) [126] lub grubość grafu [127].

Generowane grafy o n wierzchołkach mają średnio n(n-1)/4 krawędzi, średni stopień wierzchołka wynosi n/2 (suma stopni wejściowego i wyjściowego), średnia liczba poziomów wynosi 2n/3, średnia liczba wierzchołków na danym poziomie wynosi 3/2 [182]. W aplikacji DAG Generator dodano następujące modyfikacje:

- zapewnienie poprzednika dla każdego wierzchołka oprócz początkowych v_{in},
- zapewnienie następnika dla każdego wierzchołka oprócz końcowych v_{out},
- krawędzie pomiędzy wierzchołkami początkowymi oraz pomiędzy końcowymi nie są dodawane (zamiast zależności (27) stosuje się zależność (28)) – zwiększenie nieokresowości,
- zamiana pozycji (i, j) na (j, i) jeśli i > j [182],
- kontrola gęstości grafu poprzez ograniczenie liczby generowanych krawędzi [126][127], co zmienia rozkład losowy grafów DAG i ich właściwości, ale umożliwia uzyskanie grafów o pożądanej gęstości.

5.2.3.2. Algorytm heurystyczny

Algorytm heurystyczny oparto na pracy [5], gdzie danymi wejściowymi dla generatora są: n_1 - maksymalna liczba poziomów w grafie, n_2 - maksymalna liczba wierzchołków na danym poziomie, d – współczynnik stopnia wierzchołka. Na podstawie danych wejściowych generowane są: liczba poziomów $p \in (0, n_1]$, liczba wierzchołków $v_i \in (0, n_2]$ dla i-tego poziomu oraz liczba krawędzi $d_{ij} \in [0, v_{i+1} * d]$ z j-tego wierzchołka na i-tym poziomie do losowo wybranych wierzchołków na poziomie i+1. Algorytm generuje grafy o stosunkowo dużej liczbie wierzchołków na danym poziomie, co sprzyja zrównoleglaniu zadań. W aplikacji DAG Generator dodano następujące modyfikacje:

- przyjęto $v_0 = v_{in}, v_{p-1} = v_{out}$,
- zwiększono o 1 liczbę krawędzi d_{ij},
- dodano krawędź z losowo wybranego wierzchołka na poziomie i-1 do wierzchołka na poziomie i, jeśli jego stopień wejściowy wynosił 0.

5.2.3.3. Algorytm minimalnego drzewa spinającego

Algorytm minimalnego drzewa spinającego (ang. *Minimum Spanning Tree*) oparto na pracy [120], gdzie DAG jest tworzony przez generację MST algorytmem Prima z grafu pełnego, którego krawędzie otrzymały losowe wagi. Następnie dodawane są losowo krawędzie skierowane do niższych warstw, aż do pożądanej gęstości grafu. W wyniku otrzymuje się grafy o rozkładzie bliskim losowemu, przy czym istnieje co najmniej 1 wierzchołek początkowy i końcowy. W aplikacji DAG Generator dodano następujące modyfikacje:

- inicjalnie generowany jest graf pełny bez krawędzi pomiędzy wierzchołkami początkowymi oraz pomiędzy końcowymi,
- zapewnienie poprzednika dla każdego wierzchołka oprócz początkowych v_{in},
- zapewnienie następnika dla każdego wierzchołka oprócz końcowych v_{out}.

5.2.3.4. Algorytm przestrzeni prawdopodobieństwa

Algorytm przestrzeni prawdopodobieństwa (ang. *probability space*) zaczerpnięto z pracy [205], w którym model $G_{dag}(v, p)$ jest przestrzenią prawdopodobieństwa zawierającą wszystkie grafy mające zbiór wierzchołków V = {1, 2, …, v} i zbiór krawędzi $E \subseteq \{ (i, j) \mid i < j \}$. Każda krawędź istnieje w grafie z prawdopodobieństwem p niezależnie od innych, przy czym p = e / 0.5*v*(v-1) determinuje liczbę krawędzi w grafie. W aplikacji DAG Generator dodano następujące modyfikacje:

- zapewnienie poprzednika dla każdego wierzchołka oprócz początkowych v_{in},
- zapewnienie następnika dla każdego wierzchołka oprócz końcowych v_{out}.

5.2.4. Definicja funkcji ograniczenia dolnego

Jednym z najbardziej istotnym czynnikiem do efektywnego znalezienia rozwiązania optymalnego przy użyciu metody B&B jest określenie funkcji ograniczenia dolnego LB, która umożliwia eliminację nierokujących gałęzi drzewa poszukiwań na wczesnym etapie. Funkcja LB estymuje minimalną wartość funkcji celu dla rozwiązań składających się ze znanych implementacji przydzielonych zadań oraz z dowolnych implementacji nieprzydzielonych zadań. Podstawowym problemem podczas estymacji minimalnej wartości funkcji celu jest to, że zawiera nieaddytywne metryki (np. czas wykonania). Czas wykonania jest wyznaczany przez algorytm kolejkowania, który bierze pod uwagę zależności pomiędzy zadaniami oraz architekturę docelowej platformy określając czas rozpoczęcia i zakończenia zadania.

Poniżej zdefiniowano funkcję LB z zerowym czasem wykonania wszystkich nieprzydzielonych zadań. W tym przypadku kolejkowanie zadań może zostać pominięte i wartość czasu wykonania dla zadań przydzielonych może być przyjęta jako dolne ograniczenie czasu wykonania wszystkich zadań. Dla każdego nieprzydzielonego zadania wybierana jest implementacja z minimalną wartością funkcji celu (przy założeniu zerowego czasu wykonania). Następnie obliczane są parametry rozwiązania przy użyciu metryk addytywnych. Opisane podejście definiuje funkcję LB.

Oznaczenia:

x – zadanie

N – liczba zadań

M_n – liczba implementacji n-tego zadania, $n \in \{1,\ldots,N\}$

$x_n^{m_n}$ – m-ta implementacja n-tego zadania, $m \in \{1,\ldots,M_n\}$, $n \in \{1,\ldots,N\}$

$X = \{x_1^{m_1},...,x_n^{m_n},...,x_N^{m_N}\}$ – kompletne rozwiązania

P – zbiór indeksów dla przydzielonych zadań
X_P – rozwiązanie częściowe, w którym niektóre zadania są nieprzydzielone
R – liczba parametrów funkcji celu, niech ostatni będzie nieaddytywnym czasem wykonania
a_r – wartość r-tego współczynnika funkcji celu, $r \in \{1,\ldots,R\}$, $a_r \geq 0$
$p_{r,n}^{m_n}$ – wartość r-tego parametru m-tej implementacji n-tego zadania, $p \geq 0$

$f(x_n^{m_n}) = \sum_{r=1}^{R} a_r \cdot p_{r,n}^{m_n}$ – wartość funkcji celu dla m-tej implementacji n-tego zadania

$P_{r,P} = \sum_{n \in P} p_{r,n}^{m_n}$ – wartość r-tego parametru rozwiązania częściowego (dla parametrów addytywnych)

$P_r = \sum_{n=1}^{N} p_{r,n}^{m_n}$ – wartość r-tego parametru rozwiązania całkowitego (dla parametrów addytywnych), $P_{r,P} \leq P_r$

$f(X) = \sum_{r=1}^{R} a_r \cdot P_r$ – wartość funkcji celu dla rozwiązania całkowitego, wywiedzionego z rozwiązania częściowego X_P

$f(X_P) = \sum_{r=1}^{R} a_r \cdot P_{r,P}$ – wartość funkcji celu dla rozwiązania częściowego X_P

$g(X_P) = f(X_P) + \sum_{n \notin P} \min_{m=\{1,\ldots,M_n\}} f(x_n^{m_n}) \big| (a_R = 0)$ – wartość funkcji LB dla rozwiązania całkowitego wywiedzionego z rozwiązania częściowego X_P.

Twierdzenie: Zaproponowana funkcja jest LB, tzn. $g(X_P) \leq f(X)$

Dowód:

$$g(X_P) \overset{g(X_P)}{=} f(X_P) + \sum_{n \notin P} \min_{m=\{1,\ldots,M_n\}} f(x_n^{m_n}) \big| (a_R = 0) \overset{f(X_P)}{=}$$

$$\sum_{r=1}^{R} a_r \cdot P_{r,P} + \sum_{n \notin P} \min_{m=\{1,\ldots,M_n\}} f(x_n^{m_n}) \big| (a_R = 0) \overset{P_{r,P} \& f(x_n^{m_n})}{=}$$

$$\sum_{r=1}^{R-1} a_r \cdot \sum_{n \in P} p_{r,n}^{m_n} + a_R \cdot P_{R,P} + \sum_{n \notin P} \min_{m=\{1,\ldots,M_n\}} \sum_{r=1}^{R-1} a_r \cdot p_{r,n}^{m_n} \overset{\min}{\leq}$$

$$\sum_{r=1}^{R-1} \sum_{n \in P} a_r \cdot p_{r,n}^{m_n} + a_R \cdot P_{R,P} + \sum_{n \notin P} \sum_{r=1}^{R-1} a_r \cdot p_{r,n}^{m_n} \overset{P_{R,P} \leq P_R}{\leq}$$

$$\sum_{r=1}^{R-1} \sum_{n \in P} a_r \cdot p_{r,n}^{m_n} + a_R \cdot P_R + \sum_{r=1}^{R-1} \sum_{n \notin P} a_r \cdot p_{r,n}^{m_n} =$$

$$\sum_{r=1}^{R-1} \sum_{n=1}^{N} a_r \cdot p_{r,n}^{m_n} + a_R \cdot P_R = \sum_{r=1}^{R-1} a_r \cdot P_r + a_R \cdot P_R = f(X)$$

5.2.5. Wyniki i wnioski z realizacji aplikacji Partitioner

Aplikacja Partitioner została przetestowana dla losowych grafów DAG o rozmiarach N<100 generowanych z aplikacji DAG Generator. Wyniki optymalizacji potwierdzono z rozwiązaniami uzyskanymi za pomocą bibliotek MILP Solver

dających rozwiązanie optymalne. Wymagało to skonwertowania problemu zdefiniowanego dla metody B&B na problem zdefiniowany dla metody MILP.

Profilowanie aplikacji Partitioner wykazało, że najwięcej czasu zajmują obliczenia funkcji celu dla węzła dodanego do listy *live* z uwagi na konieczność wyznaczenia kolejkowania przydzielonych węzłów. Problem ten rozwiązano przy pomocy metody, która dodając nowe zadanie heurystycznie modyfikuje bieżące kolejkowanie. Dodano także możliwość serializacji listy *live* do pliku i deserializację z pliku, jeśli istnieje potrzeba przerwania obliczeń i ich wznowienia.

Aplikacja Partitioner wraz z DAG Generator są doskonałymi narzędziami do przeprowadzania eksperymentów na grafach DAG, do analizy wpływu parametrów i ograniczeń na decyzje podejmowane przy podziale. Aplikacja Partitioner pozwala na podawanie parametrów metody B&B dotyczących strategii iteracji, strategii rozgałęzień i strategii wyboru podproblemu. Dobór tych parametrów umożliwia przyspieszenie działania metody B&B, co opisano w rozdziale 5.3.

5.3. Analiza wrażliwości i automatyczny dobór parametrów metody B&B

Analiza wrażliwości metody optymalizacji ma na celu sprawdzenie:

- wpływu wartości danych wejściowych na wartość rozwiązania optymalnego [57] - odpowiada na pytanie dotyczące dokładności ekstrakcji parametrów wejściowych;
- wpływu parametrów metody optymalizacji na czas działania metody - ma na celu sprawdzenie, które parametry należy dobierać, aby skrócić czas obliczeń.

Przy projektowaniu z wykorzystaniem gotowych modułów dostępne są dokładne informacje o wartościach parametrów, z wykorzystaniem których odbywa się dalsza optymalizacja, nie wykonuje się więc analizy wrażliwości wpływu wartości danych wejściowych na wartość rozwiązania optymalnego. Istotne jest badanie wpływu parametrów na czas działania metody. Dla B&B parametry mają charakter decyzyjny.

Dobór parametrów jest stosowany w wielu metodach optymalizacyjnych. W pracy [170] opisano strojenie parametrów algorytmu SA. W pracy [160] przedstawiono autokalibrację parametrów metody TS, dzięki której osiąga dużą wydajność. W pracy [139] zaprezentowano statyczny dobór parametrów algorytmu MIBS.

Automatyczny dobór parametrów metody B&B bazujący na dynamicznej analizie wrażliwości przedstawiono szerzej w pracy [251]. Analizę wpływu funkcji ograniczenia dolnego w zastosowaniu dla sieci ATM zawiera praca [266].

5.3.1. Analiza wrażliwości parametrów

Ograniczenia mają istotne znaczenie dla procesu obliczeń, gdyż umożliwiają odrzucenie gałęzi, które nie prowadzą do możliwych rozwiązań. W celu zmniejszenia przestrzeni poszukiwań warto dawać jak najściślejsze ograniczenia. Stosuje się następujące metody przetwarzania ograniczeń [154]:

- usuwanie zbędnych ograniczeń,
- zacieśnianie ograniczeń,
- uzyskiwanie rozwiązania bezpośrednio z analizy ograniczeń,
- redukcja liczby implementacji – jeśli istnieją implementacje tego samego typu (SW lub HW) o lepszych wartościach wszystkich parametrów,
- szacowanie stopnia ograniczeń przez wyznaczenie stosunku ograniczenia do wartości ekstremalnej [169].

Wartość rozwiązania początkowego ma istotne znaczenie dla czasu obliczeń. Rozwiązanie początkowe bliskie optymalnemu może znacznie przyspieszyć proces obliczeniowy poprzez odrzucanie gałęzi w przestrzeni poszukiwań tak szybko jak to tylko możliwe, jeszcze przed znalezieniem pierwszego dopuszczalnego rozwiązania. W pracy [62] wykazano, że jeśli rozwiązanie optymalne jest podane jako rozwiązanie początkowe, wówczas przy strategii wyboru DFS metoda analizuje tylko podproblemy krytyczne. W strategii wyboru BFS wartość rozwiązania początkowego najsilniej determinuje czas obliczeń. W tym przypadku odrzucanie rozwiązań częściowych odbywa się na podstawie wartości rozwiązania początkowego, a dopiero potem generowane są rozwiązania całkowite. W strategii wyboru BeFS czas obliczeń słabiej zależy od wartości rozwiązania początkowego. W pracy [52] pokazano, że rozwiązanie początkowe uzyskiwane metodą metryki odpowiedniości do implementacji w HW było średnio 8,4% gorsze (a maksymalnie 27% gorsze) od optymalnego.

Obie strategie iteracji (gorliwa i leniwa) mają jednakową złożoność obliczeniową, różnią się jednak nieznacznie czasem obliczeń. Strategia leniwa dodaje więcej

podproblemów do zbioru *live*, lecz dokonuje ich sprawdzania w późniejszym czasie. Jeśli w międzyczasie uzyska się lepsze rozwiązanie, podproblemy mogą zostać odrzucone [32].

Strategia wyboru podproblemu odzwierciedla kompromis pomiędzy małym rozmiarem listy *live* a maksymalnym wykorzystaniem dostępnej pamięci. Kolejność wyboru podproblemu do analizy ma istotne znaczenie dla czasu obliczeń:

- BFS charakteryzuje się największym zużyciem pamięci, gdyż wymaga dodania do listy wszystkich podproblemów krytycznych na raz, oraz największym czasem obliczeń. Największy udział ma tutaj operacja dodawania do listy i usuwania z listy. Jeśli ponadto nie dostarczymy rozwiązania początkowego, wówczas do listy zostaną dodawane wszystkie podproblemy, co spowoduje wykładnicze zużycie pamięci.
- BeFS preferuje wybór podproblemów bliższych korzeniowi drzewa. Podproblemy bliższe korzeniowi drzewa nie są z reguły dobrze szacowane ani przez funkcję ograniczenia dolnego, ani przez funkcję kontroli ograniczeń, więc zazwyczaj generowane są kolejne podproblemy i dodawane do listy. BeFS charakteryzuje się nieco mniejszym zużyciem pamięci, gdyż wymaga dodania do listy wszystkich problemów pośrednich, ale są one usuwane. Czas obliczeń jest krótszy niż w metodzie BFS. Po znalezieniu optymalnego rozwiązania na liście *live* nie ma podproblemów krytycznych, więc w strategii gorliwej nie jest już obliczana żadna wartość funkcji ograniczenia dolnego, a jedynie dowodzi się optymalności znalezionego rozwiązania. W strategii leniwej po znalezieniu optymalnego rozwiązania obliczane są wartości funkcji ograniczenia dolnego. W przypadku dużej ilości podproblemów krytycznych strategia BeFS cechuje się dużym zużyciem pamięci.
- Strategia szukania wgłąb DFS charakteryzuje się najmniejszym zużyciem pamięci, gdyż w danym momencie na liście znajduje się co najwyżej n*k (lub suma po i $\leq$ n wszystkich k_i) podproblemów. W związku z tym operacje na pamięci wykonują się stosunkowo szybko. Strategia ta jest w najmniejszym stopniu zależna od wartości rozwiązania początkowego, gdyż najszybciej z omawianych strategii znajduje pierwsze rozwiązanie dopuszczalne, które może być traktowane jako rozwiązanie początkowe. Wadą strategii DFS jest wykonywanie dużej ilości obliczeń ograniczenia dolnego, jeśli wartość rozwiązania początkowego jest daleka od optimum. DFS może być wydajnie używane ze strategią gorliwą i leniwą i umożliwia użycie prostej rekursji.

Strategia rozgałęzień ma istotne znaczenie dla czasu obliczeń. Dobra kolejność analizy zadań może znacznie przyspieszyć optymalizację [57]. Ogólną ideą jest uporządkowanie ograniczeń i zmiennych w ten sposób, aby testować najpierw najbardziej restrykcyjne z nich. Umożliwi to wczesne odrzucenie węzła.

5.3.2. Przyjęte założenia

Eksperymenty przeprowadzono przy użyciu aplikacji Partitioner i DAG Generator. Zbadano wpływ poszczególnych parametrów metody B&B na czas obliczeń dla losowych grafów. Przebadano zależność czasu obliczeń od rozwiązania początkowego, strategii iteracji, strategii rozgałęzień i strategii wyboru podproblemu. Za pomocą aplikacji DAG Generator wygenerowano losowe grafy o rozmiarach N=19, 25, 30 wierzchołków i z użyciem aplikacji Partitioner przeprowadzano optymalizację wygenerowanych problemów przy ustalonych wszystkich parametrach i zmieniającym się jednym parametrze.

Z przedstawionej analizy wrażliwości wynika, że największy wpływ na czas obliczeń metody B&B mają strategia wyboru podproblemu, strategia rozgałęzień oraz rozwiązanie początkowe.

Podstawowym problemem automatycznego doboru parametrów jest wyznaczenie spodziewanego czasu obliczeń metody B&B. Eksperymenty wykazały, że zależność liczby przebadanych węzłów w funkcji czasu nie jest liniowa, ani nie obserwuje się żadnych wyraźnych trendów. Wobec tego nie da się przewidzieć dokładnego czasu obliczeń, ani zagwarantować najkrótszego czasu obliczeń dla wybranej kolejności. Różnica czasu działania metody B&B powinna być większa niż czas przeznaczony na automatyczny dobór parametrów. Dąży się zatem do maksymalizacji wyrażenia:

$$\frac{T_{obl}(P_0)}{T_{obl}(P_{opt})+T_{opt}} \tag{29}$$

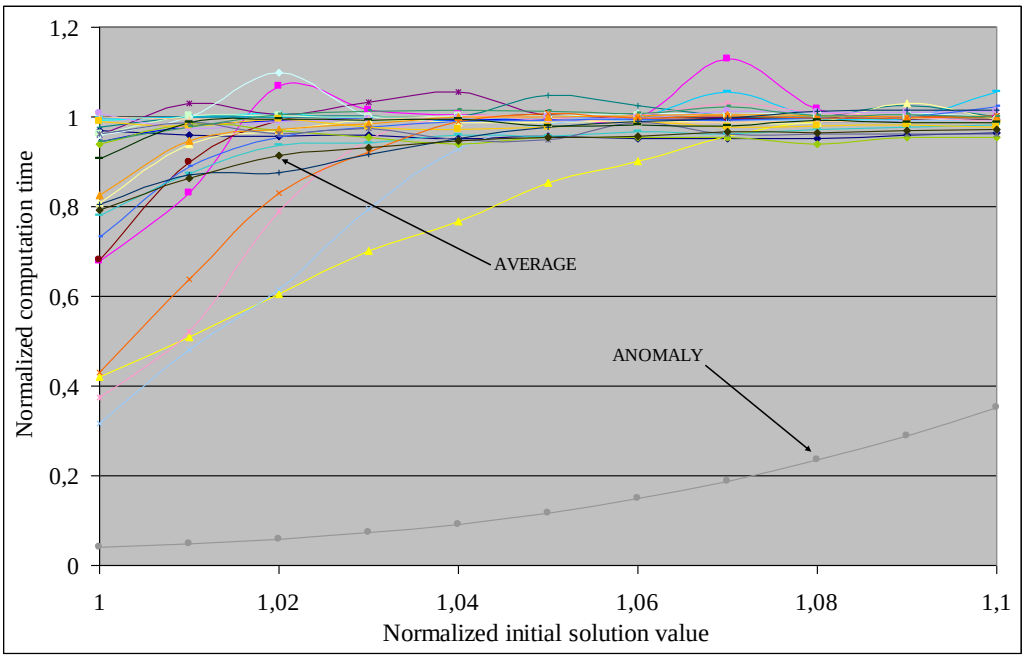
gdzie:

T_{obl} – spodziewany czas obliczeń metody B&B dla zbioru parametrów,
P_0 – zbiór parametrów początkowych,
P_{opt} – zbiór parametrów otrzymanych w wyniku optymalizacji,
T_{opt} – czas automatycznego doboru parametrów.

Strategię wyboru podproblemu oraz strategię rozgałęzień dobierano algorytmem symulowanego wyżarzania SA. W kolejnych iteracjach stopniowo zwiększano czas działania algorytmu B&B w celu osiągnięcia lepszych estymat czasu działania.

Rozwiązanie początkowe znajdowano również algorytmem symulowanego wyżarzania SA. Parametry dobrano tak, aby czas działania algorytmu SA nie przekraczał 1% estymowanego czasu działania metody B&B.

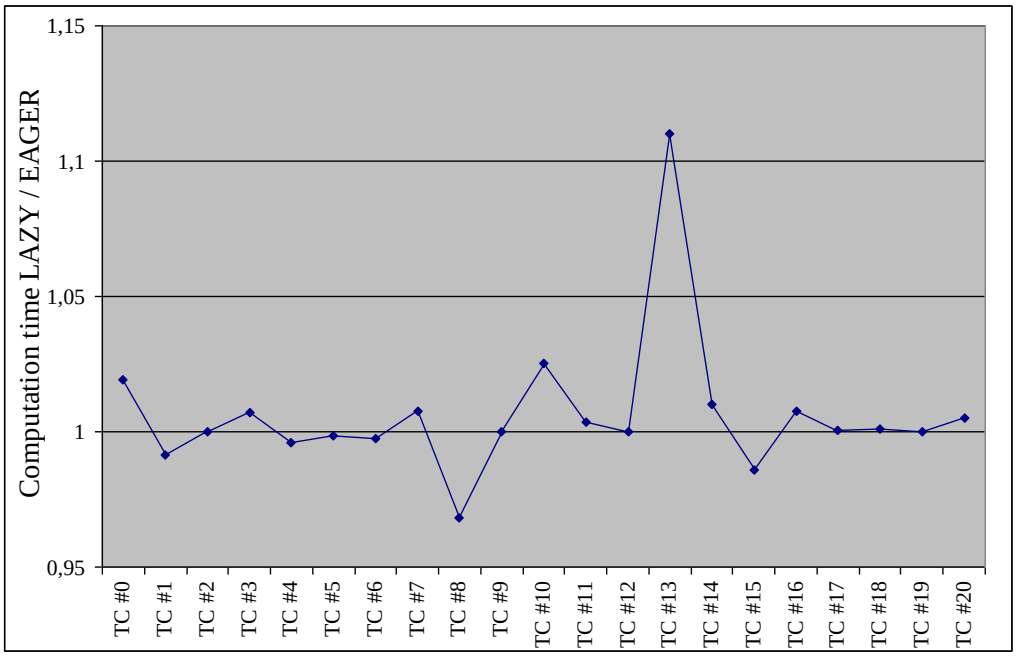
5.3.3. Wyniki badania parametrów metody B&B



Rysunek 7. Zależność czasu obliczeń od jakości rozwiązania początkowego dla n=25

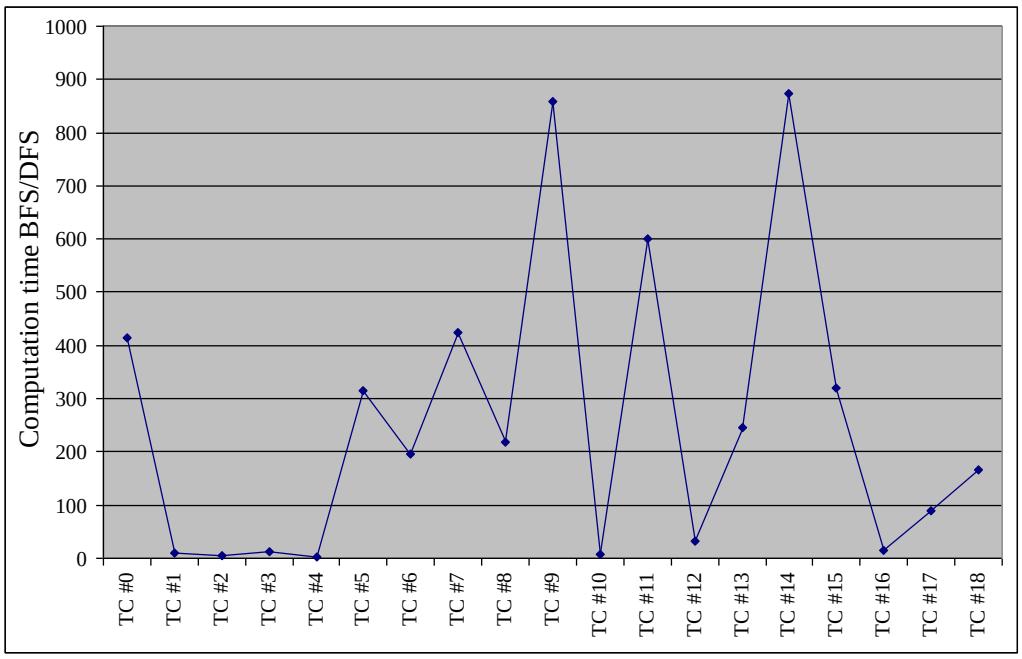
Rysunek 7 przedstawia wyniki zależności czasu obliczeń od jakości rozwiązania początkowego. Użycie rozwiązania optymalnego jako rozwiązania początkowego w strategii wyboru DFS dla rozmiaru problemu n = 25 powoduje średnio skrócenie czasu obliczeń o 20%. W celu osiągnięcia istotnego przyspieszenia obliczeń, rozwiązanie początkowe powinno być nie więcej niż 1-2% gorsze niż optymalne.

Badania doświadczalne wykazały istnienie przypadków szczególnych, dla których rozwiązanie początkowe silnie wpływa na czas obliczeń. W tych przypadkach pierwsze rozwiązanie dopuszczalne jest jednocześnie optymalnym i jest znajdowane dopiero przy końcu procesu poszukiwań. Ustalenie rozwiązania początkowego powoduje odrzucanie większej ilości gałęzi na wcześniejszym etapie poszukiwań.

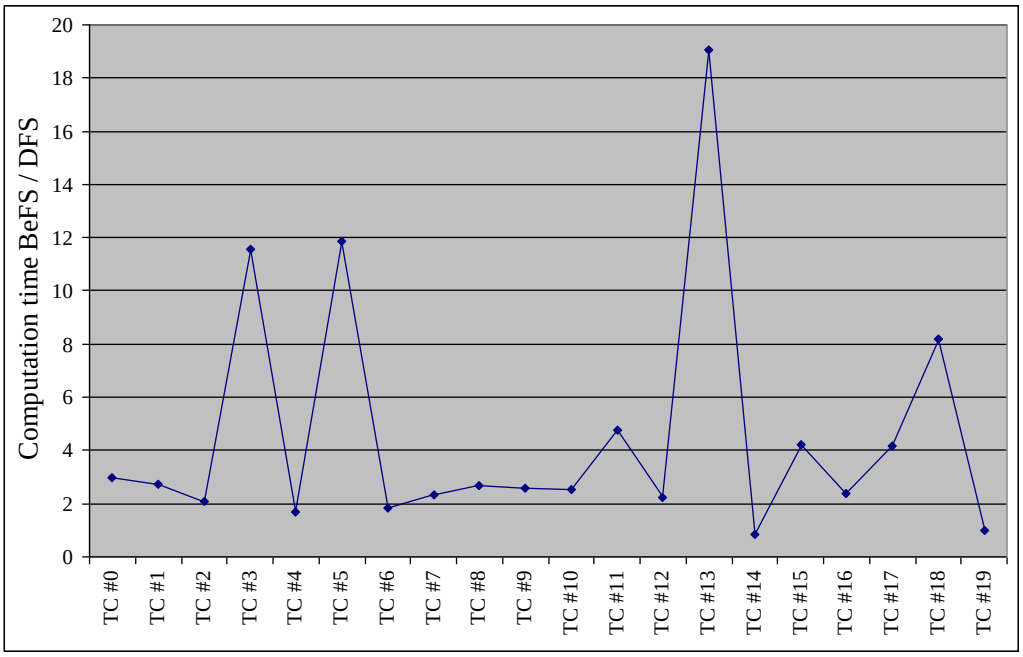


Rysunek 8. Stosunek czasu obliczeń w strategiach iteracji leniwej i gorliwej

Rysunek 8 przedstawia wyniki stosunku czasu obliczeń w strategiach iteracji leniwej i gorliwej. Obie strategie mają niemal ten sam czas obliczeń. Strategia gorliwa dla rozmiaru problemu n=25 okazała się lepsza o 0,6% w średnim 11% w maksymalnym przypadku. Z uwagi na fakt, iż zmiana strategii iteracji nie owocuje znaczącym przyspieszeniem czasu obliczeń, nie ma potrzeby automatycznego doboru tego parametru. W pozostałych eksperymentach używano strategii gorliwej.

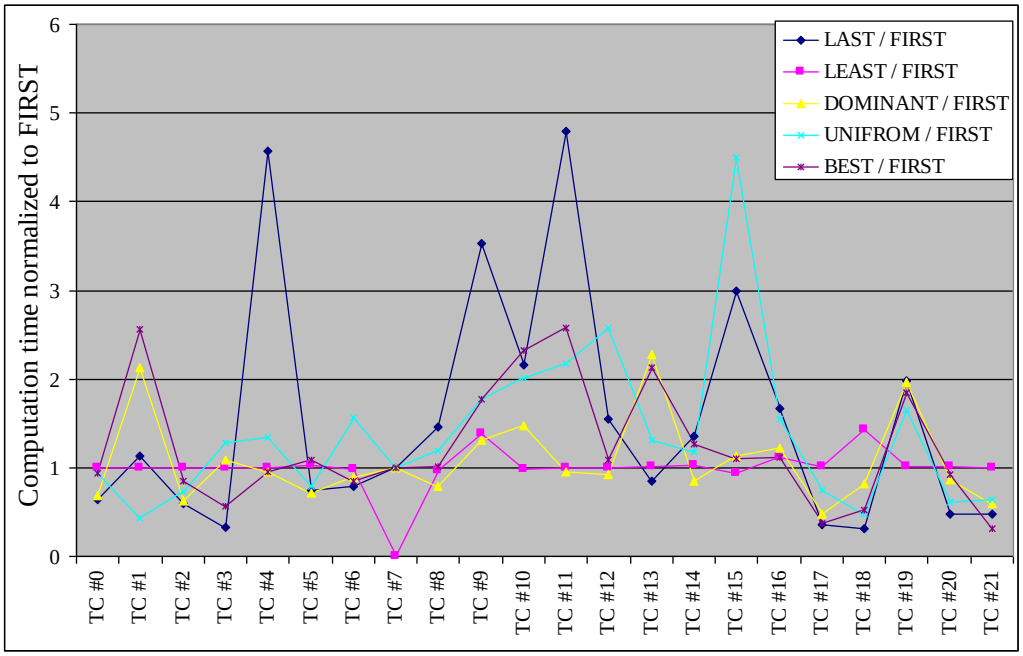


Rysunek 9. Czas obliczeń w strategii wyboru podproblemu BFS w odniesieniu do DFS



Rysunek 10. Czas obliczeń w strategii wyboru podproblemu BeFS w odniesieniu do DFS

Rysunek 9 i Rysunek 10 przedstawiają wyniki czasu obliczeń strategiach BFS i BeFS w odniesieniu do DFS. Strategia BFS w najgorszym przypadku jest niemal 900 razy wolniejsza, a BeFS odpowiednio 20 razy wolniejsza. Ponieważ strategia DFS jest znacząco szybsza dla przebadanego zbioru grafów DAG, w pozostałych eksperymentach używano strategii DFS.



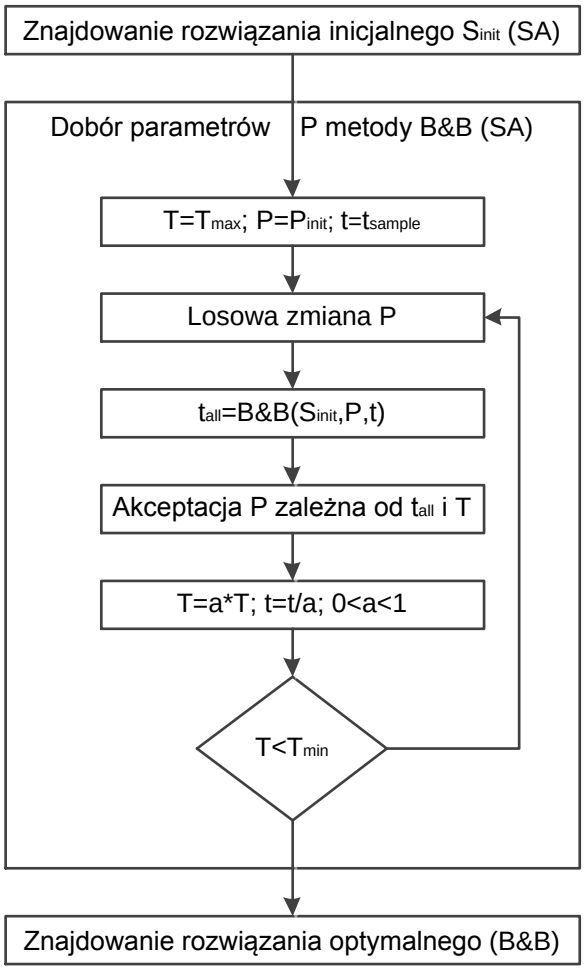
Rysunek 11. Czas obliczeń w różnych strategiach rozgałęzień w odniesieniu od FIRST

Rysunek 11 przedstawia wyniki czasu obliczeń w różnych strategiach rozgałęzień. Badania dowiodły, że dla rozmiaru problemu n=30, czas obliczeń może różnić się nawet 15-krotnie w zależności od kolejności analizy zadań. Nie ma strategii wyraźnie dominującej, choć statystycznie najlepsze wyniki osiąga się przy strategii FIRST. Strategia rozgałęzień może być parametrem podlegającym automatycznemu strojeniu w metodzie B&B.

5.3.4. Automatyczny dobór parametrów metody B&B

Przeprowadzono badania z wykorzystaniem automatycznego doboru parametrów metody B&B. Pomierzono inicjalny czas działania metody B&B przy standardowych parametrach P_{init}: bez rozwiązania początkowego, strategia iteracji gorliwa, strategia wyboru podproblemu DFS i strategia rozgałęzień FIRST. Rysunek 12 i

Algorytm 16 przedstawiają schemat automatycznego doboru parametrów metody B&B. Zastosowano algorytm SA do szybkiego uzyskania rozwiązania początkowego S_{init}, który stanowił punkt startowy do kolejnych optymalizacji metodą B&B. Pierwsze uruchomienie metody B&B odbywało się na czas $t=t_{sample}$ z losowo zmienionymi parametrami P, po czym na podstawie przebadanej przestrzeni projektowej estymowano całkowity czas t_{all} potrzebny do obliczeń. Parametry akceptowane były w zależności od estymowanego czasu obliczeń t_{all} i bieżącej temperatury T według schematu Boltzmanna. Kolejna iteracja algorytmu SA schładzała temperaturę T, aby stopniowo ograniczać akceptację gorszych rozwiązań, ale jednocześnie wydłużała czas t działania metody B&B, aby uzyskiwać bardziej wiarygodne estymaty. Po osiągnięciu zadanej liczby iteracji (badano krótkie cykle wyżarzania 10-12 iteracji, każdy trwający 1-10 sekund), optymalizację wznawiano przy użyciu najlepszego znalezionego zestawu parametrów metody B&B. Przy zastosowaniu serializacji do pliku i deserializacji z pliku możliwe było wznowienie działania metody B&B z miejsca, w którym została ona przerwana. Parametry T_{max}, T_{min}, a, t były dobierane w zależności od rozmiaru problemu i czasu trwania metody B&B.



Rysunek 12. Schemat doboru parametrów metody B&B

Algorytm 16. Schemat doboru parametrów metody B&B

1. Znajdowanie rozwiązania inicjalnego S_{init} metodą symulowanego wyżarzania;
2. Dobór parametrów metodą symulowanego wyżarzania:

- Inicjalizacja: $T=T_{max}$; $P=P_{init}$; $t=t_{sample}$;
- Losowa zmiana parametrów P
- Uruchomienie metody podziału i ograniczeń B&B z parametrami P, rozwiązaniem początkowym S_{init} na czas t_{sample} i estymacja całkowitego czasu obliczeń t_{all};
- Akceptacja parametrów P zależna od t_{all} i T
- Chłodzenie: T=a*T; t=t/a; 0<a<1

3. Znajdowanie rozwiązania optymalnego S_{opt} metodą podziału i ograniczeń z parametrami P_{opt};

Zaprezentowany powyżej schemat doboru parametrów zastosowano do rozwiązania 4 przypadków testowych o rozmiarze 30 węzłów wygenerowanych za pomocą DAG Generatora. Pomierzono czas działania metody B&B przy zastosowaniu parametrów inicjalnych oraz parametrów zoptymalizowanych algorytmem SA. Przyspieszenie wyznaczano za pomocą zależności (29). Tabela 6 przedstawia wyniki automatycznego doboru parametrów metody B&B dla 4 przypadków testowych. Eksperymenty pokazały, że w wyniku automatycznego doboru parametrów uzyskano w skrajnym przypadku niemal x300 przyspieszenie dla jednego z przypadków testowych, a w średnim przypadku przyspieszenie x130.

Tabela 6. Czas obliczeń metody B&B dla 4 przypadków testowych (30 węzłów)

Test	Czas metody B&B (inicjalny)	Czas metody B&B (zoptymalizowany)	Przyspieszenie
TC #0	9764,219 s	90,515 s	107,87
TC #1	813,422 s	28,469 s	28,57
TC #2	3139,625 s	10,625 s	295,49
TC #3	1464,250 s	12,375 s	118,32

5.3.5. Wnioski z badania parametrów metody B&B

Badanie wpływu parametrów metody B&B na czas obliczeń pozwoliło zidentyfikować kluczowe parametry i oszacować potencjał przyspieszenia metody.

Metody generacji rozwiązania początkowego powinny działać w bardzo krótkim czasie, czas przeznaczony na obliczenie dobrego rozwiązania początkowego powinien stanowić nie więcej niż kilka % ogólnego czasu obliczeń [62]. Rozwiązanie początkowe może być wyznaczone ogólnymi metodami heurystycznymi: symulowane wyżarzanie SA, algorytm genetyczny GA lub poszukiwanie tabu TS [62]. Wartości parametrów symulowanego wyżarzania zależą od rozmiaru grafu [85]. Zastosowanie symulowanego wyżarzania wymaga dobrania zestawu operatorów, które modyfikują numer implementacji dla każdego węzła oraz zastosowanie funkcji kary dla rozwiązań, które nie spełniają narzuconych ograniczeń.

Strategia wyboru podproblemu determinuje liczbę węzłów krytycznych. Jeśli podamy optymalne rozwiązanie początkowe, wówczas liczba węzłów krytycznych jest taka sama we wszystkich trzech metodach. Dla nieoptymalnego rozwiązania początkowego liczba węzłów krytycznych jest największa dla BFS, mniejsza dla BeFS a najmniejsza dla DFS.

Strategia rozgałęzień powinna zapewniać, że węzły krytyczne znajdują się jak najbliżej liści, natomiast węzły niekrytyczne jak najbliżej korzenia. Jeśli liczba rozgałęzień (liczba implementacji zadania) jest różna, to najmniejsze drzewo

powstaje, gdy najpierw umieszcza się węzły z najmniejszą liczbą rozgałęzień, a węzły z największą liczbą na końcu.

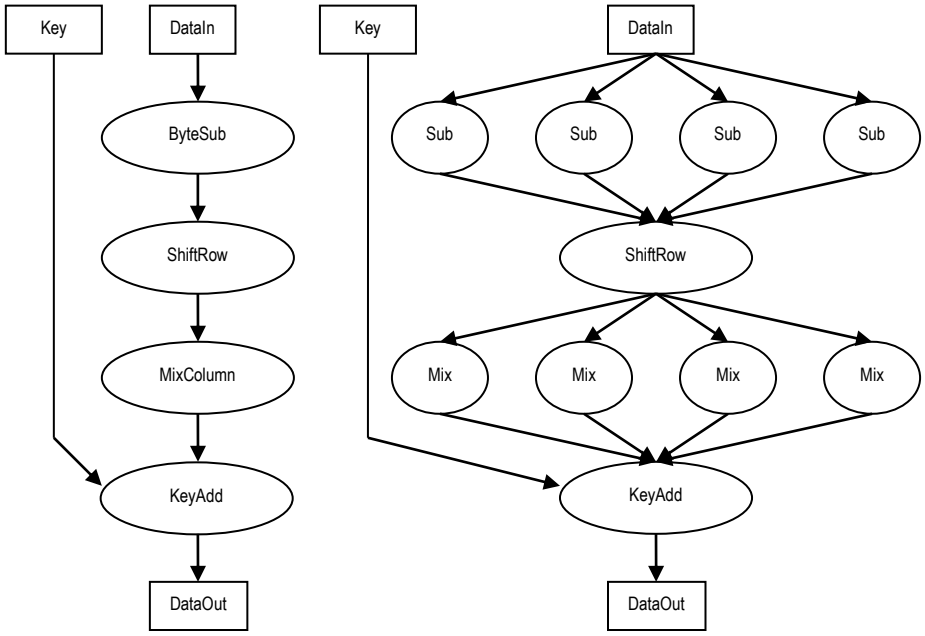
Jak pokazano na przykładach, automatyczny dobór parametrów może przyspieszać działanie metody B&B o czynnik x100. Czas działania metody nie jest czynnikiem ograniczającym zastosowanie metody do bardziej złożonych problemów.

5.4. Implementacja algorytmu kryptograficznego na heterogenicznej platformie SoC

Implementację algorytmów kryptograficznych na platformach SoC wykonał dyplomant w ramach pracy dyplomowej [72], tezy rozwijano we wspólnej pracy [248] oraz niezależnie w pracy [249].

5.4.1. Wybór algorytmu kryptograficznego

Najlepszym kandydatem do implementacji heterogenicznej jest algorytm, który zawiera moduły zarówno lepiej realizowane sprzętowo jak i lepiej realizowane programowo. Spośród analizowanych algorytmów takie właściwości posiada Twofish i MARS [46]. Ponadto algorytmy Rijndael i Twofish posiadają wiele możliwości kompromisu implementacji [257]. Z uwagi na powszechność zastosowania, do realizacji przyjęto algorytm kryptograficzny Rijndael, opisany szczegółowo w rozdziale 8.7. Zaimplementowano moduł szyfrowania ze 128-bitowym blokiem danych i 128-bitowym kluczem szyfru wraz z modułem generacji podkluczy rundy [248]. Sieć działań pojedynczej rundy można przedstawić za pomocą grafu DAG. Rysunek 13 pokazuje, że niektóre operacje można podzielić na kilka operacji operujących na mniejszej ilości danych. Wynikający stąd problem współdzielenia zasobów jest rozwiązany przez algorytm B&B.



Rysunek 13. Przykładowe grafy DAG dla pojedynczej rundy algorytmu Rijndael

5.4.2. Platforma implementacyjna i środowisko projektowania

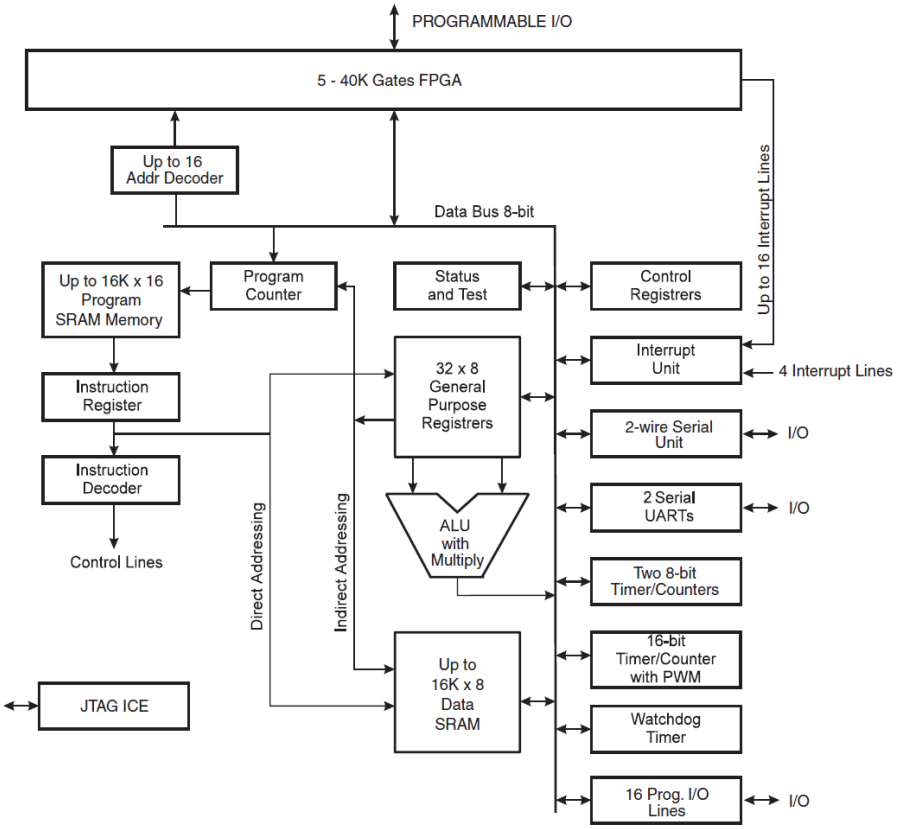
Realizację wykonano w heterogenicznym układzie SoC Atmel FPSLIC AT94K40L, zawierającym procesor AVR i logikę FPGA [14] na płycie prototypowej ATSTK94 [13]. FPSLIC jest stosunkowo małym i prostym układem SoC cechującym się niskim poborem mocy i małą powierzchnią, nadającym się w szczególności do zastosowań w kartach inteligentnych i urządzeniach przenośnych. Rysunek 14 przedstawia architekturę układu FPSLIC, który składa się z:

- 8-bitowego procesora RISC AVR o architekturze Harvard (rozdzielone magistrale danych i instrukcji) z rozszerzonym zbiorem poleceń, o wydajności 19MIPS przy 25MHz;

- układu FPGA AT40K (2304 komórek FPGA, 40 tysięcy bramek logicznych) charakteryzującego się efektywną strukturą połączeń;
- pamięci SRAM o pojemności 36kB, współdzielona pomiędzy pamięć programu (20-32 kB) i danych (4-16 kB) i czasie dostępu równym 15ns;
- układów UART, PWM oraz układów czasowo-licznikowych.

Komunikacja między mikroprocesorem i logiką programowalną może odbywać się poprzez pamięć współdzieloną, wewnętrzną 8-bitową magistralę danych lub przy wykorzystaniu 16 linii przerwań. Jednostką nadrzędną w komunikacji jest procesor, który wybiera jeden z 16 rejestrów IO w FPGA.

Układ FPSLIC umożliwia pracę w trybach o obniżonym poborze mocy. W analizowanej aplikacji użyteczny jest tryb Low-Power Idle, w którym wyłączany jest procesor, zachowana jest zawartość rejestrów i pamięci SRAM, działają układy UART i przerwania. Wejście w tryb obniżonego poboru mocy następuje programowo po wykonaniu instrukcji SLEEP, a powrót do normalnej pracy procesora następuje po przyjęciu przerwania, np. z układu UART lub z FPGA. Maksymalna częstotliwość zegara wynosi 25 MHz.



Rysunek 14. Architektura układu FPSLIC serii AT94K [14]

Użyto środowiska projektowego SystemDesigner dedykowanego dla układów FPSLIC, które integruje różne narzędzia innych producentów [248]:

- ModelSim - symulator HDL,
- LeonardoSpectrum v2001 - synteza logiczna,
- Figaro v7.6.7 - synteza fizyczna,
- IAR WorkBench, Image Craft Compiler v6.21 - kompilator C z profilerem kodu,
- AVRStudio v4.11 - symulator ISS dla AVR,
- Seamless CVE – symulator systemów heterogenicznych.

5.4.3. Strategia projektowania

Zgodnie z metodologią projektowania systemów SoC przedstawioną w rozdziale 1.1, pierwszym krokiem było utworzenie heterogenicznej specyfikacji układu w językach VHDL i C. Wyboru języków opisu dokonano na podstawie analizy języków specyfikacji systemu przeprowadzonej w pracy [192], z uwagi na łatwość specyfikacji algorytmu, przeniesienie istniejącego kodu referencyjnego oraz z uwagi na istnienie kompilatorów języka C dla wbudowanych procesorów i syntezerów języka dla wbudowanych FPGA w układach SoC. Wszystkie bloki funkcjonalne zostały zrealizowane na kilka sposobów tak, aby odzwierciedlić możliwość realizacji iteracyjnych lub równoległych pojedynczych bloków oraz współdzielenia zasobów. Dla każdego z 4 modułów algorytmu Rijndael dokonano 3 różnych realizacji sprzętowych i 3 programowych.

Poprawne funkcjonowanie zaprojektowanych bloków funkcjonalnych i specyfikacji całego systemu zweryfikowano za pomocą symulatora HDL dla modułów sprzętowych i symulatora ISS dla modułów programowych. Koszty układowe takie jak zajętość zasobów sprzętowych i czas wykonania otrzymano z syntezera logicznego Leonardo Spectrum. Koszty programowe takie jak zużycie pamięci i czas wykonania uzyskano przez kompilację kodu źródłowego i profilowanie kodu wynikowego. Ponieważ Atmel nie dostarcza narzędzi estymacji poboru mocy, dla każdej realizacji sprzętowej i programowej dokonano pomiarów zużycia mocy metodą różnicową. Tak wyznaczone parametry stanowią dane wejściowe do procesu podziału. Wykonano także symulację w środowisku heterogenicznej symulacji opisanym w rozdziale 5.1 w trybie ISS+HDL, w którym CPU było modelowane przez model ISS symulatora AVR, a logika FPGA była symulowana przez symulator HDL. Umożliwiło to zweryfikowanie zdefiniowanych interfejsów sprzętowo-programowych.

Ograniczenia określono na podstawie parametrów architektury układów FPSLIC: ilość pamięci i zasobów sprzętowych oraz na podstawie wymagań funkcjonalnych: moc i czas przetwarzania. Funkcja celu jest kombinacją liniową iloczynów współczynników i wartości parametrów, przy czym dominującą rolę odgrywa parametr mocy i czasu przetwarzania.

Utworzono grafy strukturalne opisujące warianty rozwiązań uwzględniając różne realizacje pojedynczych bloków oraz możliwość zrównoleglenia i współdzielenia zasobów. Brak odpowiednich narzędzi wymusił manualne wykonanie tego kroku. Grafy przedstawiają jedną rundę podstawową algorytmu Rijdnael wraz z generacją podkluczy rundy.

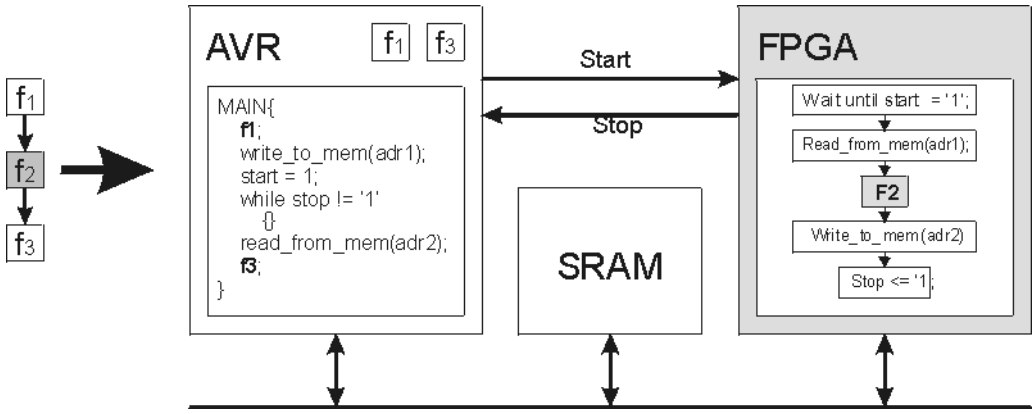
W pierwszym podejściu opisanym w pracach [72][248] do podziału wybrano algorytm MILP z planowaniem heurystycznym. Dane z estymacji kosztów zostały przeniesione do aplikacji HW_SW_Partitioner [233], która posłużyła do wygenerowania grafu podziału, czyli przydziału modułów do realizacji programowej i układowej. Funkcja celu optymalizacji została ustalona tak, aby algorytm szukał poprawnego rozwiązania minimalizując przy tym zużycie zasobów sprzętowych.

W drugim podejściu przedstawionym w pracy [249] do podziału zastosowano algorytm B&B. Dane z estymacji kosztów umieszczono w atrybutach grafu DAG podawanego na wejście aplikacji Partitioner opisanej w rozdziale 5.2. Z uwagi na fakt, iż algorytm B&B generuje rozwiązanie optymalne, interesującą miarą jest czas działania algorytmu oraz liczba analizowanych węzłów. Przebadane zostały warianty podziału przy różnych ograniczeniach zasobów i wydajności. Na każdym z grafów przeprowadzono kilkakrotnie podziały systemu, ustalając przy tym różną ilość dostępnych zasobów układowych oraz różne ograniczenia czasowe. Ilość

dostępnych zasobów sprzętowych dobierano tak, aby zasymulować możliwości systemów o małej ilości zasobów sprzętowych np. kart inteligentnych.

Z otrzymanych rozwiązań wybrano jedno i zgodnie z nim utworzono dwa pliki zawierające specyfikacje wszystkich elementów przeznaczonych do realizacji sprzętowej w języku VHDL i programowej w języku C oraz opis mechanizmów komunikacji. Wykonano współbieżną symulację za pomocą programu Seamless CVE. Po weryfikacji specyfikacji przeprowadzono syntezę logiczną i fizyczną modułów opisanych w języku VHDL za pomocą narzędzi Leonardo Spectrum oraz Figaro. Wykonano współbieżną symulację przy wykorzystaniu plików wynikowych wygenerowanych w procesie syntezy fizycznej w celu sprawdzenia poprawnego działania systemu. Realizację fizyczną zrealizowano na zestawie ewaluacyjnym taktowanym zegarem 18.432MHz. Komunikacja między częścią sprzętową i programową została zrealizowana za pomocą pamięci współdzielonej oraz linii adresowej (szczegóły przedstawia Rysunek 15).

Po procesie podziału dokonano realizacji heterogenicznej używając wyznaczonych implementacji poszczególnych modułów. Określono wartości wszystkich parametrów dla uzyskanej realizacji heterogenicznej i porównano z wartościami wyznaczonymi w procesie podziału. Zrealizowany algorytm Rijndael uruchomiono na płycie prototypowej ATSTK94 i przetestowano za pomocą aplikacji komunikującej się z płytą przez interfejs UART.



Rysunek 15. Schemat blokowy przykładowej realizacji kontroli i komunikacji [248]

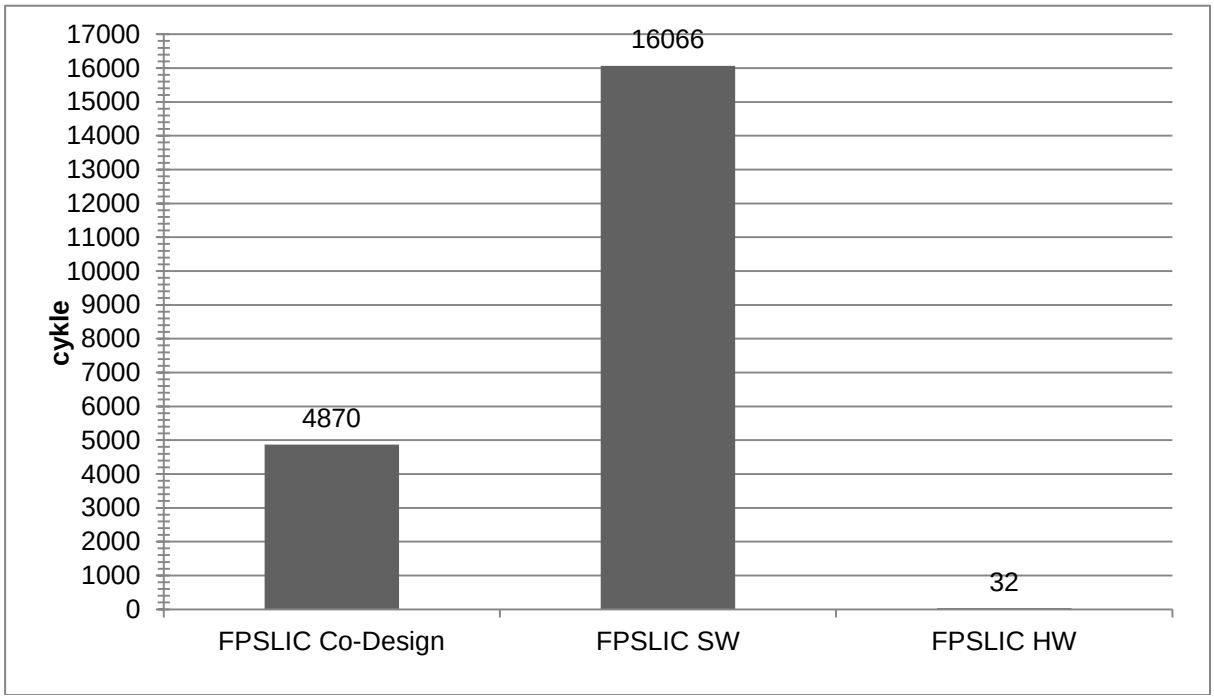
5.4.4. Wyniki implementacji

W pierwszej serii eksperymentów opisanych w pracach [72][248] dokonano podziału algorytmem MILP z heurystycznym planowaniem. Tabela 7 przedstawia wyniki podziału dla różnych ograniczeń i różnych wariantów realizacji modułów. Rozwiązania w kolumnach a) wykonują operację ShiftRow równolegle na wszystkich 128 bitach, natomiast w kolumnach b) operacja jest wykonywana iteracyjnie na blokach 32-bitowych. Oszacowanie nie uwzględnia kosztów sprzętowych modułów służących do komunikacji pomiędzy częścią sprzętową i programową, ani mechanizmów kontrolnych, których koszt może się istotnie różnić w poszczególnych realizacjach. Do realizacji sprzętowej wybrano wariant, który przedstawia Tabela 7 w zacieniowanej kolumnie b).

Porównując ze sobą wyniki wydajności realizacji sprzętowej, programowej i mieszanej, które przedstawia Rysunek 16, można zaobserwować możliwości różnorodnych realizacji, różniących się pod względem wydajności i zajętości zasobów. Kompromis pomiędzy parametrami jest osiągalny przez optymalizację jednego kosztem zwiększenia pozostałych.

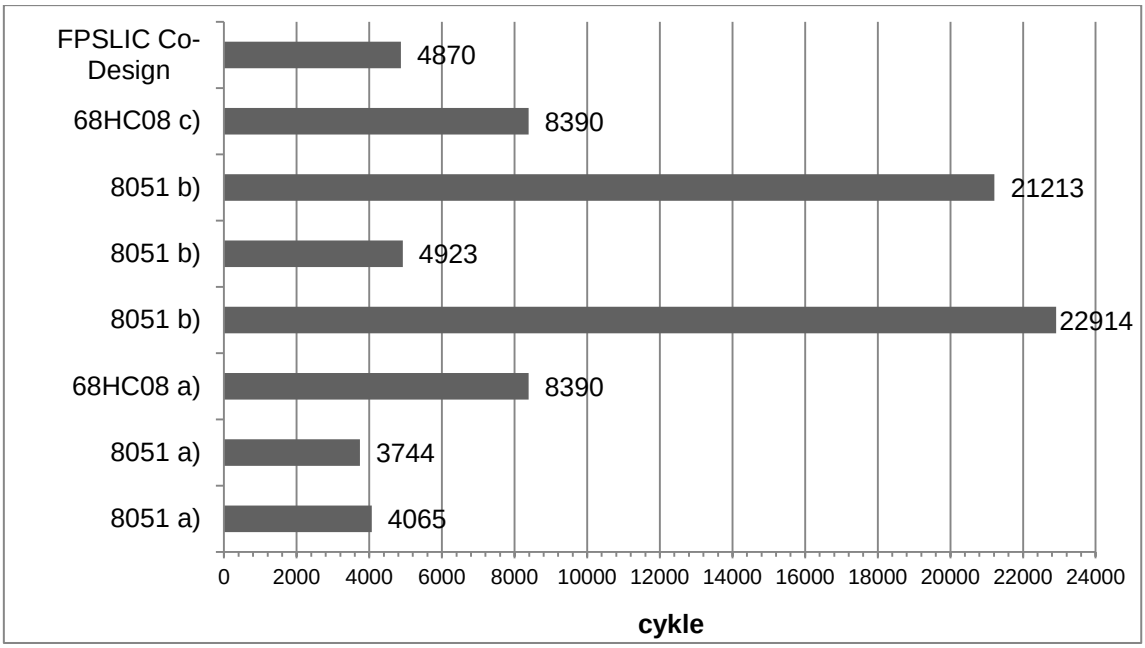
Tabela 7. Podział dwóch wybranych przykładów rozwiązania realizacji algorytmu Rijndael

	Ograniczenia							
	a)					**b)**		
Czas wykonania **MAX_TIME** [cykle]	6000	5000	4000	3000	2500	3000	2500	2000
Zasoby sprzętowe **MAX_HARDWARE_AREA** [instancje]	200	300	350	500	900	200	400	800
	Wyniki							
	a)					**b)**		
Czas wykonania [cykle]	6329	5257	4185	3113	2515	3063	2455	1393
Prędkość przetwarzania [Mbit/s] (zegar 18.432MHz)	0,37	0,45	0,56	0,76	0,94	0,77	0,96	1,69
Zasoby sprzętowe [instancje]	191	255	319	383	858	191	319	730
Pamięć programowa [byte]	1178	1068	958	848	592	848	738	482
Pamięć danych [byte]	522	522	522	522	256	266	266	256



Rysunek 16. Porównanie wydajności zaimplementowanych rozwiązań CoDesign, SW oraz HW

Rysunek 17 ukazuje większą wydajność rozwiązania mieszanego w stosunku do realizacji programowej wykonanej w niniejszej pracy jak i większości publikowanych w literaturze [167]. Różnica jest tym bardziej istotna, że nie dokonano optymalizacji kodu programowego w języku C, ani nie korzystano z wydajnego języka asemblera, jak miało to miejsce w wypadku innych rozwiązań.



Rysunek 17. Porównanie prędkości przetwarzania z innymi wybranymi realizacjami [72][167]

W drugiej serii eksperymentów opisanych w pracy [249] wyniki syntezy logicznej i fizycznej wykazały, że czas wykonania wszystkich realizacji sprzętowych modułów algorytmu Rijndael jest równy jednemu taktowi zegara. Wynika to faktu, że Rijdnael wykonuje operacje, które są stosunkowo łatwo implementowane w układach FPGA, jak opisano w rozdziale 8.7.2. Mała pojemność logiczna układu FPSLIC powoduje, że operacja ByteSub realizowana w postaci pamięci ROM 8x8 nie może zostać w pełni zrównoleglona. Architektura układu FPSLIC nie umożliwia korzystania z pełnej 128-bitowej ścieżki przetwarzania, zatem operację ByteSub zrealizowano w sposób sekwencyjny 8-bitowy.

Moduły programowe wykonują proste rozkazy logiczne lub działania na tablicach bajtowych. Wyjątkiem jest operacja MixColumn, która wykorzystuje mnożenie w ciele Galois GF(2^8). Realizacja za pomocą funkcji *xtime()* [64] jest podatna na atak czasowy, stąd potrzeba implementacji tablicowej, która znacznie wydłuża czas obliczeń.

Pomiar mocy zużywanej przez poszczególne operacje okazał się być stosunkowo skomplikowany. Płyta prototypowa ATSTK94 posiada tylko jedno napięcie zasilające dla wszystkich układów, co powoduje nakładanie się profili mocy. Aby ominąć ten problem, dokonywano implementacji tylko jednego modułu w danym momencie, wykonywano określoną liczbę operacji, a następnie wyznaczano różnicę pomiędzy uśrednionym w czasie poborem mocy płyty z danym modułem i bez danego modułu.

W algorytmie B&B zastosowano strategię gorliwą, a zadania wybierano zgodnie z ich kolejnością w grafie DAG. Kolejne węzły wybierano przy użyciu strategii DFS oraz BeFS. Strategia DFS nie wymaga rozwiązania początkowego i wymaga analizy około 2% wierzchołków w drzewie poszukiwań (32 spośród 1554). Strategia BeFS wymaga dobrego punktu startowego, w przeciwnym przypadku zachowuje się podobnie do strategii BFS. Czas obliczeń dla jednej rundy algorytmu AES, który jest problemem o rozmiarze N=4, jest mniejszy od 1ms.

Zbadano możliwości przyspieszenia działania algorytmu B&B za pomocą różnych kombinacji ograniczeń i współczynników funkcji celu. Szybsze działanie algorytmu można uzyskać stosując dobre rozwiązanie początkowe oraz nakładając ściślejsze ograniczenia na parametry rozwiązania, co powoduje częstsze odrzucanie rozwiązań

częściowych. Spowolnienie działania algorytmu jest widoczne w przypadkach, gdy w funkcji celu dominuje współczynnik czasu wykonania, gdyż funkcja LB szacuje dolne ograniczenie przez wartość parametru z rozwiązania częściowego, co rzadziej eliminuje węzły.

W wyniku procesu podziału operacje KeyAdd, ShiftRow oraz ByteSub zrealizowano programowo, a operację MixColumn – sprzętowo. Algorytm AES wykonuje się w 14080 cyklach zegara, co przy częstotliwości zegara 18.432 MHz daje 1309 szyfrowań na sekundę i przepustowość 167563 bps. Wyznaczone parametry implementacji heterogenicznej różnią się od obliczonych za pomocą algorytmu podziału:

- większy czas wykonania zadania, z uwagi na komunikację między modułami programowymi i sprzętowymi,
- mniejsza zajętość pamięci danych, gdyż każde zadanie zwalnia pamięć stosu po wykonaniu,
- mniejszy pobór mocy, z powodu niedokładnej metody pomiaru mocy dla poszczególnych modułów.

Zrealizowany algorytm AES uruchomiono na płycie prototypowej ATSTK94, a wyniki testów wykazały funkcjonalną poprawność działania. Z uwagi na dodatkową obsługę portu UART i konieczność kopiowania buforów osiągnięto przepustowość około 123500 bps, co jest stosunkowo dobrym wynikiem, znacznie przewyższającym realizacje homogeniczne.

5.4.5. Wnioski z implementacji

Przedstawiony proces projektowania i realizacji algorytmu Rijndael ukazał zalety wykorzystywania technik Codesign [248]:

- Parametry fizycznego rozwiązania mogą specyfikowane na etapie projektowania. Metoda zapewnia spełnienie narzuconych ograniczeń, unikając jednocześnie nadmiarowego szacowania kosztów. Jest to cecha szczególnie istotna, albowiem współcześnie realizowane projekty SoC są zazwyczaj błędnie szacowane.
- Możliwość wyboru realizacji ze zbioru poprawnych rozwiązań. Projektant sam decyduje o kompromisie między wydajnością a kosztem realizacji znając wartości obu tych parametrów dla wszystkich rozwiązań.
- Optymalizacja podziału bloków funkcjonalnych pozwala na uzyskanie dużych przepustowości na relatywnie prostym układzie.
- Skrócenie czasu projektowania przez zastosowanie języków wysokiego poziomu (C i VHDL).

Głównym problemem zastosowania technik Codesign jest brak zintegrowanego i automatycznego środowiska projektowego, które pozwoliłoby na ich wykorzystanie na skalę przemysłową. Dostępne rozwiązania pozwalają jedynie na wspólną symulację części układowych i programowych (Seamless CVE) [248].

6. Wnioski końcowe

W niniejszej pracy przedstawiono problemy projektowania i optymalizacji sprzętowo-programowych wbudowanych systemów przetwarzania danych. We współbieżnym projektowaniu sprzętu i oprogramowania najważniejszym problemem optymalizacji jest dokonanie podziału funkcjonalności na sprzęt i oprogramowanie. W celu skrócenia czasu dostarczenia na rynek stosuje się projektowanie z użyciem gotowych platform, co nakłada ścisłe ograniczenia na przestrzeń projektową, a także, z powodu narzutu komunikacji, indukuje podział gruboziarnisty i zmniejsza rozmiar grafu DAG. Zwiększając jednocześnie stopień ponownego użycia zaprojektowanych modułów, dostępnych jest wiele gotowych implementacji modułów, co narzuca konieczność wyboru ze zbioru implementacji, ale zarazem dostarcza dokładnych parametrów poszczególnych zadań, umożliwiając optymalizację metodom dokładnym. Dzięki dokładnym parametrom optymalizacja może być wykonywana tylko raz w cyklu projektowym, można zatem przeznaczyć większy (lecz przewidywalny) budżet czasu na optymalizację, ale także umożliwić przerwanie obliczeń po upływie określonego czasu, uzyskując rozwiązanie z gwarancją alfa-optymalności. Duży wolumen produkcyjny systemów wbudowanych powoduje, że z powodów ekonomicznych optymalizowanych jest jednocześnie wiele parametrów. Nawet minimalne ograniczenie kosztów skutkuje znacznym obniżeniem przy dużym wolumenie produkcyjnym, stąd też preferowane są metody optymalizacyjne dające rozwiązanie dokładne (globalnie optymalne).

Przedstawione powyżej tendencje projektowe doprowadziły do wyspecyfikowania wymagań na pożądaną metodę rozwiązania problemu podziału (omówione w rozdziale 1.1):

- przewidywalny czas obliczeń;
- wczesne uzyskanie rozwiązania dopuszczalnego;
- określenie alfa-optymalności;
- możliwość uzyskania rozwiązania optymalnego;
- rozwiązanie rozszerzonego problemu podziału;
- wielokryterialna funkcja celu;
- ograniczenia na wiele parametrów.

W kolejnych podrozdziałach rozdziału 3 dokonano analizy metod optymalizacyjnych stosowanych do problemu podziału wraz z przeglądem literatury. Tabela 3 w rozdziale 3.14 podsumowuje cechy poszczególnych metod optymalizacji i wskazuje, że metoda podziału i ograniczeń spełnia wszystkie wyspecyfikowane powyżej wymagania. Metoda charakteryzuje się wykładniczą złożonością obliczeniową, ale dla podziału gruboziarnistego ze stosunkowo niewielkim grafem DAG nie stanowi to istotnego ograniczenia. Istnieje możliwość estymacji czasu obliczeń w trakcie działania algorytmu oraz przyspieszania algorytmu na drodze zacieśniania ograniczeń i doboru parametrów. Metoda umożliwia wczesne uzyskanie rozwiązania dopuszczalnego i jako jedyna określa alfa-optymalność rozwiązania korzystając z funkcji ograniczenia dolnego LB. Metoda jest metodą enumeracyjną zapewniającą uzyskanie rozwiązania dokładnego (globalnie optymalnego). Rozszerzony problem podziału jest rozwiązywany przez podział politomiczny na każdym poziomie drzewa poszukiwań. Wielokryterialna funkcja celu jest uwzględniana przy ewaluacji rozwiązania częściowego i obliczaniu funkcji ograniczenia dolnego. Działanie algorytmu jest szybsze, jeśli funkcja celu nie zawiera metryk globalnych (np. czasu wykonania) i ma postać rekursywną (np. addytywną).

Ograniczenia na wiele parametrów są sprawdzane przy konstruowaniu rozwiązań częściowych a także przy odcinaniu gałęzi. Ścisłe ograniczenia powodują przyspieszenie działania metody.

W rozdziale 5.2 opisano implementację metody podziału i ograniczeń. Aplikację Partitioner zastosowano dla grafów generowanych w sposób losowy z wielokryterialną funkcją celu, ograniczeniami wielu parametrów i licznym zbiorem implementacji dla każdego zadania. W rozdziale 5.4 zaprezentowano zastosowanie aplikacji Partitioner do realizacji algorytmu szyfrowania w heterogenicznej platformie SoC.

Tym samym dowiedziono pierwszej tezy sformułowanej w rozdziale 1.2:

Metoda podziału i ograniczeń umożliwia wielokryterialną optymalizację rozwiązania rozszerzonego problemu podziału funkcjonalności na sprzęt i oprogramowanie przy zadanych ograniczeniach.

Metoda podziału i ograniczeń ma złożoność wykładniczą w najgorszym przypadku, jednak w najlepszym przypadku może mieć stosunkowo krótki czas obliczeń. Kluczowe jest dobranie parametrów metody tak, aby dla danego problemu wyeliminować jak najwięcej gałęzi na jak najwyższym poziomie. Nie jest możliwe analityczne określenie jak należy dobierać parametry metody dla konkretnej instancji zadania, uzasadnione jest zatem zastosowanie metaheurystyki. Zbadano wpływ parametrów metody na czas działania metody w rozwiązaniu losowych problemów wygenerowanych za pomocą DAG Generator. Na podstawie wyników przeprowadzonych eksperymentów opisanych w rozdziale 5.3.3 określono parametry podlegające strojeniu: strategia rozgałęzień, strategia wyboru podproblemu i strategia iteracji. Efekt zastosowania określonego zestawu parametrów można zaobserwować uruchamiając metodę podziału i ograniczeń na określony czas i estymując całkowity czas działania. Estymacja całkowitego czasu działania nie jest trywialna, gdyż w trakcie poszukiwań eliminowanych jest bardzo wiele gałęzi zawierających różną liczbę węzłów. Z uwagi na małą ilość potrzebnych uruchomień metody podziału i ograniczeń, parametry dobierano algorytmem symulowanego wyżarzania. W rozdziale 5.3.4 wykazano, że automatyczny dobór parametrów umożliwia uzyskanie najlepszego przypadku (w sensie złożoności obliczeniowej) i pozwala na osiągnięcie znaczącego przyspieszenia obliczeń.

Tym samym dowiedziono drugiej tezy sformułowanej w rozdziale 1.2:

Automatyczny dobór parametrów metody podziału i ograniczeń umożliwia przyspieszenie optymalizacji rozwiązania problemu podziału funkcjonalności na sprzęt i oprogramowanie.

W ten sposób wszystkie tezy sformułowane w rozdziale 1.2 zostały dowiedzione, a cel pracy został osiągnięty.

Oryginalnym dorobkiem autora są:

- taksonomia, analiza i porównanie metod optymalizacji stosowanych przy problemie podziału (rozdział 3);
- projekt i implementacja środowiska heterogenicznej symulacji sprzętowo-programowej (rozdział 5.1), rozwijanego obecnie przez firmę Alatek;
- projekt i implementacja aplikacji Partitioner implementującej metodę B&B do rozwiązania problemu podziału (rozdział 5.2);
- zastosowanie autokalibracji parametrów aplikacji Partitioner przyspieszającej działanie metody B&B (rozdział 5.3);

- implementacje algorytmu Rijndael z użyciem technik projektowania mieszanego sprzętowo-programowego zrealizowane w systemach on-chip Atmel FPSLIC (rozdział 5.4);
- implementacje algorytmów kryptograficznych w technologii Xilinx Virtex II (rozdział 8.8), przewyższające parametrami inne dostępne na rynku w tej technologii,
- metoda wyrównywania mocy polegająca na duplikacji logiki po syntezie logicznej operująca bezpośrednio na formacie EDIF i zastosowanie symulatora mocy Xilinx XPower do analizy różnicowej mocy przed wykonaniem fizycznego prototypu (rozdział 8.10).

7. Bibliografia

[1] Jay K. Adams, Donald E. Thomas, "Multiple-Process Behavioral Synthesis for Mixed Hardware-Software Systems", Proceedings of the 8th International Symposium on System Synthesis, ISSS 1995, pp. 10-15, Cannes, France, September 13-15, 1995. ISBN 0-89791-771-5.

[2] Mehdi-Laurent Akkar, Christophe Giraud, "An implementation of DES and AES, secure against some attacks", Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001, LNCS 2162, pp. 309-318, Paris, France, May 14-16, 2001. ISBN 3-540-42521-7.

[3] Ghassan Al-Hayek, Yves Le Trayon, Chantal Robach, "Impact of System Partitioning on Test Cost", Journal of IEEE Design & Test of Computers, vol. 14, no. 1, pp. 64-74, IEEE Computer Society Press, Los Alamitos, California, January 1997. ISSN 0740-7475.

[4] Altera Corporation, "FLEX 10K Embedded Programmable Logic Device Family Data Sheet", Data Sheet, ver. 4.2, San Jose, California, January 2003.

[5] Nancy M. Amato, Ping An, "Task Scheduling and Parallel Mesh-Sweeps in Transport Computations", Technical Report 00-009, Department of Computer Science, Texas A&M University, College Station, Texas, January 15, 2000.

[6] Ross Anderson, Eli Biham, Lars Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard", Proceedings of the 1st AES Candidate Conference, Ventura, California, August 20-22, 1998.

[7] Ross Anderson, Eli Biham, Lars Knudsen, "Serpent and Smartcards", Proceedings of the 3rd International Conference on Smart Card Research and Applications, CARDIS 1998, LNCS 1820, pp. 246-253, Louvain-la-Neuve, Belgium, September 14-16, 1998. ISBN 3-540-67923-5.

[8] Kazumaro Aoki, Helger Lipmaa, "Fast Implementations of AES Candidates", Proceedings of the 3rd AES Candidate Conference, pp. 106-120, New York, New York, April 13-14, 2000.

[9] Apple Inc., Khronos Group Inc., "OpenCL 1.1 Specification", Technical Specification, rev. 36, Cupertino, California, September 30, 2010.

[10] Peter Arato, Sandor Juhasz, Zoltan A. Mann, Andrasz Orban, David Papp, „Hardware-software partitioning in embedded system design", Proceedings of the IEEE International Symposium on Intelligent Signal Processing, WISP 2003, pp. 197-202, Budapest, Hungary, September 4-6, 2003. ISBN 0-7803-7864-4.

[11] ARM Inc., „ARM RDI 1.5.1", Technical Documentation, ARM RDI-0057-CUST-ESPC-A, Cambridge, Great Britain, January 24, 2001.

[12] ARM Inc., "AMBA Specification", online publication, 2010.

[13] Atmel Corporation, "Starter Kit. FPSLIC™ Programmable SLI ATSTK94", Data Sheet, San Jose, California, January 2002.

[14] Atmel Corporation, "AT94K Series Field Programmable System Level Integrated Circuit", Data Sheet, San Jose, California, January 2008.

[15] Michel Auguin, Karim Ben Chehida, Jean-Philippe Diguet, Xavier Fornari, Anne-Marie Fouiliart, Christian Gamrat, Guy Gogniat, Phillipe Kajfasz, Yannick Le Moullec, „Partitioning and Codesign tools & methodology for Reconfigurable Computing: The EPICURE philosophy", Proceedings of the 3rd International Workshop on Embedded Computer Systems: Architecture, Modeling and Simulation, SAMOS 2003, LNCS 3133, Samos, Greece, July 21-23, 2003. ISBN 3-540-22377-0.

[16] Jakob Axelsson, "Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems", PhD Dissertation, Linkoping University, Linkoping, Sweden, October, 1995. ISBN 91-7871-616-0.

[17] Jakob Axelsson, „Three Search Strategies for Architecture Synthesis and Partitioning of Real-Time Systems", IDA Technical Report, LiTH-IDA-R-96-32, Linkoping University, Linkoping, Sweden, 1996. ISSN 0281-4250.

[18] Jakob Axelsson, "A Hardware/Software Codesign Approach to System-Level Design of Real-Time Applications", Proceedings of the Swedish National Conference on Real-Time Systems, SNART 1997, pp. 21-22, Lund, Sweden, August 21-22, 1997.
[19] Jakob Axelsson, "A Case Study in Heterogeneous Implementation of Automotive Real-Time Systems", Proceedings of the 6th International Workshop on Hardware/Software Codesign, CODES 1998, Seattle, Washington, March 15-18, 1998. ISBN 0-8186-8442-9.
[20] Abdenour Azzedine, Jean-Philippe Diguet, Jean-Luc Pilippe, "Large Exploration for HW/SW partitioning of Multirate and Aperiodic Real-Time Systems", Proceedings of the 10th International Symposium on Hardware/Software Codesign, CODES 2002, pp. 85-90, Estes Park, Colorado, May 6-8, 2002. ISBN 1-58113-542-4.
[21] David A. Bader, William E. Hart, Cynthia A. Phillips, "Parallel Algorithm Design for Branch and Bound", in "Tutorials on Emerging Methodologies and Applications in Operations Research", Kluwer Academic Press, Chapter 5, pp. 1-44, 2005. ISBN 978-0-387-22826-6.
[22] Alessandro Balboni, William Forniciari, Donatella Sciuto "Partitioning and Exploration Strategies in the TOSCA Co-Design Flow", Proceedings of the 4th International Workshop on Hardware/Software Codesign, CODES 1996, pp. 62-69, Pittsburgh, Pennsylvania, March 18-20, 1996. ISBN 0-8186-7243-9.
[23] Alessandro Balboni, William Fornaciari, Donatella Sciuto "Co-Synthesis and Co-Simulation of Control-Dominated Embedded Systems", International Journal of Design Automation for Embedded Systems, vol. 1, no. 3, pp. 257-289, Springer, June 1996. ISSN 0929-5585.
[24] Massimo Baleani, Frank Gennari, Yunjian Jiang, Yatish Patel, Robert K. Brayton, Alberto Sangiovanni-Vincentelli, "HW/SW Partitioning and Code Generation of Embedded Control Applications on a Reconfigurable Architecture Platform", Proceedings of the 10th International Symposium on Hardware/Software Codesign, CODES 2002, pp. 151-156, Estes Park, Colorado, May 6-8, 2002. ISBN 1-58113-542-4.
[25] Sudarshan Banerjee, Nikil Dutt, „Very Fast Simulated Annealing for HW-SW Partitioning", CECS Technical Report #04-#18, Center for Embedded Computer Systems, University of California, Irvine, California, June 2004.
[26] Sudarshan Banerjee, Nikil Dutt, „Efficient Search Space Exploration for HW-SW Partitioning", Proceedings of the 2nd International Conference on Hardware/Software Codesign and System Synthesis, CODES + ISSS 2004, pp. 122-127, Stockholm, Sweden, September 8-10, 2004. ISBN 1-58113-937-3.
[27] Hagai Bar-El, "Introduction to side channel attacks", White Paper, Discretix Technologies Ltd., Netanya, Israel, 1999.
[28] Edna Barros, Wolfgang Rosenstiel, "A Method for Hardware/Software Partitioning", Proceedings of the 6th Annual European Conference on Computer Systems and Software Engineering, CompEuro 1992, pp. 580-585, The Hague, The Netherlands, May 4-8, 1992.
[29] Sandro Bartolini, Roberto Giorgi, Enrico Martinelli, "Instruction Set Extensions for Cryptographic Applications", in "Cryptographic Engineering", Springer, Chapter 9, pp. 191-233, 2009. ISBN 978-0-387-71816-3.
[30] Lawrence E. Bassham III, "Efficiency Testing of ANSI C Implementations of Round 2 Candidate Algorithm for the Advanced Encryption Standard", Proceedings of the 3rd AES Candidate Conference, pp. 136-148, New York, New York, April 13-14, 2000.
[31] Lejla Batina, Alireza Hojdat, David Hwang, Kazuo Sakiyama, Ingrid Verbauwhede, "Reconfigurable Architectures for Curve-Based Cryptography on Embedded Micro-Controllers", Proceedings of the 16th International Conference on Field Programmable Logic and Applications, FPL 2006, pp. 667-670, Madrid, Spain, August 28-30, 2006.
[32] Roman Batoukov, Tor Sørevik, "A Generic Parallel Branch and Bound Environment on a Network of Workstations", Proceedings of the Conference on High Performance Computing on Hewlett-Packard Systems, HiPer 1999, pp. 474-483, Tromse, Norway, June 27-30, 1999.

[33] Luca Benini, Alessandro Bogliolo, Giovanni De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management", Journal of IEEE Transactions on Very Large Scale Integration (VLSI) Systems – Special section on low-power electronics and design, vol. 8, no. 3, pp. 299-316, June 2000. ISSN 1063-8210.

[34] Luca Benini, Alberto Macii, Enrico Macii, Elvira Omerbegovic, Massimo Poncino, Fabrizio Pro, „Energy-aware design techniques for differential power analysis protection", Proceedings of the 40th Annual Design Automation Conference, DAC 2003, pp. 36-41, Anaheim, California, June 2-6, 2003. ISBN: 1-58113-688-9.

[35] Guido Bertoni, Luca Breveglieri, Matteo Venturi, "ECC Hardware Coprocessor for 8-bit System and Power Consumption Considerations", Proceedings of the 3rd International Conference on Information Technology: New Generations, ITNG 2006, pp. 573-574, Las Vegas, Nevada, April 10-12, 2006. ISBN: 0-7695-2497-4.

[36] Jean-Luc Beuchat, "FPGA Implementations of the RC6 Block Cipher", Proceedings of the 13th International Conference on Field Programmable Logic and Applications, FPL 2003, LNCS 2778, pp. 101-110, Lisbon, Portugal, September 1-3, 2003. ISBN 3-540-40822-3.

[37] Karthikeyan Bhasyam, Kia Bazargan, "HW/SW Codesign Incorporating Edge Delays Using Dynamic Programming", Proceedings of the Euromicro Symposium on Digital System Design, DSD 2003, pp. 264-271, Belek-Antalya, Turkey, September 1-6, 2003. ISBN 0-7695-2003-0.

[38] Nguyen Ngoc Binh, Masaharu Imai, Akichika Shiomi, Noboyuki Hikichi, "A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate counts", Proceedings of the 33rd Annual Design Automation Conference, DAC 1996, pp. 527-532, Las Vegas, Nevada, June 03-07, 1996. ISBN 0-89791-779-0.

[39] Nguyen Ngoc Binh, Masaharu Imai, Akichika Shiomi, "A New HW/SW Partitioning Algorithm for Synthesizing the Highest Performance Pipelined ASIPs with Multiple Identical FUs", Proceedings of the European Design Automation Conference, EURO-DAC 1996, pp. 126-133, Geneva, Switzerland, September 16-20, 1996. ISBN 0-8186-7573-X.

[40] Jacek Błażewicz, Wojciech Cellary, Roman Słowiński, Jan Węglarz, „Badania Operacyjne dla Informatyków", Wydawnictwa Naukowo-Techniczne, Warszawa, Polska, 1983. ISBN 83-204-0519-X.

[41] Piotr Bora, Tomasz Czajka, "Implementation of the Serpent Algorithm Using Altera FPGA Devices", Public Comment on AES Candidate Algorithm – Round 2, May 13, 2000.

[42] Stephen Boyd, Arpita Ghosh, Alessandro Magnani, "Branch and Bound Methods", Lecture Notes for EE392o, Stanford University, Palo Alto, California, November 1, 2003.

[43] Stephen Boyd, "Branch and Bound Methods", Lecture Notes for EE364b, Stanford University, Palo Alto, California, March 11, 2007.

[44] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, M. Scott Marshall, "GraphML Progress Report. Structural Layer Proposal", Proceedings of the 9th International Symposium on Graph Drawing, GD 2001, LNCS 2265, pp. 501-512, Vienna, Austria, September 23-26, 2001. ISBN 3-540-43309-0.

[45] Arie de Bruin, Alexander H.G. Rinnooy Kan, Harry W.J.M. Trienekens, "A simulation tool for the performance evaluation of parallel branch and bound algorithms", Journal of Mathematical Programming, vol. 42, no. 2, pp. 245-271, Springer-Verlag, Secaucus, New Jersey, September 1988. ISSN 0025-5610.

[46] Jerome Burke, John McDonald, Todd Austin, "Architectural Support for Fast Symmetric-Key Cryptography", Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2000, pp. 178-189, Cambridge, Massachusetts, November 13-15, 2000. ISBN 1-58113-317-0.

[47] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen Matyas, Luke O'Connor, Mohammad Peyravian, David Safford, Nevenko Zunic, "MARS – a candidate cipher for AES", Proceedings of the 1st AES Candidate Conference, Ventura, California, August 20-22, 1998.

[48] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen Matyas, Luke O'Connor, Mohammad Peyravian, David Safford, Nevenko Zunic, "MARS and the AES Selection Criteria", Public Comment on AES Candidate Algorithm – Round 2, May 15, 2000.
[49] Vincenzo Catania, Michele Malgeri, Marco Russo, "Applying Fuzzy Logic to Codesign Partitioning", IEEE Micro, vol. 17, no. 3, pp. 62-70, IEEE Computer Society Press, Los Alamitos, California, May-June 1997. ISSN 0272-1732.
[50] Biman Chakraborty, Ting Chen, Tulika Mitra, Abhik Roychoudhury, "Handling Constraints in Multi-Objective GA for Embedded System Design", Proceedings of the 19th International Conference on VLSI Design, VLSID 2006, pp. 305-310, Hyderabad, India, January 3-7, 2006. ISBN 0-7695-2502-4.
[51] Jui-Ming Chang, Massoud Pedram, "Codex-dp: Codesign of Communicating System Using Dynamic Programming", Proceedings of the Conference on Design, Automation and Test in Europe, DATE 1999, pp. 568-573, Munich, Germany, March 9-12, 1999. ISBN 0-7695-0078-1.
[52] Karam S. Chatha, Ranga Vemuri, "A Tool for Partitioning and Pipelined Scheduling of Hardware-Software Systems", Proceedings of the 11th International Symposium on System Synthesis, ISSS 1998, pp. 145-151, Taiwan, China, December 2-4, 1998. ISBN 0-8186-8623-5.
[53] Karam S. Chatha, Ranga Vemuri, "An Iterative Algorithm for Partitioning and Scheduling of Area Constrained HW-SW Systems", Proceedings of the 10th IEEE International Workshop on Rapid System Prototyping, RSP 1999, pp. 134-139, Clearwater, Florida, June 16-18, 1999. ISBN 0-7695-0246-6.
[54] Karam S. Chatha, Ranga Vemuri, "An Iterative Algorithm for Hardware-Software Partitioning, Hardware Design Space Exploration and Scheduling", Design Automation for Embedded Systems, vol. 5, no. 3-4, pp. 281-293, Kluwer Academic Publishers, August 2000. ISSN 0929-5585.
[55] Olivier Chapelle, Vikas Sindhwani, S. Sathiya Keerthi, "Branch and Bound for Semi-Supervised Support Vector Machines", Advances in Neural Information Processing Systems, Proceedings of the 20th Annual Conference on Neural Information Processing Systems, NIPS 2006, pp. 217-224, Vancouver, Canada, December 4-7, 2006.
[56] Karim Ben Chehida, Michel Auguin, "HW / SW Partitioning Approach For Reconfigurable System Design", Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES 2002, pp. 247-251, Grenoble, France, October 8-11, 2002.
[57] John W. Chinneck, "Practical Optimization: a Gentle Introduction", Lecture Notes, Systems and Computer Engineering, Carleton University, Ottawa, Canada, December 12, 2010.
[58] Massimiliano Chiodo, Paolo Giusto, Attila Jurecska, Harry C. Hsieh, Alberto Sangiovanni-Vincencelli, Luciano Lavagno, "Hardware-Software Codesign of Embedded Systems", IEEE Micro, vol. 14, no. 4, pp. 26-36, August 1994.
[59] Pawel Chodowiec, Kris Gaj, "Implementation of the Twofish Cipher Using FPGA Devices", Technical Report, Electrical and Computer Engineering, George Mason University, Fairfax, Virginia, July 1999.
[60] Pawel Chodowiec, Po Khuon, Kris Gaj, "Fast implementations of secret-key block ciphers using mixed inner- and outer-round pipelining", Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2001, pp. 94-102, Monterrey, California, February 11-13, 2001.
[61] Craig S.K. Clapp, "Instruction-level Parallelism in AES Candidates", Proceedings of the 2nd AES Candidate Conference, Rome, Italy, March 22-23, 1999.
[62] Jens Clausen, „Branch and Bound Algorithms – Principles and Examples", Department of Computer Science, University of Copenhagen, Copenhagen, Denmark, March 12, 1999.
[63] Don Coppersmith, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Mohammad Peyravian, David Safford, Nevenko Zunic, "IBM Comments",

Proceedings of the 3rd AES Candidate Conference, pp. 333-336, New York City, New York, April 13-14, 2000.
[64] Joan Daemen, Vincent Rijmen, "The Rijndael Block Cipher. AES Proposal: Rijndael", Proceedings of the 1st AES Candidate Conference, Ventura, California, August 20-22, 1998.
[65] Joan Daemen, Vincent Rijmen, "Rijndael for AES", Proceedings of the 3rd AES Candidate Conference, pp. 343-348, New York City, New York, April 13-14, 2000.
[66] Joan Daemen, Vincent Rijmen, "The AES Second Round Comments of the Rijndael", NIST Submission, May 12, 2000.
[67] Joan Daemen, Vincent Rijmen, Paulo S.L.M. Baretto, "Rijndael: beyond the AES", Mikulasska Kryptobesidka, 3rd Czech and Slovak Cryptology Workshop, pp. 1-10, Prague, Czech Republic, December 2-3, 2002.
[68] Andreas Dandalis, Viktor K. Prasanna, Jose D.P. Rolim, "An Adaptive Cryptographic Engine for IPSec Architectures", Proceedings of the 8th IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2000, pp. 132-144, Napa Valley, California, April 17-19, 2000.
[69] Andreas Dandalis, Viktor K. Prasanna, Jose D.P. Rolim, "A Comparative Study of Performance of AES Final Candidates Using FPGAs", Proceedings of the 2nd International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, pp. 125-140, Worcester, Massachusetts, August 17-18, 2000.
[70] Gary Dare, Ross Nelson, "Seamless Hardware/Software Co-Verification of FPGAs", White Paper, Mentor Graphics, Wilsonville, Oregon, 2003.
[71] Mark De Clercq, Vincent Levesque, "A VHDL Implementation of the Twofish Block Cipher", Final Project, Computer Architecture Laboratory 304-487 A, McGill Electrical & Computer Engineering, Montreal, Canada, December 1, 1999.
[72] Lech Dembek, „Zastosowanie technik hardware-software codesign w implementacji algorytmu kryptograficznego AES w układzie FPSLIC", Praca Dyplomowa, Wydział Elektroniki Telekomunikacji i Informatyki Politechniki Gdańskiej, Gdańsk, Polska, 20 październik 2003.
[73] Giovanni De Micheli, Rajesh Gupta, „Hardware/Software Co-design", Proceedings of the IEEE Micro vol. 85, no. 3, pp. 349-365, March 1997.
[74] Giovanni De Micheli, "Synteza i Optymalizacja Układów Cyfrowych", Wydawnictwa Naukowo-Techniczne, Warszawa, Polska, 1998. ISBN 83-204-2219-1.
[75] Giovanni De Micheli, Rolf Ernst, Wayne Wolf, "Readings in Hardware/Software Co-Design", Morgan Kaufmann Publishers, San Francisco, California, 2002. ISBN 1-55869-702-1.
[76] Alain Denise, Marcio Vasconcellos, Dominic J. A. Welsh, "The Random Planar Graph", Congressus Numerantium, vol. 113, pp. 61-79, University of Manitoba, Winnipeg, Canada, 1996.
[77] Robert P. Dick, David L. Rhodes, Wayne Wolf, "TGFF: Task Graphs for Free", Proceedings of the 6th International Workshop on Hardware/Software Codesign, CODES / CASHE 1998, pp. 97-101, Seattle, Washington, March 15-18, 1998. ISBN 0-8186-8442-9.
[78] Lidia Dutkiewicz, "Algorytmy Decyzyjne: Metoda Podziału i Ograniczeń", Materiały do wykładu, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Akademia Górniczo-Hutnicza, Kraków, Polska, 21 marca 2007.
[79] R.J. Easter, E.W. Chencinski, Edward J. D'Avignon, S.R. Greenspan, W.A. Merz, C.D. Norberg, „S/390 Parallel Enterprise Server CMOS Cryptographic Coprocessor", IBM Journal on Research and Development, vol. 43, no. 5, pp. 761-776, Riverton, New York, September 1999. ISSN 0018-8646.
[80] Hans Eberle, Arvinderpal Wander, Nils Gura, Sheueling Chang-Shantz, Vipul Gupta, „Architectural Extensions for Elliptic Curve Cryptography over GF(2m) on 8-bit Microprocessors", Proceedings of the 16th IEEE International Conference on Application-Specific Systems, Architecture and Processors, ASAP 2005, pp. 343-349, Samos, Greece, July 23-25, 2005.

[81] Adam J. Elbirt, Christof Paar, "An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher", Proceedings of the ACM/SIGDA 8th International Symposium on Field Programmable Gate Arrays, FPGA 2000, pp. 33-40, Monterey, California, February 10-11, 2000.
[82] Adam J. Elbirt, Wei Yip, Brendon Chetwynd, Christof Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", Proceedings of the 3rd AES Candidate Conference, pp. 13-27, New York City, New York, April 13-14, 2000.
[83] Adam J. Elbirt, Wei Yip, Brendon Chetwynd, Christof Paar, "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 9, no. 4, pp. 545-557, August 2001.
[84] Petru Eles, Zebo Peng, Krzysztof Kuchcinski, Alexa Doboli, "Hardware/ Software Partitioning of VHDL System Specifications", Proceedings of the European Design Automation Conference, EURODAC 1996, pp. 434-439, Geneva, Switzerland, September 16-20, 1996.
[85] Petru Eles, Zebo Peng, Krzysztof Kuchcinski, Alexa Doboli, "Hardware/Software Partitioning with Iterative Improvement Heuristics", Proceedings of the 9th International Symposium on System Synthesis, ISSS 1996, pp. 71-76, La Jolla, California, November 6-8, 1996.
[86] Petru Eles, Zebo Peng, Krzysztof Kuchcinski, Alexa Doboli, "System Level Hardware/Software Partitioning based on Simulated Annealing and Tabu Search", Design Automation for Embedded Systems, vol. 2, no. 1, pp. 5-33, January 1997.
[87] Christian C. Enz, Amre El-Hoiydi, Jean-Dominique Decotignie, Vincent Peiris, "WiseNET: An Ultralow-Power Wireless Sensor Network Solution", IEEE Computer, vol. 37, no. 8, pp. 62–70, August 2004.
[88] Rolf Ernst, Jorg Henkel, Thomas Benner, "Hardware-Software Co-Synthesis for Microcontrollers", Journal of IEEE Design & Test of Computers, vol. 10, no. 4, pp. 64-75, Los Alamitos, California, October 1993.
[89] Rolf Ernst, „Codesign of Embedded Systems: Status and Trends", Journal of IEEE Design & Test of Computers, vol. 15, no. 2, pp. 45-54, Los Alamitos, California, April 1998.
[90] Markus Ernst, Michael Jung, Felix Madlener, Sorin A. Huss, Rainer Bluemel, „A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2n)", Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002, LNCS 2523, pp. 381-399, Redwood City, California, August 13-15, 2002.
[91] Antonio J. Esteves, Alberto J. Proenca, „A Partition Methodology to Develop Data Flow Dominated Embedded Systems", Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES 2004, Hamilton, Canada, June 15-18, 2004.
[92] Antonio J. Esteves, Alberto J. Proenca, „A Hardware/Software Partition Methodology Targeted to an FPGA/CPLD Architecture", Proceedings of the Conference on Reconfigurable Systems (Jornadas sobre Sistemas Reconfiguráveis), REC 2005, Faro, Portugal, February 21, 2005.
[93] Paul N. Fahn, Peter K. Pearson, "IPA: A New Class of Power Attacks", Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems, CHES 1999, LNCS 1717, pp. 173-186, Worcester, Massachusetts, August 12-13, 1999.
[94] Charles M. Fiduccia, Robert M. Mattheyses, „A Linear-time Heuristic for Improving Network Partitions", Proceedings of the 19th Design Automation Conference, DAC 1982, pp. 175-181, Las Vegas, Nevada, June 14-16, 1982. ISBN 0-89791-020-6.
[95] Viktor Fischer, „Realization of the Round 2 AES Candidates using Altera FPGA", Proceedings of the 3rd AES Candidate Conference, pp. 13-14, New York City, New York, April 13-14, 2000.

[96] Krzysztof Gaj, Pawel Chodowiec, „Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware", Proceedings of the 3rd AES Candidate Conference, pp. 40-56, New York City, New York, April 13-14, 2000.
[97] Krzysztof Gaj, Pawel Chodowiec, „Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays", Proceedings of the RSA Security Conference – Cryptographer's Track, CT RSA 2001, pp. 84-99, San Francisco, California, April 8-12, 2001.
[98] Carlo Galuzzi, Dimitris Theodoropoulos, Roel Meeuws, Koen Bertels, „Automatic Instruction-Set Extensions with the Linear Complexity Spiral Search", Proceedings of the International Conference on Reconfigurable Computing and FPGAs, RECONFIG 2008, pp. 31-36, Cancun, Mexico, December 3-5, 2008.
[99] Emden Ganser, Eleftherios Koutsofios, Stephen North, "Drawing graphs with dot", User's Manual, January 26, 2006.
[100] Jerzy Gawinecki, Piotr Bora, "Analiza bezpieczeństwa implementacji sprzętowych algorytmów szyfrowania informacji", Biuletyn Wojskowej Akademii Technicznej, vol. 57, no. 4, pp. 101-116, Warszawa, Polska, 2008.
[101] Brian Gladman, "Implementation Experience with AES Candidate Algorithms", Proceedings of the 2nd AES Candidate Conference, Rome, Italy, March 22-23, 1999.
[102] Fred Glover, Lee Tangedahl, "Dynamic Strategies for Branch and Bound", OMEGA, The International Journal of Management Science, vol. 4, No. 5, pp. 571-575, June 1976.
[103] Fred Glover, "Tabu Search – Part I", ORSA Journal on Computing, vol. 1, no. 3, pp. 190-206, Summer 1989.
[104] Fred Glover, "Tabu Search – Part II", ORSA Journal on Computing, vol. 2, no. 1, pp. 4-32, Winter 1990.
[105] Guy Gogniat, Wayne Burleson, Mike O'Malley, Lilian Bossuet, "IPSec Implementation Project using FPGA and Microcontroller", Proceedings of the IEEE International Workshop on Reconfigurable Computing Education, Karlsruhe, Germany, March 1, 2006.
[106] Marcin Gomułkiewicz, Mirosław Kutyłowski, "Hamming Weight Attacks on Cryptographic Hardware – Breaking Masking Defense", Proceedings of the 7th European Symposium on Research in Computer Security, ESORICS 2002, LNCS 2502, pp. 90-103, Zurich, Switzerland, October 14-16, 2002.
[107] Jie Gong, Daniel D. Gajski, Sanjiv Narayan, "Software estimation from executable specifications", Journal of Computer and Software Engineering – Special issue: Hardware-Software Codesign, vol. 2, no. 3, pp. 239-258, Norwood, New York, March 1994.
[108] James Ross Goodman, "Energy Scalable Reconfigurable Cryptographic Hardware for Portable Applications", PhD Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 11, 2000.
[109] Jesper Grode, Peter Voigt Knudsen, Jan Madsen, "Hardware resource allocation for hardware/software partitioning in LYCOS system", Proceedings of the Conference on Design Automation and Test in Europe, DATE 1998, pp. 22-27, Paris, France, February 23-26, 1998.
[110] Xu Guo, Zhimin Chen, Patrick Schaumont, "Energy and Performance Evaluation of an FPGA-Based SoC Platform with AES and PRESENT Coprocessors", Proceedings of the 8th International Workshop on Embedded Computer Systems: Architecture, Modeling and Simulation, SAMOS 2008, LNCS 5114, pp.106-115, Samos, Greece, July 21-24, 2008.
[111] Rajesh Kumar Gupta, Giovanni De Micheli, "Hardware-Software Cosynthesis for Digital Systems", IEEE Design & Test of Computers, vol. 10, no. 3, pp. 29-41, July 1993.
[112] Rajesh Kumar Gupta, "Co-synthesis of hardware and software for digital embedded systems", PhD Dissertation, Stanford University, Stanford, California, December 10, 1993.

[113] Gael Hachez, Francois Koeune, Jean-Jacques Quisquater, "cAESar results: Implementation of Four AES Candidates on Two Smart Cards", Proceedings of the 2nd AES Candidate Conference, pp. 95-108, Rome, Italy, March 22-23, 1999.
[114] David Harris, Deverl Stokes, "Executing an RTOS on Simulated Hardware using Co-verification", Proceedings of the 2000 Embedded System Conference, San Jose, California, September 26-28, 2000.
[115] Jörg Henkel, Rolf Ernst, "An Approach to Automated Hardware/Software Partitioning Using a Flexible Granularity that is driven by High Level Estimation Techniques", Journal IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 9, no. 2, pp. 273-289, April 2001.
[116] Alain Hertz, Eric Taillard, Dominique de Werra, "A Tutorial on Tabu Search", Proceedings of the Giornate di Lavoro, Entreprise Systems: Management of Technological and Organizational Changes, AIRO 1995, pp. 13-24, Ancona, Italia, September 20-22, 1995.
[117] Michael Himsolt, "GML: A portable Graph File Format", Technical Report, Universität Passau, Passau, Germany, 1997.
[118] Alireza Hodjat, Lejla Batina, David Hwang, Ingrid Verbauwhede, "HW/SW co-design of a hyperelliptic curve cryptosystem using a microcode instruction set coprocessor", the VLSI Journal of Integration, Special issue: Embedded cryptographic hardware, vol. 40, no. 1, pp. 45-51, Amsterdam, The Netherlands, January 2007.
[119] Thomas Hollstein, Jürgen Becker, Andreas Kirschbaum, Manfred Glesner, "HiPART: A New Hierarchical Semi-Interactive HW/SW Partitioning Approach with Fast Debugging for Real-Time Embedded Systems", Proceedings of the 6th International Workshop on Hardware/Software Codesign, CODES/CACHE 1998, pp. 29-33, Seattle, Washington, March 15-18, 1998.
[120] Monica Van Horn, Angela Richter, Dian Lopez, "A Random Graph Generator", Proceedings of the 36th Midwest Instructional Computing Conference, MICS 2003, Duluth, Minnesota, April 11-13, 2003.
[121] Yue Huang, Yong-Soo Kim, "Applying Hybrid Neural Fuzzy System to Embedded System Hardware/Software Partitioning", Proceedings of the 3rd International Conference on Intelligent Computing: Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence, ICIC 2007, LNCS 4682, pp. 660-669, Qingdao, China, August 21-24, 2007.
[122] Yue Huang, Yong-Soo Kim, "Boltzmann Machine Incorporated Hybrid Neural Fuzzy System for Hardware/Software Partitioning in Embedded System Design", Proceedings of the 4th International Conference on Modeling Decisions for Artificial Intelligence, MDAI 2007, LNAI 4612, pp. 307-317, Kitakyushu, Japan, August 16-18, 2007.
[123] David Hwang, Patrick Schaumont, Yi Fan, Alireza Hodjat, Bo-Cheng Lai, Kazuo Sakiyama, Shenglin Yang, Ingrid Verbauwhede, "Design Flow for HW/SW Acceleration Transparency in the ThumbPod Secure Embedded System", Proceedings of the 40th Annual Design Automation Conference, DAC 2003, pp. 60-65, Anaheim, California, June 2-6, 2003.
[124] Toshihide Ibaraki, "Branch and Bound", Annals of Operations Research, Vol. 10, No. 1, December 1987. ISSN 0254-5330.
[125] Tetsuya Ichikawa, Tomomi Kasuya, Mitsuru Matsui, "Hardware Evaluation of the AES Finalists", Proceedings of the 3rd AES Candidate Conference, pp. 279-285, New York City, New York, April 13-14, 2000.
[126] Jaime S. Ide, Fabio G. Cozman, "Random Generation of Bayesian Networks", Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence, SBIA 2002, LNCS 2507, pp. 366-375, Porto de Galinhas/Recife, Brazil, November 11-14, 2002.
[127] Jaime S. Ide, Fabio G. Cozman, Fabio T. Ramos, "Generation of Random Bayesian Networks with Constraints on Induced Width, with Application to the Average Analysis of d-Connectivity, Quasi-random Sampling, and Loopy Propagation", Proceedings of the

16th European Conference on Artificial Intelligence, ECAI 2004, pp. 323-327, Valencia, Spain, August 22-27, 2004.
[128] S. Janssens, J. Thomas, W. Borremans, Patrick Gijsels, Ingrid Verbauwhede, Frederick Vercauteren, Bart Preneel, Joseph Vandewalle, "Hardware/Software Co-Design of an Elliptic Curve Public-Key Cryptosystem", Proceedings of the IEEE Workshop on Signal Processing Systems, SiPS 2001, pp. 209-216, Antwerp, Belgium, September 26-28, 2001.
[129] Axel Jantsch, Peteer Ellervee, Johnny Öberg, Ahmed Hemani, Hannu Tenhunen, "Hardware/Software Partitioning and Minimizing Memory Interface Traffic", Proceedings of the European Design Automation Conference, EURODAC 1994, pp. 226-231, Grenoble, France, September 19-22, 1994.
[130] Axel Jantsch, Peteer Ellervee, Johnny Oberg, Ahmed Hemani, Hannu Tenhunen, "A Software Oriented Approach to Hardware/Software Codesign", Proceedings of the Poster Session of the International Conference on Compiler Construction, CC 1994, Edinburgh, England, April 7-9, 1994.
[131] Aubin Jarry, Henri Casanova, Francine Berman, "DAGSim: A Simulator for DAG Scheduling Algorithms", Research Report No 2000-46, Laboratorie de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, Lyon, France, December, 2000.
[132] Wojciech Jędruch, "Sztuczna Inteligencja", Materiały do wykładu, Katedra Systemów Automatyki, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, Gdańsk, Polska, kwiecień 2010.
[133] Christopher Jenkins, Michael Schulte, John Glossner, „Instruction Set Extensions for the Advanced Encryption Standard on a Multithreaded Software Defined Radio Platform", International Journal of High Performance Systems Architecture, vol. 2, no. 3/4, pp. 203-214, Geneva, Switzerland, August 2010.
[134] Chenqian Jiang, Jinian Bian, Kang Zhao, Kim Tong, "A Component Library for IPCore Reuse Based System Level Design Automation", Proceedings of the 7th International Conference on Computer Aided Industrial Design and Conceptual Design, CAIDCD 2006, pp. 1-6, HangZhou, China, November 17-19, 2006. ISBN 1-4244-0683-8.
[135] Wu Jigang, Srikanthan Thambipillai, "A Branch-and-Bound Algorithm for Hardware/Software Partitioning", Proceedings of the 4th IEEE International Symposium on Signal Processing and Information Technology, ISSPIT 2004, pp. 526-529, Rome, Italy, December 18-21, 2004.
[136] Wu Jigang, Srikanthan Thambipillai, "Low-complex Dynamic Programming Algorithm for Hardware/Software Partitioning", Information Processing Letters, vol. 98, no. 2, pp. 41-46, April 30, 2006.
[137] Vincent Journot, „Evaluation of Serpent, one of the AES finalists, on 8-bit microcontrollers", Online publication, Computer Laboratory of Faculty of Computer Science and Technology of Cambridge University, Cambrdige, England, May 11, 2000.
[138] Asawaree Kalavade, Edward A. Lee, "A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem", Proceedings of the 3rd International Workshop on Hardware/Software Codesign, CODES 1994, pp. 42-48, Grenoble, France, September 22-24, 1994. ISBN 0-8186-6315-4.
[139] Asawaree Kalavade, Edward A. Lee, "The Extended Partitioning Problem: Hardware/ Software Mapping, Scheduling and Implementation-bin Selection", Journal of Design Automation for Embedded Systems, vol. 2, no. 2, pp. 125-163, Netherlands, March 1997.
[140] Geoffrey Keating, "Performance Analysis of AES Candidates on the 6805 CPU Core", Proceedings of the 2nd AES Candidate Conference, pp. 109-114, Rome, Italy, March 22-23, 1999.
[141] Brian W. Kernighan, Shen Lin, "An Efficient Heuristics Procedure for Partitioning Graphs", Bell System Technical Journal, vol. 49, no. 2, pp. 291-307, 1970.
[142] Alexander Kharry, "High-Speed FPGA Implementation of 128-bit Rijndael Symmetric Block and Stream Ciphers", Master of Science Thesis, Polytechnic University, Brooklyn, New York, June 2002.

[143] Holger M. Kienle, "Exchange Format Bibliography", ACM SIGSOFT, Software Engineering Notes vol. 26 no. 1, pp. 56-60, January 2001.
[144] Scott Kirkpatrick, C. Daniel Gelatt, Mario P. Vecchi, "Optimization by simulated annealing", Science, New Series, vol. 220, no. 4598, pp. 671-680, May 13, 1983.
[145] Darko Kirovski, Chunho Lee, Miodrag Potkojnak, William H. Mangione-Smith, "Synthesis of Power-Efficient Memory-Intensive Systems-on-Chip", Journal of IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 18, no. 9, pp. 1316-1326, September 1999.
[146] Bastian Kneer, Martin Holzer, Markus Rupp, "HW/SW Partitioning Using High Level Metrics", Proceedings of the 2nd International Conference on Computing, Communications and Control Technologies, CCCT 2004, pp. 33-38, Austin, Texas, August 14-17, 2004.
[147] Peter Voigt Knudsen, Jan Madsen, "PACE: A dynamic programming algorithm for hardware/software partitioning", Proceedings of the 4th International Workshop on Hardware/Software Codesign, CODES 1996, pp. 85-93, Pittsburgh, Pennsylvania, March 18-20, 1996.
[148] Lars R. Knudsen, "Some Thoughts on the AES Process", Public Comment on AES Candidate Algorithm – Round 1, April 15, 1999.
[149] Lars R. Knudsen, Havard Raddum, "Recommendation to NIST for the AES", Public Comments on AES Candidate Algorithms – Round 2, May 15, 2000.
[150] Paul Kocher, Joshua Jaffe, Benjamin Jun, "Introduction to differential power analysis and related attacks", Technical Report, Cryptography Research Inc., San Francisco, California, 1998.
[151] Paul Kocher, Joshua Jaffe, Benjamin Jun, "Differential power analysis", Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO 1999, LNCS 1666, pp. 388-397, Santa Barbara, California, August 15-19, 1999.
[152] Manuel Koschuch, Joachim Lechner, Andreas Weitzer, Johann Grossschaedl, Alexander Schekely, Stefan Tillich, Johannes Wolkerstorfer, "Hardware/Software Co-Design of Elliptic Curve Cryptography on an 8051 Microcontroller", Proceedings of the 8th International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006, pp. 430-444, Yokohama, Japan, October 10-13, 2006.
[153] Sławomir Kozieł, Władysław Szcześniak, "Evolutionary algorithm for electronic systems partitioning and its application in VLSI design", Proceedings of the 6th IEEE International Conference on Electronics, Circuits and Systems, ICECS 1999, vol. 3, pp. 1411-1414, Pafos, Cyprus, September 5-8, 1999. ISBN 0-7803-5682-9.
[154] Daniel Kroening, Ofer Strichman, "Decisions Procedures. An Algorithmic Point of View", Springer-Verlag, Berlin, Germany, 2008. ISBN 978-3-540-74104-6.
[155] Sandeep Kumar, Christof Paar, "Reconfigurable Instruction Set Extension for Enabling ECC on an 8-bit Processor", Proceedings of 14th International Conference on Field Programmable Logic and Application, FPL 2004, LNCS 3203, pp. 586-595, Leuven, Belgium, August 30-September 1, 2004.
[156] Henry Kuo, Ingrid Verbauwhede, Patrick Schaumont, "A 2.29 Gbits/sec, 56 mW Non-Pipelined Rijndael AES Encryption IC in a 1.8 V, 0.18 um CMOS Technology", Proceedings of the IEEE Custom Integrated Circuits Conference 2002, pp. 147-150, Orlando, Florida, May 12-15, 2002.
[157] Henry Kuo, Patrick Gijsels, W. Borremans, S. Janssens, J. Thomas, Ingrid Verbauwhede, "VLSI Implementation Strategies for Cryptography", Project Report 2000-2001 for MICRO Project 00-097, 2002.
[158] Mirosław Kutyłowski, Willy B. Strothmann, "Kryptografia. Teoria i praktyka zabezpieczania systemów komputerowych", Oficyna Wydawnicza Read Me, Warszawa, Poland, 1998. ISBN 83-7147-087-8.
[159] Elizabeth Dirkes Lagnese, Donald E. Thomas, "Architectural Partitioning for System-Level Design", Proceedings of the 26th Design Automation Conference, DAC 1989, pp. 62-67, Las Vegas, Nevada, June 25-29, 1989.

[160] Manuel Laguna, Fred Glover, "What is Tabu Search", Colorado Business Review, vol. LXI, no. 5, September 1996.
[161] Kanishka Lahiri, Anand Raghunathan, Sujit Dey, "Performance Analysis of Systems with Multi-Channel Communication Architectures", Proceedings of the 13th International Conference on VLSI Design, VLSID 2000, pp. 530-537, Calcutta, India, January 4-7, 2000.
[162] Kanishka Lahiri, Anand Raghunathan, Sujit Dey, "Fast System-Level Power Profiling for Battery Efficient System Design", Proceedings of the 10th International Symposium on Hardware/Software Codesign, CODES 2002, pp. 157-162, Estes Park, Colorado, May 6-8, 2002.
[163] Yeong-Kang Lai, Liang-Gee Chen, Jian-Yi Lai, Tai-Ming Parng, "VLSI Architecture Design and Implementation for Twofish Block Cipher", Proceedings of the International Symposium on Circuits and Systems, IEEE 2002, vol. 2, pp. 356-359, Phoenix-Scottsdale, Arizona, May 26-29, 2002.
[164] Serge Leef, „A Methodology for Virtual Hardware/Software Integration", White Paper, Mentor Graphics, Wilsonville, Oregon, 1995.
[165] Yves Le Trayon, Ghassan Al Hayek, Chantal Robach, "Testability-Oriented Hardware/Software Partitioning", Proceedings of the IEEE International Test Conference on Test and Design Validity, ITC 1996, pp. 725-731, Washington, DC, October 20-25, 1996.
[166] Yanbing Li, Tim Callahan, Ervan Darnell, Randolph Harr, Uday Kurkure, Jon Stockwood, "Hardware-Software Co-Design of Embedded Reconfigurable Architectures", Proceedings of the 37th Annual Design Automation Conference, DAC 2000, pp. 507-512, Los Angeles, California, June 5-9, 2000.
[167] Helger Lipmaa, "AES Candidates: A Survey of Implementations", Online publication for the 2nd AES Candidate Conference, Rome, Italy, March 22-23, 1999.
[168] Marisa Luiza Lopez-Vallejo, Carlos Angel Iglesias, Juan Carlos Lopez, "Applying the Propose&Revise Strategy to the Hardware-Software Partitioning Problem", Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 1998, vol. 1, LNCS 1415, pp. 169-179, Castellón, Spain, June 1-4, 1998.
[169] Marisa Lopez-Vallejo, Juan Carlos Lopez, "Multi-way Clustering Techniques for System Level Partitioning", Proceedings of the 14th Annual IEEE International ASIC/SoC Conference, pp. 242-247, Arlington, Virginia, September 12-15, 2001.
[170] Marisa Lopez-Vallejo, Juan Carlos Lopez, "On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques", Journal ACM Transactions on Design Automation of Electronic Systems, TODAES, vol. 8, no. 3, pp. 269-297, New York City, New York, July 2003.
[171] Adrian K. Lutz, Jürg Treichler, Frank K. Gürkaynak, Hubert Kaeslin, Gerard Basler, Antonina Erni, Stefan Reichmuth, Pieter Rommens, Stephan Oetiker, Wolfgang Fichtner, "2Gbit/s Hardware Realizations of Rijndael and Serpent: A Comparative Analysis", Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2002, LNCS 2523, pp. 144-158, Redwood City, California, August 13-15, 2002.
[172] Tadeusz Łuba, Bogdan Zbierzchowski, "Komputerowe projektowanie układów cyfrowych", Wydawnictwa Komunikacji i Łączności, Warszawa, Polska, 2000. ISBN 83-206-1364-7.
[173] Krzysztof Maćkowiak, „Kryptografia Kwantowa", online publication, 2006.
[174] Rabi Mahapatra, "More on Partitioning. Extended Partitioning for Embedded (Signal Processing) Applications", Lecture Notes for CPSC489-501, Texas A&M University, College Station, Texas, Fall 2000.
[175] Gayathri Manikutty, Heather Hanson, „Hardware/Software Partitioning of Synchronous Dataflow Graphs in the ACS domain of Ptolemy", Final Report, EE382 C-9: Embedded Software Systems, University of Texas, Austin, Texas, May 12, 1999.

[176] Zoltan Adam Mann, Andras Orban, Viktor Farkas, "Evaluating the Kernighan-Lin Heuristics for Hardware/Software Partitioning", International Journal of Applied Mathematics and Computer Science, vol. 17, vol. 2, no. 2, pp. 249-267, Warsaw / Zielona Gora, Poland, June 2007. ISSN 1641-876X.
[177] Peter Marwedel, „Hardware/Software Codesign", Lecture Notes for IES, Universitet Dortmund, Dortmund, Germany, 2005.
[178] Oleg Maslennikow, Michał Białko, Piotr Pawłowski, Robert Berezowski, „Przerzutniki prądowe dla logiki wielowartościowej i arytmetyki resztowej", Materiały II Krajowej Konferencji Elektroniki, KKE 2003, pp. 725-730, Kołobrzeg, Polska, 9-12 czerwiec 2003. ISSN 0033-2089.
[179] Yusuke Matsuoka, Patrick Schaumont, Kris Tiri, Ingrid Verbauwhede, „Java Cryptography on KVM and its Performance and Security Optimization using HW/SW Co-design Techniques", Proceedings of the International Conference on Compilers, Architecture and Synthesis of Embedded Systems, CASES'04, pp. 303-311, Washington, DC, September 25, 2004.
[180] Larry T. McDaniel III, "An investigation of differential power analysis on FPGA-based encryption systems", Master of Science Thesis, Virginia Polytechnic Institute, Blacksburg, Virginia, May 29, 2003.
[181] Marie McLoone, John V. McCanny, „High Performance Single-Chip FPGA Rijndael Algorithm Implementations", Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2001, LNCS 2162, pp.65-76, Paris, France, May 14-16, 2001.
[182] Guy Melacon, Isabelle Dutour, Mirelle Bousquet-Melou, "Random Generation of Dags for Graph Drawing", Technical Report INS-R0005, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, February 29, 2000.
[183] Guy Melacon, Isabelle Dutour, Mirelle Bousquet-Melou, "Random Generation of Directed Acyclic Graphs", Proceedings of the Euroconference on Combinatorics, Graph Theory and Applications, Comb 2001, Electronic Notes in Discrete Mathematics, vol. 10, pp. 202-207, Barcelona, Spain, September 12-15, 2001.
[184] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, "Handbook of Applied Cryptography", CRC Press, Boca Raton, Florida, October 16, 1996. ISBN 0-8493-8523-7.
[185] Microsoft Corporation, "GPGPU Computing Horizons: Developing and Deploying for Microsoft Windows", Microsoft HPC Whitepaper, Redmond, Washington, May, 2010.
[186] Randi Moe, Tor Sørevik, „Parallel Branch and Bound Algorithms on Internet Connected Workstations", Proceedings of the 1st International Conference on High Performance Computing and Communications, HPCC 2005, LNCS 3726, pp. 768-775, Sorrento, Italy, September 21-23, 2005.
[187] Emeka Mosanya, Christof Teuscher, Hector Fabio Restrepo, Patrick Galley, Eduardo Sanchez, „CryptoBooster: A Reconfigurable and Modular Cryptographic Coprocessor", Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems, CHES 1999, LNCS 1717, pp. 246-256, Worcester, Massachusetts, August 12-13, 1999.
[188] Piotr Mroczkowski, „Implementation of the block cipher Rijndael using Altera FPGA", Public Comments on AES Candidate Algorithms – Round 2, May 10, 2000.
[189] Allen Newell, „The knowledge level", Artificial Intelligence, vol. 18, no. 1, pp. 87-127, 1982.
[190] Ralf Niemann, Peter Marwedel, "Hardware/Software Partitioning Using Integer Programming", Proceedings of the European Conference on Design and Test, EDTC 1996, pp. 473-480, Paris, France, March 11-14, 1996.
[191] Ralf Niemann, Peter Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming", Design Automation for Embedded Systems, special issue: Partitioning Methods for Embedded Systems, vol. 2, no. 2, pp. 165-193, March 1997.

[192] Ralf Niemann, "Hardware/Software Co-Design for Data Flow Dominated Embedded Systems", Kluwer Academic Publishers, Boston, Massachusetts, 1998. ISBN 0-7923-8299-4.
[193] Vladimir I. Norkin, Georg Ch. Pflug, Andrzej Ruszczyński, "A Branch and Bound Method for Stochastic Global Optimization", Journal of Mathematical Programming, vol. 83, no. 3, pp. 425-450, November 1, 1998.
[194] Hynuok Oh, Soonhoi Ha, „Partitioning Framework For Less Restricted Partitioning Problems", Proceedings of the 6th International Conference on VLSI and CAD, ICVC 1999, pp. 99-102, Seoul, South Korea, October 26-27, 1999.
[195] John T. Olson, Jerzy W. Rozenblit, "Framework For Hardware/Software Partitioning Utilizing Bayesian Belief Networks", Proceedings of the International Conference on Systems, Man and Cybernetics, vol. 4, pp. 3983-3988, San Diego, California, October 11-14, 1998. ISBN 0-7803-4778-1.
[196] John T. Olson, Jerzy W. Rozenblit, Witold Jacak, "Hardware/Software Partitioning Using Bayesian Belief Networks", IEEE Transactions on Systems, Man and Cybernetics, part A: Systems and Humans, vol. 37, no. 5, pp. 655-668, Edmonton, Canada, September 2007.
[197] Billy Ornellas, William Yip, „Twofish Encryption Algorithm on Field Programmable Gate Arrays (FPGAs)", EE496 Spring 2000 Design Project, Department of Electrical Engineering, University of Hawaii, Manoa, Hawaii, May 2000.
[198] Siddika Berna Örs, Frank Gürkaynak, Elisabeth Oswald, Bart Preneel, "Power-analysis attack on an ASIC AES implementation", Proceedings of the International Conference on Information Technology: Coding and Computing, ITCC 2004, pp. 546-552, Las Vegas, Nevada, April 5-7, 2004.
[199] Dag Arne Osvik, „Speeding up Serpent", Proceedings of the 3rd AES Candidate Conference, pp. 317-330, New York City, New York, April 13-14, 2000.
[200] David A. Papa, Igor L. Markov, „Hypergraph Partitioning and Clustering", In Approximation Algorithms and Metaheuristics, Teofilo Gonzalez, ed., pp. 61.1-61.19, Chapman Hall / CRC Press, May 15, 2007. ISBN 9781584885504.
[201] Hadi Parandeh-Afshar, Mohsen Yousefpour, Ali Tootoonchian, Mahmoud Reza Hashemi, Omid Fatemi, „A Window-Based Automatic Hardware/Software Partitioning Heuristics", The Arabian Journal for Science and Engineering, vol. 32, no. 2C, pp. 27-40, Dhahran, Saudi Arabia, December 2007.
[202] Sudeep Pasricha, Nikil Dutt, "COSMECA: Application Specific Co-Synthesis of Memory and Communication Architectures for MPSoC", Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, pp. 700-705, Munich, Germany, March 6-10, 2006.
[203] Ranjetta M. Patil, "Hardware Implementation of Twofish Block Cipher", ECE 646 (Cryptography and Computer Network Security) Project Report, George Mason University, Fairfax, Virginia, 2005.
[204] Cameron Patterson, "A Dynamic FPGA Implementation of the Serpent Block Cipher", Proceedings of the 2nd International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, LNCS 1965, pp. 141-155, Worcester, Massachusetts, August 17-18, 2000.
[205] David J. Pearce, Paul H. J. Kelly, "A Batch Algorithm for Maintaining a Topological Order", Proceedings of the 33rd Australasian Conference on Computer Science, ACSC 2010, vol. 102, pp. 79-88, Brisbane, Australia, January 18-22, 2010.
[206] Matthew Pierson, Bryan Brady, "Low cost differential power analysis (DPA) resistant crypto-chips", EE 244 Project Presentation, University of California, Berkeley, California, 2006.
[207] Luigi Pomante, "Co-Design of Multiprocessor Embedded Systems: an Heuristic Multi-Level Partitioning Methodology", Proceedings of the 16th IFIP World Computer Congress on Chip Design Automation, ICDA 2000, pp. 421-425, Beijing, China, August 2000.
[208] Shiv Prakash, Alice C. Parker, "SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems", Proceedings of the 19th Annual International Symposium on

Computer Architecture, ISCA 1992, pp. 434-434, Queensland, Australia, May 19-21, 1992. ISBN 0-89791-509-7

[209] Steven Prestwich, "Three Implementations of Branch and Bound in CLP", Proceedings of the 4th Compulog-Net Workshop on Parallelism and Implementation Technologies, Bonn, Germany, September 6, 1996.

[210] Moshin Riaz, Howard M. Heys, "The FPGA Implementation of the RC6 and CAST-256 Encryption Algorithms", Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 1999, vol. 1, pp. 367-372, Edmonton, Canada, May 9-12, 1999.

[211] Ronald L. Rivest, Matt J.B. Robshaw, Ray Sidney, Yiqun Lisa Yin, "The RC6™ Block Cipher", Proceedings of the 1st AES Candidate Conference, Ventura, California, August 20-22, 1998.

[212] Ander Royo, Javier Moran, Juan Carlos Lopez, "Design and Implementation of a Coprocessor for Cryptography Applications", Proceedings of the European Conference on Design and Test, EDTC 1997, pp. 213-217, Paris, France, March 17-20, 1997.

[213] Danuta Rutkowska, Maciej Piliński, Leszek Rutkowski, "Sieci Neuronowe, Algorytmy Genetyczne i Systemy Rozmyte", Wydawnictwo Naukowe PWN, Warszawa, Polska, 1999. ISBN 83-01-12304-4.

[214] Roman Rykaczewski, „Kryptograficzne zabezpieczenie transmisji", Materiały do wykładu, Katedra Sieci Teleinformacyjnych, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, Gdańsk, Polska, 2009.

[215] Kazuo Sakiyama, Lejla Batina, Patrick Schaumont, Ingrid Verbauwhede, "HW/SW Co-design of TA/SAP-resistant Public-key Cryptosystems", Proceedings of the Workshop on Cryptographic Advances in Secure Hardware, CRASH 2005, Leuven, Belgium, September 6-7, 2005.

[216] Kazuo Sakiyama, Lejla Batina, Bart Preneel, Ingrid Verbauwhede, "HW/SW Co-design for Accelerating Public-Key Cryptosystems over GF(p) on the 8051 µ-controller", Proceedings of the World Automation Congress, WAC 2006, Special Session on Information Security and Hardware Implementations, pp. 1-6, Budapest, Hungary, July 24-26, 2006.

[217] Kazuo Sakiyama, Lejla Batina, Bart Preneel, Ingrid Verbauwhede, "Superscalar Coprocessor for High-speed Curve-based Cryptography", Proceedings of the 8th International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006, pp. 415-429, Yokohama, Japan, October 10-13, 2006.

[218] Fumihiko Sano, Masanobu Koike, Shinichi Kawamura, Masue Shiba, "Performance Evaluation of AES Finalists on the High-End Smart Card", Proceedings of the 3rd AES Candidate Conference, pp. 82-93, New York City, New York, April 13-14, 2000.

[219] Akashi Satoh, Nobuyuki Ooba, Kohji Takano, Edward D'Avignon "High-Speed MARS Hardware", Proceedings of the 3rd AES Candidate Conference, pp. 305-316, New York City, New York, April 13-14, 2000.

[220] John J.G. Savard, "The Advanced Encryption Standard (Rijndael)", online publication available at http://www.quadibloc.com/crypto/co040401.htm.

[221] Hanno Scharwaechter, David Kammler, Andreas Wieferink, Manuel Hohenauer, Kingshuk Karuri, Jianjiang Ceng, Reiner Leupers, Gerd Ascheid, Heinrich Meyr, "ASIP Architecture Exploration for Efficient IPSec Encryption: A Case Study", ACM Transactions on Embedded Computing Systems, vol. 6, no. 2, article 12, May 2007.

[222] Patrick Schaumont, Ingrid Verbauwhede, "Domain-Specific Codesign for Embedded Security", Journal of Computer, vol. 36, no. 4, pp. 68-74, Los Alamitos, California, April 2003.

[223] Patrick Schaumont, Ingrid Verbauwhede, "Hardware Software Codesign for Stream Ciphers", Proceedings of the ECRYPT Workshop: State of the Art of Stream Ciphers, SACS 2007, pp. 106-116, Bochum, Germany, January 30-February 1, 2007.

[224] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, "Twofish: A 128-Bit Block Cipher", Proceedings of the 1st AES Candidate Conference, Ventura, California, August 20-22, 1998.

[225] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, "Twofish, A Block Encryption Algorithm", Presentation in the 1st AES Candidate Conference, Ventura, California, August 20-22, 1998.
[226] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, "Performance Comparison of the AES Submissions", Proceedings of the 2nd AES Candidate Conference, pp. 15-34, Rome, Italy, March 22-23, 1999.
[227] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, "Twofish: New Results", Presentation in the 2nd AES Candidate Conference, Rome, Italy, March 22-23, 1999.
[228] Bruce Schneier, Doug Whiting, "A Performance Comparison of the Five AES Finalists", Proceedings of the 3rd AES Candidate Conference, pp. 123-135, New York City, New York, April 13-14, 2000.
[229] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Niels Ferguson, "Comments on Twofish as an AES Candidate", Proceedings of the 3rd AES Candidate Conference, pp. 355-356, New York City, New York, April 13-14, 2000.
[230] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, Tadayoshi Kohno, Mike Stay, "The Twofish Team's Final Comments on AES Selection", Public Comments on AES Candidate Algorithms – Round 2, May 15, 2000.
[231] Aviral Shrivastava, Mohit Kumar, Sanjiv Kapoor, Shashi Kumar, M. Balakrishnan, "Optimal Hardware/Software Partitioning for Concurrent Specification using Dynamic Programming", Proceedings of the 13th International Conference on Very Large Scale Integration (VLSI) Design, VLSID 2000, pp. 110-113, Calcutta, India, January 4-7, 2000.
[232] Martin Simka, Viktor Fischer, Milos Drutarovsky, "Hardware-Software Codesign in Embedded Asymmetric Cryptography Application – a Case Study", Proceedings of the 13th International Conference on Field Programmable Logic and Applications, FPL 2003, pp. 1075-1078, Lisbon, Portugal, September 1-3, 2003.
[233] Kamil Skowroński, „Zastosowanie metod hardware software codesign w projektowaniu systemów cyfrowych", Praca Dyplomowa, Wydział Elektroniki Telekomunikacji i Informatyki Politechniki Gdańskiej, Gdańsk, Polska, sierpień 2001.
[234] Zbigniew Skowroński, „Dekompozycja Systemów Mikroprocesorowych Współpracujących ze Specjalizowanymi Układami Cyfrowymi na Część Sprzętową i Programową", Materiały I Krajowej Konferencji Reprogramowalne Układy Cyfrowe, RUC 1998, pp.75-82, Szczecin, Polska, 12-13 marca 1998.
[235] Frank Slomka, Matthias Dorfel, Ralf Munzenberger, Richard Hofmann, „Hardware/Software Codesign and Rapid Prototyping of Embedded Systems", Journal of IEEE Design & Test, vol. 17, no. 2, pp. 28-38, April 2000.
[236] Adam Słowik, Michał Białko, "Ewolucyjne projektowanie filtrów cyfrowych IIR o nietypowych charakterystykach amplitudowych", Materiały III Krajowej Konferencji Elektroniki, KKE 2004, pp. 345-350, Kołobrzeg, Polska, 16-18 czerwca 2004.
[237] Adam Słowik, Michał Białko,"Ewolucyjne projektowanie i optymalizacja kombinacyjnych układów cyfrowych ze względu na liczbę tranzystorów", Materiały IV Krajowej Konferencji Elektroniki, KKE 2005, pp. 207-212, Darłówko Wschodnie, Polska, 12-15 czerwca 2005.
[238] Juha Pekka Soininen, „A NOC Design Methodology", Systems on Chip Workshop, Villach, Austria, September 17, 2001.
[239] Przemysław Sołtan, Natalia Maslennikow, Oleg Maslennikow, „Modelowanie reprogramowalnych układów prądowych pracujących w logice wielowartościowej", Materiały XII Krajowej Konferencji Komputerowe Wspomaganie Badań Naukowych, KOWBAN 2005, pp. 133-138, Polanica Zdrój, Polska, 26-28 października 2005. ISBN 83-7374-036-8.
[240] Vinoo Nat Srinivasan, Sriram Radhakrishnan, Ranga Vemuri, „Hardware Software Partitioning with Integrated Hardware Design Space Exploration", Proceedings of the Conference on Design, Automation and Test in Europe, DATE 1998, pp. 28-35, Paris, France, February 23-26, 1998.

[241] Greg Stitt, Roman Lysecky, Frank Vahid, „Dynamic Hardware/Software Partitioning: A First Approach", Proceedings of the 40th Design Automation Conference, DAC 2003, pp. 250-255, Anaheim, California, June 2-6, 2003.
[242] Marek Strachacki, „Analiza i realizacja w układach FPGA algorytmów Rijndael, Serpent i Twofish", Praca Dyplomowa, Wydział Elektroniki Telekomunikacji i Informatyki Politechniki Gdańskiej, Gdańsk, Polska, grudzień 2001.
[243] Marek Strachacki, „Programowalne układy FPGA w sprzętowych implementacjach algorytmów kryptograficznych", Seminarium Katedralne, Wydział Elektroniki Telekomunikacji i Informatyki Politechniki Gdańskiej, Gdańsk, Polska, 16 grudnia 2002.
[244] Marek Strachacki, „Realizacja algorytmów szyfrowania symetrycznego w układach FPGA Xilinx Virtex II", Materiały VI Krajowej Konferencji Naukowej Reprogramowalne Układy Cyfrowe, RUC 2003, pp. 171-178, Szczecin, Polska, 8-9 maja 2003.
[245] Marek Strachacki, „Zastosowanie układów FPGA w kryptografii", Materiały X Krajowej Konferencji Komputerowe Wspomaganie Badań Naukowych, KOWBAN 2003, pp. 281-286, Polanica Zdrój, Polska, 22-24 października 2003.
[246] Marek Strachacki, „Przegląd metod współbieżnej symulacji heterogenicznych systemów wbudowanych", Materiały VII Krajowej Konferencji Naukowej Reprogramowalne Układy Cyfrowe, RUC 2004, pp. 133-140, Szczecin, Polska, 13-14 maja 2004.
[247] Marek Strachacki, „Realizacja środowiska współbieżnej symulacji i emulacji heterogenicznych systemów wbudowanych", Materiały VIII Krajowej Konferencji Naukowej Reprogramowalne Układy Cyfrowe, RUC 2005, pp. 223-230, Szczecin, Polska, 12-13 maja 2005.
[248] Marek Strachacki, Lech Dembek, „Heterogeniczna realizacja algorytmu AES w układzie SoC FPSLIC z zastosowaniem technik projektowania sprzętowo-programowego", Materiały III Krajowej Konferencji Technologie Informacyjne, TI 2005, pp. 787-794, Gdańsk, Polska, 22-25 maja 2005.
[249] Marek Strachacki, „Zastosowanie metody podziału i ograniczeń w wielokryterialnym problemie podziału HW/SW do implementacji algorytmów kryptograficznych", Materiały V Krajowej Konferencji Technologie Informacyjne, TI 2007, pp. 593-600, Gdańsk, Polska, 21-23 maja 2007.
[250] Marek Strachacki, Robert Piotrowski, „Metody sprzętowej implementacji kryptografii odpornej na kryptoanalizę", Materiały V Krajowej Konferencji Technologie Informacyjne, TI 2007, pp. 601-608, Gdańsk, Polska, 21-23 maja 2007.
[251] Marek Strachacki, „Speedup of Branch and Bound Method for Hardware/Software Partitioning", 1st International Conference on Information Technology, TI 2008, pp. 1-4, Gdańsk, Poland, May 18-21, 2008.
[252] Marek Strachacki, Stanisław Szczepański, „Implementation of AES Algorithm Resistant to DPA Analysis", Proceedings of the 15th IEEE International Conference On Electronics, Circuits and Systems, ICESC 2008, pp. 214-217, San Julian, Malta, August 31-September 3, 2008.
[253] Marek Strachacki, Stanisław Szczepański, „Power Equalization of AES FPGA Implementation", Bulletin of the Polish Academy of Sciences, Technical Sciences, vol. 58, no. 1, pp. 125-128, Warszawa, Polska, marzec 2010.
[254] Masahiko Takenaka, Naoya Torii, Kouichi Itoh, Jun Yajima, „Performance Comparison of 5 AES Candidates with New Performance Evaluation Tool", Proceedings of the 3rd AES Candidate Conference, pp. 123-135, New York City, New York, April 13-14, 2000.
[255] R. Reed Taylor, Seth Copen Goldstein, "A High-Performance Flexible Architecture for Cryptography", Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems, CHES 1999, LNCS 1717, pp. 231-245, Worcester, Massachusetts, August 12-13, 1999.
[256] Russel Tessier, „Reconfigurable Computing, Lecture 22: Hardware/Software Codesign: Automatic Compilation to Reconfigurable Coprocessors", Lecture Notes for ECE 697F, University of Massachusetts, Amherst, Massachusetts, November 29, 2004.

[257] Stefan Tillich, Martin Feldhofer, Thomas Popp, Johann Großschädl, „Area, Delay, and Power Characteristics of Standard-Cell Implementation of the AES S-Box", Journal of Signal Processing Systems, vol. 50, no. 2, pp. 251-261, January 17, 2008.

[258] Kris Tiri, Ingrid Verbauwhede, "Synthesis of Secure FPGA Implementations", Proceedings of the 13th International Workshop on Logic and Synthesis, IWLS 2004, pp. 224-231, Temecula, California, June 2-4, 2004.

[259] Vivek Tiwari, Sharad Malik, Andrew Wolfe, „Power Analysis of Embedded Software: A First Step Towards Software Power Minimization", Journal of IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 4, pp. 437-445, Piscataway, New Jersey, December 1994.

[260] Frank Vahid, Daniel Gajski, Jie Gong, „A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning", Proceedings of the IEEE/ACM European Design Automation Conference, EuroDAC 1994, pp. 214-219, Grenoble, France, September 19-22, 1994.

[261] Frank Vahid, Daniel Gajski, „Clustering for improved system-level functional partitioning", Proceedings of the 8th International Symposium on System Synthesis, ISSS 1995, pp. 28-35, Cannes, France, September 13-15, 1995.

[262] Frank Vahid, "Modifying Min-Cut for Hardware and Software Functional Partitioning", Proceedings of the 5th International Workshop on Hardware-Software Co-Design, CODES 1997, pp. 43-48, Braunschweig, Germany, March 24-26, 1997. ISBN: 0-8186-7895-X.

[263] Frank Vahid, Thuy Dm Le, "Extending the Kernighan/Lin Heuristics for Hardware and Software Functional Partitioning", Journal of Design Automation for Embedded Systems, vol. 2, no. 2, pp. 237-261, Springer Netherlands, March 1997. ISSN 0929-5585.

[264] Fabian Vargas, Eduardo Augusto Bezerra, L. Wulff, Daniel Barros Jr. „Optimizing HW/SW Codesign Towards Reliability for Critical-Application Systems", Proceedings of the 7th Asian Test Symposium, ATS 1998, pp. 52-57, Singapore, December 2-4, 1998.

[265] Ingrid Verbauwhede, „Real-Time Embedded Processors: Architectures and Design Methods", Project Report 2001-2002 for MICRO Project 01-090, University of California, Los Angeles, California, 2003.

[266] Krzysztof Walkowiak, „The Branch and Bound Algorithm for a Backup Virtual Path Assignment in Survivable ATM Networks", International Journal of Applied Math in Computer Science, vol.12, no. 2, pp. 257-267, March 10, 2002.

[267] Gang Wang, Wenrui Gong, Ryan Kastner, „A new approach for Task Level Computational Resource Bi-partitioning", Proceedings of the International Conference on Parallel and Distributed Computing and Systems, PDCS 2003, Marina del Rey, California, November 3-5, 2003.

[268] Perry H. Wang, Jamison D. Collins, Gautham N. Chinya, Hong Jiang, Xinmin Tian, Milind Girkar, Nick Y. Yang, Guei-Yuan Lueh, Hong Wang, „EXOCHI: Architecture and Programming Environment for A Heterogeneous Multi-core Multithreaded System", Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2007, pp. 156-166, San Diego, California, June 11-13, 2007.

[269] Nicholas C. Weaver, John Wawrzynek, „A Comparison of the AES Candidates Amenability to FPGA Implementation", Proceedings of the 3rd AES Candidate Conference, pp. 28-39, New York City, New York, April 13-14, 2000.

[270] Brian Weeks, Mark Bean, Tom Rozylowicz, Chris Ficke, „Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithm", Proceedings of the 3rd AES Candidate Conference, pp. 286-304, New York City, New York, April 13-14, 2000.

[271] Doug Whiting, Bruce Schneier, „Improved Twofish Implementations", Twofish Technical Report #3, Counterpane Systems, December 2, 1998.

[272] Theerayod Wiangtong, Peter Y.K. Cheung, Wayne Luk, „Tabu Search with Intensification Strategy for Functional Partitioning in Hardware-Software Codesign", Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2002, pp. 297-298, Napa, California, April 22-24, 2002.

[273] Theerayod Wiangtong, Peter Y.K. Cheung, Wayne Luk, „Cluster-Driven Hardware/Software Partitioning and Scheduling Approach for a Reconfigurable Computer System", Proceedings of the 13th Conference on Field Programmable Logic and Application, FPL 2003, LNCS 2778, pp. 1071-1074, Lisbon, Portugal, September 1-3, 2003.
[274] Wayne Wolf, „A Decade of Hardware/Software Codesign", IEEE Computer, vol. 36, no. 4, pp. 38-43, April 2003.
[275] Lisa Wu, "Fast Flexible Architectures for Secure Communication", Master of Science Thesis, Advanced Computer Architecture Laboratory, University of Michigan, Ann Arbor, Michigan, April 22, 2001.
[276] Shu Xiao, Edmund M.-K. Lai, "A Branch and Bound Algorithm for Fast Power-Aware Scheduling of VLIW Instructions", School of Computer Engineering, Nanyang Technological University, Singapore, 2003.
[277] Xilinx Inc., "Virtex-II Platform FPGAs: Complete Data Sheet", Product Specification, DS031 (v3.5), San Jose, California, November 5, 2007.
[278] Xun Xiong, Peter Gutberlet, Wolfgang Rosenstiel, "Automatic Generation of Interprocess Communication in the PARAGON System", Proceedings of the 7th IEEE International Workshop on Rapid System Prototyping, RSP 1996, pp.24-29, Thessaloniki, Greece, June 19-21, 1996.
[279] Jun Yajima, Masahiko Takenaka, Naoya Torii, Kouichi Itoh, „Results of New Boolean Functions Search for Serpent S-boxes", Public Comments on AES Candidate Algorithms – Round 2, May 11, 2000.
[280] Ti-Yen Yen, Wayne Wolf, "Communication Synthesis for Distributed Embedded Systems", Proceedings of the IEEE International Conference on Computer Aided-Design, ICCAD 1995, pp. 288-294, San Jose, California, November 5-9, 1995.
[281] Lotfi Askar Zadeh, "Fuzzy Sets", Information and Control, vol. 8, no. 3, pp. 338-353, 1965. ISSN 0019-9958.

8. Dodatek A: Wprowadzenie do algorytmów kryptograficznych

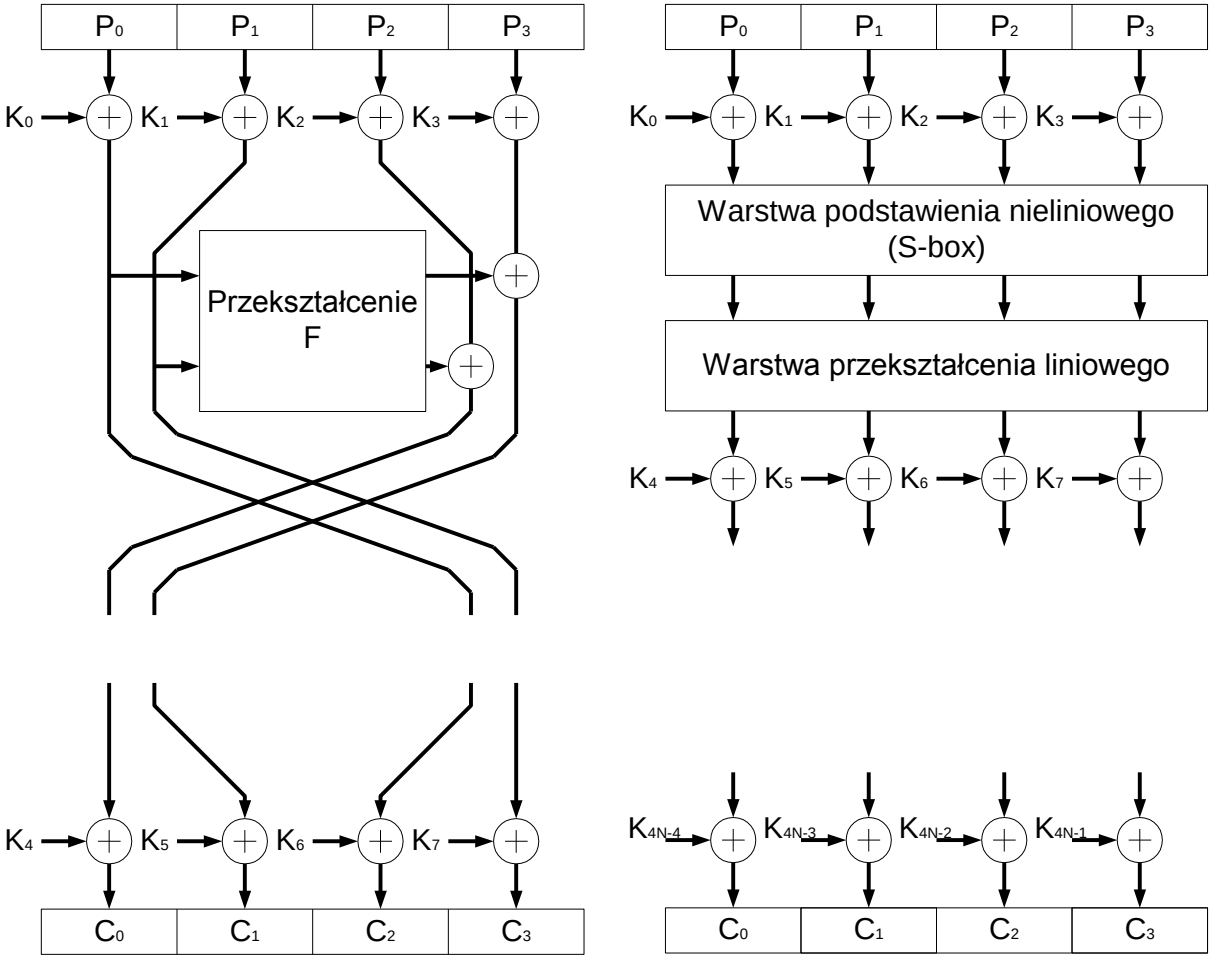
Tradycyjnie algorytmy kryptograficzne ze względu na właściwości kluczy można podzielić na [158][184][214][243]:

- algorytmy symetryczne (z kluczem prywatnym), w których ten sam tajny klucz służy do szyfrowania jak i do odszyfrowania (symetria klucza),
- algorytmy asymetryczne (z kluczem publicznym), w których szyfrowanie i odszyfrowanie dokonywane są różnymi kluczami komplementarnymi, z których jeden jest tajny, a drugi publicznie dostępny (asymetria klucza).

Algorytmy asymetryczne służą najczęściej do ustalania tajnego klucza sesji dla komunikujących się stron. Sama komunikacja pomiędzy stronami szyfrowana jest za pomocą algorytmów symetrycznych z tajnym kluczem sesji.

8.1. Struktury algorytmów symetrycznych

Struktury algorytmów symetrycznych przedstawia Rysunek 18 [243]:



Rysunek 18. Schemat sieci Feistela i sieci permutacyjno-podstawieniowej [243]

- sieć Feistela [214][224][242][243]:
  - szyfrowanie:

$$(A_1, B_1) = (B_0 \oplus F(A_0, K_0), A_0) \tag{30}$$

  - deszyfrowanie:

$$(A_0, B_0) = \left(B_{1,} A_1 \oplus F(B_1, K_0)\right) \tag{31}$$

gdzie:

A_0, B_0 – wektor wejściowy dwóch słów N-bitowych,
A_1, B_1 – wektor wyjściowy dwóch słów N-bitowych,
K_0 – klucz rundy,
F – przekształcenie zależne od klucza.

- w jednej rundzie dokonuje naprzemiennych przekształceń części stanu (zależność (30)),
- zależne od klucza przekształcenie F nie musi być bijekcją, gdyż do deszyfrowania nie jest wykorzystywane przekształcenie odwrotne (zależność (31)),
- szyfrowanie i odszyfrowanie wykorzystuje tą samą strukturę, a jedyna różnica to podawanie kluczy w odwrotnej kolejności,
- przykładami są DES, IDEA, MARS, RC5, RC6, Blowfish, Twofish,

- sieć permutacyjno-podstawieniowa [64][214][242][243]:
  - w jednej rundzie przekształceniu podlega cały stan,
  - przekształcenie rundy musi być bijekcją, gdyż do deszyfrowania jest wykorzystywane przekształcenie odwrotne,
  - przekształcenie rundy składa się 3 warstw operacji: podstawienia nieliniowego SBox, liniowej warstwy rozpraszającej, dodawania klucza,
  - odszyfrowanie wymaga algorytmu odwrotnego z zastosowaniem operacji odwrotnych i podawania kluczy w odwrotnej kolejności,
  - przykładami są Serpent, Rijndael.

8.2. Współczesne symetryczne algorytmy kryptograficzne

Od 1977 roku algorytm DES opracowany przez IBM był najszerzej stosowanym symetrycznym algorytmem kryptograficznym. DES jest 16-rundową siecią Feistela o długości bloku 64 bity i efektywnej długości klucza 56 bitów [214]. Z powodu zbyt krótkiego klucza DES, postępów w kryptoanalizie i nieefektywności realizacji programowych algorytmu, w 1997 roku NIST zainicjował program publicznego wyboru nowego szyfru blokowego AES, który miał zastąpić DES [214][242]. W wyniku dwóch konferencji – w 1998 roku w Ventura i w 1999 roku w Rzymie – wyłoniono pięć algorytmów finalistów konkursu AES:

- MARS, autorstwa zespołu IBM (16-rundowa sieć Feistela),
- RC6, zaproponowany przez zespół pod przewodnictwem Ronalda Rivesta (20-rundowa sieć Feistela),
- Rijndael, zgłoszony przez Joana Daemena i Vincenta Rijmena (10-rundowa sieć permutacyjno-podstawieniowa) – algorytm ten został wybrany jako AES w listopadzie 2000 roku,
- Serpent, autorstwa Rossa Andersona, Eli Bihama i Larsa Knudsena (32-rundowa sieć permutacyjno-podstawieniowa),
- Twofish, zaproponowany przez zespół pod przewodnictwem Bruce Schneiera (16-rundowa sieć Feistela).

Algorytmy rundy finałowej AES charakteryzują bieżące trendy w projektowaniu szyfrów symetrycznych pod kątem struktur, w podejściu do kwestii bezpieczeństwa i wykorzystują następujące operacje podstawowe [82][95][242][243] (Tabela 8):

- dodawanie modulo 2 (XOR),
- permutacje, stałe przesunięcia, stałe rotacje,

- zmienne przesunięcia, zmienne rotacje,
- podstawienia SBox,
- mnożenie przez stałą w ciele GF(2^8),
- 32-bitowe dodawanie i odejmowanie arytmetyczne,
- 32-bitowe mnożenie arytmetyczne.

Tabela 8. Operacje wykorzystywane w algorytmach kryptograficznych [82][242][243]

Algorytm	XOR	Dodawanie Mod 2^{32}	Odejmowanie Mod 2^{32}	Stałe Przesunięcia	Zmienne Rotacje	Mnożenie Mod 2^{32}	Mnożenie GF(2^8)	LUT
MARS	+	+	+	+	+	+		+
RC6	+	+		+	+	+		
Rijndael	+			+			+	+
Serpent	+			+				+
Twofish	+	+		+			+	+

Poniższe studium przypadków opisuje algorytmy rundy finałowej, ich podstawowe parametry – długość bloku, długość klucza, ilość kluczy rundy, ilość rund, następnie podstawowe operacje i krótki przegląd kryptoanalizy. Przy opisie realizacji programowych rozważana jest możliwość wykorzystania instrukcji MMX na Pentium, teoretyczna równoległość algorytmu na poziomie instrukcji, a także charakterystyka realizacji na procesorach 8-bitowych. Przy opisie realizacji sprzętowych zastosowano następujące oznaczenia do oznaczenia architektury [82][96][242][243][245]:

- LU *n* – iteracyjna z rozwinięciem *n* rund, po których zastosowano pojedynczy stopień rejestrowy, jednocześnie szyfrowany jest 1 blok,
- Pipelined *n* – potokowa zewnętrzna z rozwinięciem *n* rund, po każdej zastosowano stopień rejestrowy, jednocześnie szyfrowanych jest *n* bloków,
- Pipelined *n/k* – potokowa mieszana z rozwinięciem *n* rund, po każdej zastosowano stopień rejestrowy, i *k* dodatkowych stopniach potoku wewnątrz każdej rundy, jednocześnie szyfrowanych jest *n*k* bloków.

Zużycie zasobów podawano dla układów ASIC w mm^2, liczbie tranzystorów lub komórek, a dla układów FPGA w liczbie LE/LC [4] lub CLB [277]. Czas generacji klucza równy 0 oznacza generację klucza współbieżnie do szyfrowania. Przepustowość szyfrowania podawano dla trybów bez sprzężenia zwrotnego. W najczęściej stosowanych trybach ze sprzężeniem zwrotnym realizacje potokowe osiągną wydajność proporcjonalnie mniejszą.

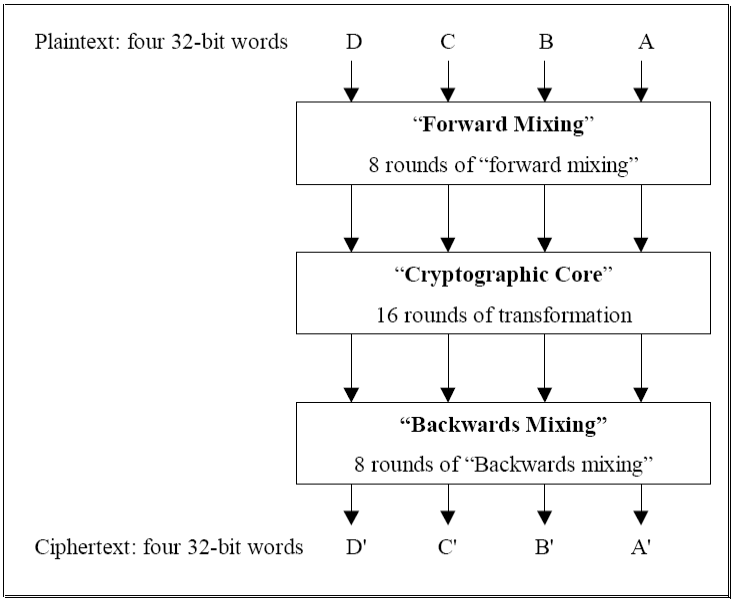
8.3. Algorytm MARS

Algorytm MARS [47] został zaprojektowany przez zespół IBM. MARS przetwarza 128-bitowy blok danych w 8 rundach mieszających w przód, 16 rundach szyfrujących i 8 rundach mieszających wstecz (patrz Rysunek 19). MARS jest siecią Feistela typu 3, w której jedno słowo danych modyfikuje pozostałe 3 słowa danych (patrz Rysunek 20). Klucz szyfrowania może być długości 128 do 448 bitów, z którego generowanych jest 40 128-bitowych kluczy rundy. Algorytm używa silnych operacji 32-bitowych oferowanych przez współczesne procesory:

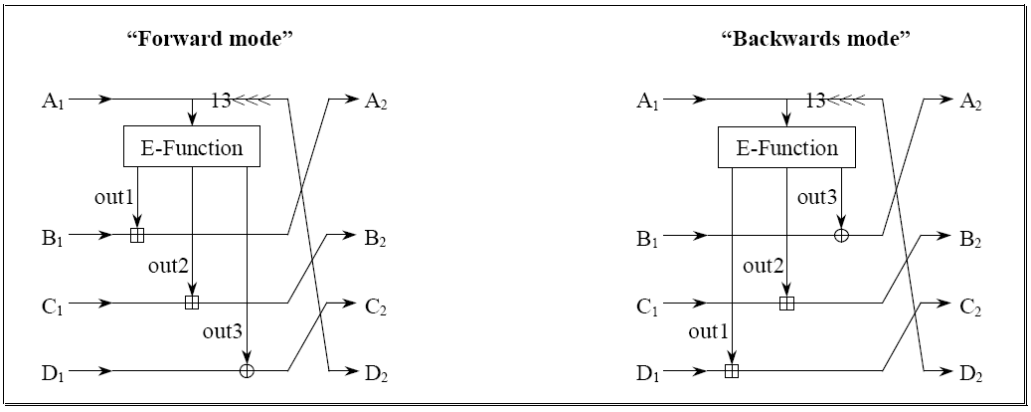
- dodawanie i odejmowanie arytmetyczne modulo 2^{32},
- mnożenie arytmetyczne modulo 2^{32},
- logiczna suma wyłączająca (XOR),
- przesunięcia cykliczne w lewo i prawo zależne od danych,
- tablica lookup (SBox) 512 słów 32-bitowych.

Algorytm MARS jest złożonym algorytmem do kryptoanalizy, choć używa operacji i struktur o znanych charakterystykach [47]. Heterogeniczna struktura algorytmu oraz

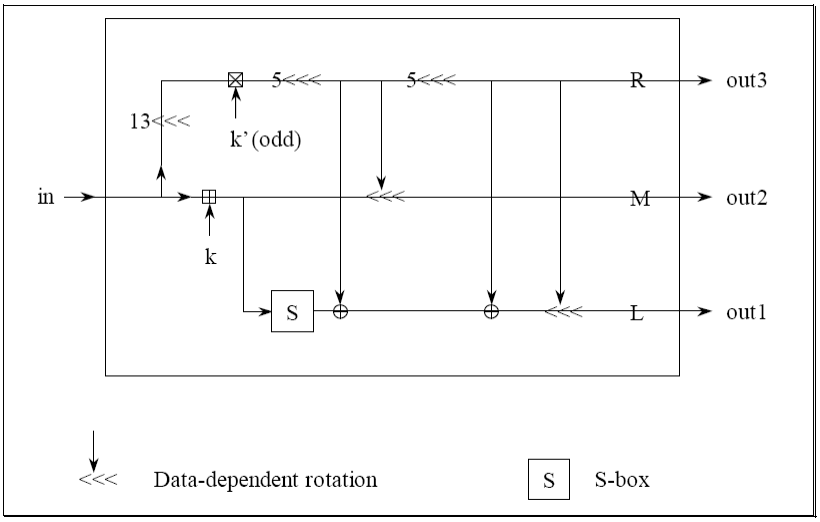
użycie wielu silnych operacji ma zabezpieczać przed znanymi atakami [48][63]. SBoxy okazały się nieco słabsze niż zakładali autorzy algorytmu [149]. Na podstawie kryptoanalizy stopień bezpieczeństwa oszacowano na 1,90 [229], choć w innych pracach [149] uznawany jest za trudny do oszacowania.



Rysunek 19. Schemat blokowy algorytmu MARS [47]



Rysunek 20. Sieć Feistela typu 3 algorytmu MARS [47]



Rysunek 21. E-Function algorytmu MARS [47]

8.3.1. Programowa realizacja algorytmu

Algorytm MARS jest złożonym algorytmem do realizacji programowej, choć jego specyfikacja w języku C jest druga najkrótsza (po algorytmie RC6) [63]. Algorytm wyznaczania kluczy rundy jest dodatkowo utrudniony przez procedurę sprawdzania słabych kluczy. Złożone operacje używane przez MARS nie wykorzystują instrukcji typu MMX [8], a operacja mnożenia może zmniejszać wydajność na niektórych procesorach [101]. Wydajność szyfrowania i generacji kluczy rundy jest niezależna od długości klucza [228]. Efektywna równoległość algorytmu na poziomie instrukcji wynosi 2,36 [61] (214 instrukcje spośród 504 są na ścieżce krytycznej). Wykonanie algorytmu na maszynie typu „dataflow" pozwala na przyspieszenie algorytmu MARS o 29% [29][46][275], co wskazuje na mały stopień równoległości operacji z powodu wysokiej zależności danych w E-Function [8][228].

Tabela 9. Realizacje algorytmu MARS na procesorze Pentium

Źródło	Rozmiar programu [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania	Liczba cykli deszyfrowania
[8]	---	---	306	---
[226][228]	---	4400	320	---
[254]	---	---	367	---
[101][167]	---	4316	369	376
[47]	---	3850	390	390
[30]	---	5494	402	402

Algorytm MARS jest trudny do implementacji na tanich kartach inteligentnych [149]. Wymaga 2kB pamięci ROM dla realizacji SBoxa. Problemem jest także realizacja przeszukiwania wygenerowanych kluczy, która jest podatna na ataki czasowe [218]. Równie podatna jest programowa realizacja zmiennych przesunięć i mnożenia 32-bitowego na procesorach 8-bitowych [7].

Tabela 10. Realizacje algorytmu MARS na procesorach 8-bitowych

Źródło	Procesor	Pamięć programu [bajty]	Pamięć danych [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania / deszyfrowania
[47][1]	---	---	---	15776	5008
[140]	6805	4136	225	110066	34163
[218]	Z80 + Koprocesor	5468	572	21742	45558
[140]	6805	4077	188	212735	358240

8.3.2. Sprzętowa realizacja algorytmu

Algorytm MARS jest stosunkowo wolny w sprzęcie i zajmuje dużą ilość zasobów sprzętowych z uwagi na stosowanie 3 różnych rodzajów rund, operacji 32-bitowego mnożenia i zmiennych przesunięć [95]. Największy czas wykonania ma operacja E-Function, a w szczególności operacja mnożenia [125][270]. Zastosowanie mnożnika z 32-bitowym wynikiem skraca ścieżkę krytyczną [219]. Duże SBoxy zwiększają zużycie zasobów sprzętowych, przez co minimalne zużycie zasobów jest drugie największe (po Serpent). Ponieważ algorytm ma strukturę sieci Feistela, moduł szyfrowania jest niemal identyczny z modułem deszyfrowania (wymaga dodatkowo 3% zasobów [96]). Operacja generacji podkluczy rundy wymaga dopasowania wzorców (ang. *string matching)*, przez co jest bardzo nieefektywna do implementacji sprzętowej [68] i zajmuje ponad 70% zasobów sprzętowych [270]. Algorytm stosuje sekwencyjne wyznaczanie podkluczy rundy. Długi czas generacji kluczy

[1] W pracy [47] podano jedynie estymaty, nie wykonano fizycznej implementacji

uniemożliwia użycie algorytmu do zastosowań, w których obsługiwana jest duża liczba różnych kluczy (np. IPSec) [229].

Tabela 11. Realizacje algorytmu MARS w układach ASIC i FPGA

Źródło	Układ	Architektura (Liczba cykli)	Całkowite zużycie zasobów	Czas generacji klucza [ns]	Przepustowość szyfrowania [Mbps]
[270]	ASIC 0.5 µm	LU 1 (34)	127 mm^2	27429	57
[47][1]	ASIC @ 250 MHz	LU 1 (50)	70000 komórek	---	640
[219]	ASIC 0.18 µm	LU 1 (34)	66800 bramek	4723	677
[125]	ASIC 0.35 µm	LU 32 (1)	2935754 bramek	1741	226
[219]	ASIC 0.18 µm	Pipelined 2 (18)	66800 bramek	4723	1280
[47][1]	ASIC @ 250 MHz	Pipelined 8 (4)	393000 komórek	---	8000
[270]	ASIC 0.5 µm	Pipelined 34 (1)	1332 mm^2	10199	2189
[96]	Xilinx Virtex	LU 1 (32)	2737 CLBs	Brak generacji	40
[97]	Xilinx Virtex	LU 1 (32)	2744 CLBs	Brak generacji	61
[68]	Xilinx Virtex	LU 1 (32)	6896 CLBs	1960	102

8.4. Algorytm RC6

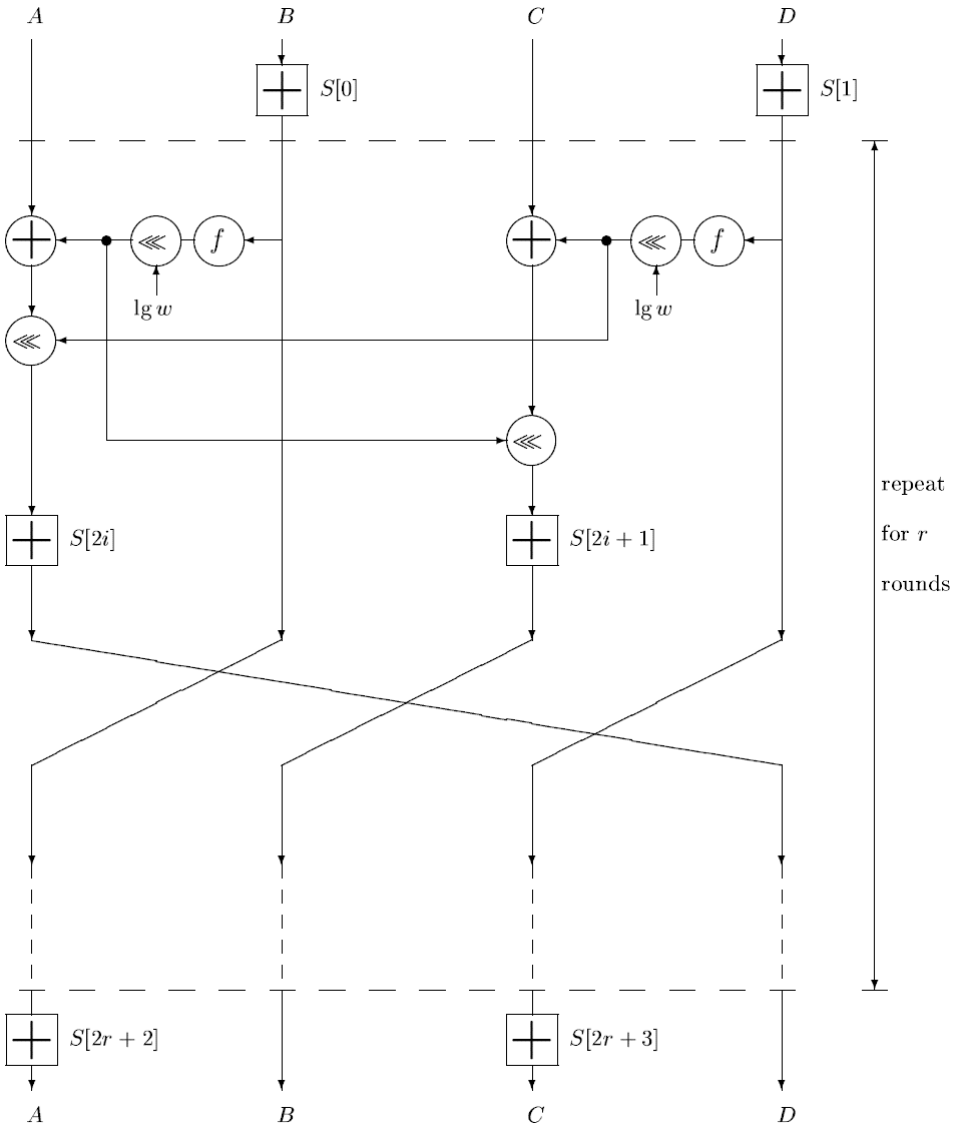
Algorytm RC6 [211] został zaprojektowany przez zespół pod przewodnictwem Ronalda Rivesta jako rozwinięcie algorytmu RC5 dopasowanego do wymagań AES. RC6 jest siecią Feistela i operuje na 4 słowach 32-bitowych, wykonuje 20 rund i używa 44 kluczy rundy (patrz Rysunek 22). Generacja kluczy rundy jest identyczna z RC5. Algorytm używa operacji 32-bitowych, które są wydajnie realizowane we współczesnych procesorach [211]:

- dodawanie i odejmowanie arytmetyczne modulo 2^{32},
- mnożenie arytmetyczne modulo 2^{32},
- logiczna suma wyłączająca (XOR),
- przesunięcie cykliczne w lewo i prawo zależne od danych.

Algorytm RC6 posiada bardzo mały margines bezpieczeństwa równy 1,18 (drugi najmniejszy po Rijndael) [229], głównie z uwagi na zbyt słabą funkcję rundy z pojedynczym punktem awarii (ang. *single point of failure*), opartą o 32-bitowe mnożenie i rotacje zależne od danych [48][63].

8.4.1. Programowa realizacja algorytmu

Algorytm RC6 ma najbardziej zwartą specyfikację [149], co owocuje najkrótszą [63] i najbardziej wydajną implementacja w języku C [226]. Analiza operacji składowych algorytmu RC6 wykazuje, że jest on szczególnie predestynowany do implementacji programowych na platformach 32-bitowych, a w szczególności na procesorze Pentium [8]. Wydajność szyfrowania i generacji kluczy rundy jest niezależna od długości klucza [228]. Efektywna równoległość algorytmu na poziomie instrukcji wynosi 1,81 [61] (181 instrukcji spośród 328 jest na ścieżce krytycznej). Wykonanie algorytmu na maszynie typu „dataflow” pozwala na przyspieszenie algorytmu RC6 o 20% [29][46][275], co wskazuje na niewielki stopień równoległości operacji z powodu znacznego użycia mnożników [8].



Rysunek 22. Schemat algorytmu RC6 [211]

Tabela 12. Realizacje algorytmu RC6 na procesorze Pentium

Źródło	Rozmiar programu [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania	Liczba cykli deszyfrowania
[8]	---	---	223	209
[167]	---	---	243	---
[226][228]	---	1700	250	---
[211]	---	1108	254	254
[254]	---	---	263	---
[101]	---	1632	270	226
[30]	---	2198	604	570

Algorytm RC6 jest łatwo realizowany na kartach inteligentnych dając stosunkowo zwarty kod (nie używa SBoxów), lecz osiąga stosunkowo słabą wydajność [113]. W standardowej implementacji wymaga co najmniej 176 bajtów pamięci danych, choć istnieje możliwość wykorzystania mniejszej ilości pamięci kosztem utraty wydajności [140]. Realizacja 32-bitowego mnożenia i zmiennych przesunięć na procesorze 8-bitowym może być podatna na ataki czasowe [7]. Stały czas wykonania mnożenia i przesunięć jest zazwyczaj dłuższy niż w przypadku zmiennego czasu wykonania [140]. Głównym problemem implementacji jest długi czas generacji kluczy rundy.

Tabela 13. Realizacje algorytmu RC6 na procesorach 8-bitowych

Źródło	Procesor	Pamięć programu [bajty]	Pamięć danych [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania / deszyfrowania
[211]	8051	---	176	27456	13535
[113]	8051	596	221	43200	14400
[140]	6805	933	226	82167	32731
[218]	Z80 + Koprocesor	1060	156	138851	34736
[137]	8051	706	77	---	58056
[140]	6805	1374	149	78808	105981

8.4.2. Sprzętowa realizacja algorytmu

Algorytm RC6 jest stosunkowo wolny w sprzęcie z uwagi na stosowanie operacji 32-bitowego mnożenia (podnoszenia do kwadratu) i zmiennych przesunięć, a uzyskana przepustowość zależy głównie od efektywności ich implementacji [83]. Ilość zużytych zasobów również zależy od efektywności implementacji wymienionych operacji [95]. Algorytm nie używa SBoxów, co powoduje drugie minimalne zużycie zasobów (po algorytmie Twofish [96]). Ponieważ algorytm ma strukturę sieci Feistela, moduł szyfrowania jest niemal identyczny z modułem deszyfrowania (wymaga dodatkowo 20% zasobów [96]). Algorytm nie umożliwia wyznaczania podkluczy rundy na bieżąco, a podklucze rundy są generowane sekwencyjnie. Długi czas generacji kluczy uniemożliwia użycie algorytmu do zastosowań, w których obsługiwana jest duża liczba różnych kluczy (np. IPSec) [229].

Tabela 14. Realizacje algorytmu RC6 w układach ASIC i FPGA

Źródło	Układ	Architektura (liczba cykli)	Całkowite zużycie zasobów	Czas generacji klucza [ns]	Przepustowość szyfrowania [Mbps]
[211][2]	ASIC 0.25 µm @ 200 MHz	LU 1 (20)	0.05 mm^2	Brak generacji	1280
[270]	ASIC 0.5 µm	LU 1 (20)	19 mm^2	13776	103
[125]	ASIC 0.35 µm	LU 20 (1)	1643037 bramek	2112	204
[270]	ASIC 0.5 µm	Pipelined 20 (1)	453 mm^2	6898	2197
[210]	Xilinx 4000	LU 1 (20)	6450 CLBs	Brak generacji	37
[96]	Xilinx 4000	LU 1 (20)	1222 CLBs	Brak generacji	46
[82]	Xilinx Virtex	LU 1 (20)	2638 CLBs	Brak generacji	88
[96]	Xilinx Virtex	LU 1 (20)	1139 CLBs	Brak generacji	104
[68]	Xilinx Virtex	LU 1 (20)	2650 CLBs	170	113
[60]	Xilinx Virtex	LU 1 (20)	1137 CLBs	Brak generacji	143
[36]	Xilinx Virtex	LU 1 (20)	1709 CLBs	Brak generacji	160
[36]	Xilinx Virtex II	LU 1 (20)	1560 CLBs	Brak generacji	290
[36]	Xilinx Virtex	LU 4 (5)	3932 CLBs	Brak generacji	160
[36]	Xilinx Virtex II	LU 4 (5)	2902 CLBs	Brak generacji	340
[82]	Xilinx Virtex	LU 10 (2)	6406 CLBs	Brak generacji	97
[82]	Xilinx Virtex	Pipelined 10/1 (2)	8999 CLBs	Brak generacji	1705
[82]	Xilinx Virtex	Pipelined 10/2 (2)	10856 CLBs	Brak generacji	2398
[36]	Xilinx Virtex	Pipelined 20 (1)	19198 CLBs	Brak generacji	10600
[60]	Xilinx Virtex	Pipelined 20/28 (1)	46800 CLBs	Brak generacji	13056
[36]	Xilinx Virtex II	Pipelined 20 (1)	8554 CLBs	Brak generacji	15200

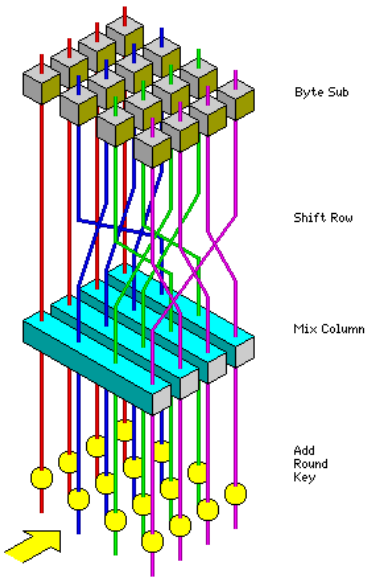
[2] W pracy [211] dokonano jedynie estymaty, nie wykonano fizycznej implementacji

8.5. Algorytm Rijndael

Algorytm Rijndael [64] został zaprojektowany przez Joana Daemena i Vincenta Rijmena jako rozwinięcie algorytmu Square dopasowanego do wymagań AES. Algorytm jest siecią permutacyjno-podstawieniową (patrz Rysunek 23) i używa 128, 192, 256-bitowych bloków danych i kluczy. W zależności od rozmiaru danych wejściowych i rozmiaru klucza wykonuje się 10, 12 lub 14 rund. Algorytm używa następujących operacji:

- logiczna suma wyłączająca (XOR) – dodawanie klucza,
- mnożenie przez stałą w ciele GF(2^8) – w liniowej warstwie mieszającej,
- tablica lookup (SBox) 8x8-bitowa - w nieliniowej warstwie podstawieniowej,
- stałe przesunięcia – w liniowej warstwie mieszającej.

Algorytm Rijndael jest najlepiej zaprojektowanym algorytmem zgłoszonym do konkursu AES [149]. Filozofia projektu opiera się o prostotę, symetrię, niezależność komponentów, wysoką równoległość i rozszerzalność [65]. Margines bezpieczeństwa jest wysoko oceniany przez autorów [66], a jednocześnie krytykowany w innych pracach i szacowany na 1,11 (najmniejszy spośród 5 algorytmów) [229]. Nieufnie jest przyjmowany zastosowany styl sieci permutacyjno-podstawieniowej, który nie został jeszcze wystarczająco przeanalizowany [48]. Dla kluczy 256-bitowych wydajność algorytmu spada, z uwagi na wykonywanie 14 rund [63].



Rysunek 23. Schemat rundy algorytmu Rijndael [220]

8.5.1. Programowa realizacja algorytmu

Algorytm Rijndael jest stosunkowo prostym algorytmem do realizacji programowej i w zestawieniach porównawczych 5 algorytmów Rijndael jest trzecim najszybszym (po RC6 i Twofish) [167]. Z uwagi na równoległość przetwarzania i proste instrukcje *load* i *xor* Rijdnael efektywnie korzysta z instrukcji typu MMX [8] oraz uzyskuje przyspieszenie na procesorach VLIW. Algorytm jest efektywnie realizowany na większości platform z uwagi na brak operacji arytmetycznych, rotacji zależnych od danych oraz prosty algorytm generacji kluczy rundy [65]. Wydajność szyfrowania i generacji kluczy rundy maleje ze wzrostem długości klucza [228]. Efektywna równoległość algorytmu na poziomie instrukcji wynosi 7,44 [61] (71 instrukcji spośród 528 jest na ścieżce krytycznej), choć w praktyce jest ograniczona przez operacje

dostępu do tablic [254]. Wykonanie algorytmu na maszynie typu „dataflow” pozwala na przyspieszenie algorytmu Rijndael w największym stopniu spośród testowanych algorytmów [29][46][275].

Tabela 15. Realizacje algorytmu Rijndael na procesorze Pentium

Źródło	Rozmiar programu [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania	Liczba cykli deszyfrowania
[67][167]	---	---	229	---
[8][167]	---	---	237	---
[67][101]	---	---	280	---
[167]	---	---	282	---
[226][228]	---	300/1400	291	---
[64]	---	---	320	---
[254]	---	---	362	---
[101]	---	305/1389	374	352
[30]	---	7307	694	712

Algorytm Rijndael realizowany na kartach inteligentnych z 8-bitowym procesorem wymaga najmniejszej ilości pamięci danych (poniżej 64 bajtów) i stosunkowo małej ilości pamięci programu (poniżej 1024 bajtów) [140]. W celu zapobiegania atakom w kanale bocznym, operację *xtime* należy zrealizować w stałym czasie wykonania [64][140]. Rijndael jest najszybszy zarówno na kartach inteligentnych o małej i dużej wydajności [140][218].

Tabela 16. Realizacje algorytmu Rijndael na procesorach 8-bitowych

Źródło	Procesor	Pamięć programu [bajty]	Pamięć danych [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania / deszyfrowania
[64][113]	8051	1016	49	---	3168
[64]	8051	826	49	---	3744
[137]	8051	826	54	---	3977
[64][113]	8051	768	49	---	4065
[64]	6808	919	36	---	8390
[140]	6805	879/1049	50/54	0/2278	9464/13538
[140]	6805	553	53	0	14945
[218]	Z80	980	66	10318	25494

8.5.2. Sprzętowa realizacja algorytmu

Rijndael jest idealny do implementacji w FPGA z uwagi na zastosowanie prostych operacji [226]. Łączy drobnoziarnistą równoległość z operacjami na tablicach *lookup* [68][269]. Największy czas wykonania ma operacja *InvMixColumn* [270]. Operacją dominującą są SBoxy 8x8-bitowe stanowiące większość zużytych zasobów [96][125], utrudniając kompaktową implementację. Studium implementacji SBoxów zawarto w [257]. Algorytm osiąga najkrótszy czas wyznaczania podkluczy rundy, najwyższą przepustowość i najlepsze wykorzystanie zasobów sprzętowych [68]. Z uwagi na permutacyjno-podstawieniową strukturę szyfru moduł szyfrowania jest odmienny od modułu deszyfrowania (wymaga dodatkowo 55% zasobów [96]). Algorytm umożliwia wyznaczanie podkluczy rundy na bieżąco i współbieżnie do szyfrowania. Rijndael jest najszybszy w zestawieniach porównawczych implementacji sprzętowych 5 algorytmów [229].

Tabela 17. Realizacje algorytmu Rijdnael w układach ASIC i FPGA

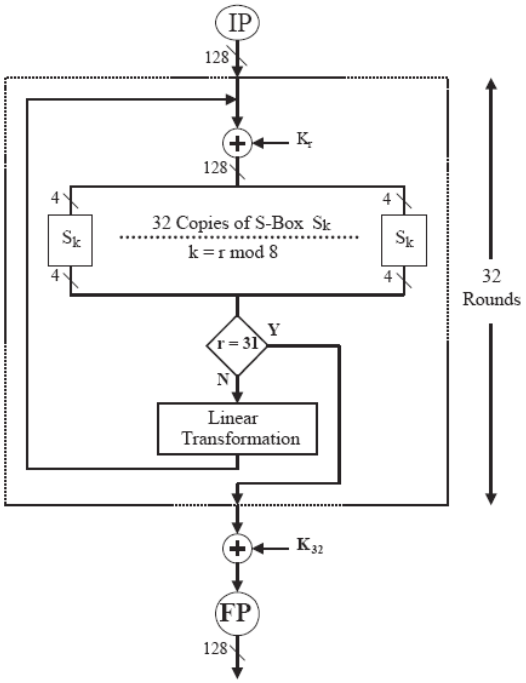
Źródło	Układ	Architektura (liczba cykli)	Całkowite użycie zasobów	Czas generacji klucza [ns]	Przepustowość szyfrowania [Mbps]
[270]	ASIC 0.5 µm	LU 1 (10)	34 mm^2	0/472	606
[156]	ASIC 0.18 µm	LU 1 (10)	173000 bramek	0	1600
[125]	ASIC 0.35 µm	LU 10 (1)	612834 bramek	57	1950
[171]	ASIC 0.6 µm	Pipelined 2 (5)	300000 tranzystorów	34/260	2260
[270]	ASIC 0.5 µm	Pipelined 10 (1)	420 mm^2	0/333	5745
[95]	Altera Flex 10K	LU 1 (10)	1585 LEs	Brak generacji	233
[188]	Altera Flex 10K	LU 1 (10)	1032 LEs	0	268
[82]	Xilinx Virtex	LU 1 (11)	3528 CLBs	Brak generacji	294
[96]	Xilinx Virtex	LU 1 (10)	2902 CLBs	Brak generacji	332
[68]	Xilinx Virtex	LU 1 (10)	5673 CLBs	70	353
[60]	Xilinx Virtex	LU 1 (10)	2507 CLBs	Brak generacji	414
[244]	Xilinx Virtex II	LU 1 (10)	1830 CLBs	0	1229
[82]	Xilinx Virtex	LU 2 (6)	5302 CLBs	Brak generacji	300
[244]	Xilinx Virtex II	LU 2 (5)	3326 CLBs	0	1486
[82]	Xilinx Virtex	LU 5 (3)	10286 CLBs	Brak generacji	237
[244]	Xilinx Virtex II	LU 5 (2)	7823 CLBs	0	1193
[244]	Xilinx Virtex II	LU 10 (1)	15021 CLBs	0	793
[242]	Xilinx Virtex II	LU 10 (1)	15215 CLBs	0	847
[142]	Xilinx Virtex	Pipelined 1/1 (10)	2068 CLBs	Brak generacji	740
[142]	Xilinx Virtex II	Pipelined 1/1 (10)	2128 CLBs	Brak generacji	1104
[82]	Xilinx Virtex	Pipelined 5/1 (2)	10992 CLBs	Brak generacji	1938
[142]	Xilinx Virtex	Pipelined 5/1 (2)	8113 CLBs	Brak generacji	3174
[142]	Xilinx Virtex II	Pipelined 5/1 (2)	8114 CLBs	Brak generacji	4735
[181]	Xilinx Virtex	Pipelined 10 (1)	2679 CLBs	0	6596
[60]	Xilinx Virtex	Pipelined 10/7 (1)	12600 CLBs + 80 RAMs	Brak generacji	12160

8.6. Algorytm Serpent

Algorytm Serpent [6] został zaprojektowany przez Rossa Andersona, Eli Bihama i Larsa Knudsena. Algorytm jest siecią permutacyjno-podstawieniową (patrz Rysunek 24) przetwarzającą 128-bitowe bloki danych (4 słowa 32-bitowe) w 32 rundach z kluczem 256-bitowym (klucze krótsze są uzupełniane do tego rozmiaru). Algorytm stosuje następujące operacje:

- logiczna suma wyłączająca (XOR) – dodawanie klucza i transformacje liniowe,
- stałe przesunięcia i rotacje – transformacje liniowe,
- 8 tablic lookup (SBox) 4x4-bitowych – w nieliniowej warstwie podstawieniowej.

Serpent uznawany jest za najbezpieczniejszy algorytm ze współczynnikiem bezpieczeństwa 3,56 [229] z uwagi na zastosowanie 32 rund szyfrowania [148][229] i silnej funkcji mieszającej [63]. Autorzy nie zdecydowali się na zastosowanie nowych i nieprzeanalizowanych idei [6]. Jednocześnie ekstremalny konserwatyzm projektu (32 rundy szyfrowania) powoduje spadek wydajności realizacji programowych i sprzętowych w wersjach nie-potokowych.



Rysunek 24. Schemat blokowy algorytmu Serpent [82]

8.6.1. Programowa realizacja algorytmu

Serpent jest prostym algorytmem, który może zostać wydajnie implementowany w trybie bitowo-równoległym (ang. *bitslice*) lub słowowo-równoległym (ang. *parallel*) na procesorach 32-bitowych. Największą przepustowość osiąga się stosując transformację liniową w trybie bitowo-równoległym [6] i SBoxy przedstawione w postaci funkcji boolowskich [199][279]. Mimo optymalizacji Serpent jest najwolniejszy spośród 5 testowanych algorytmów (czterokrotnie wolniejszy od RC6) [8][167], głównie z powodu zastosowania 32 rund. Wydajność szyfrowania i generacji kluczy rundy jest niezależna od długości klucza [228]. Efektywna równoległość algorytmu na poziomie instrukcji wynosi 2,54 [61] (526 instrukcji spośród 1336 jest na ścieżce krytycznej). Większą przepustowość uzyskuje się na procesorach RISC [226].

Tabela 18. Realizacje algorytmu Serpent na procesorze Pentium

Źródło	Rozmiar programu [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania	Liczba cykli deszyfrowania
[199]	---	1106	759	770
[226][228]	---	2500	900	1100
[101][199]	---	1290	945	951
[101]	---	2402	952	914
[254]	---	---	985	---
[167]	---	---	992	---
[6][199]	---	---	1170	1301
[30]	---	7000	1437	1276
[6]	---	---	1738	---

Algorytm jest łatwy do realizacji na kartach inteligentnych [7]. Może być zaimplementowany w systemach z 64 bajtami pamięci RAM oraz 1K lub 2K bajtami pamięci ROM w trybach bitowo-równoległym i słowowo-równoległym. Analiza ścieżki krytycznej [140] wykazała, że 40% czasu jest wykonywana transformacja afiniczna podczas generacji kluczy, a 30% czasu są obliczane SBoxy (każdy zajmuje 560 cykli). Algorytm jest najwolniejszy spośród testowanych algorytmów [137][140][218].

Tabela 19. Realizacje algorytmu Serpent na procesorach 8-bitowych

Źródło	Procesor	Pamięć programu [bajty]	Pamięć danych [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania / deszyfrowania
[7][3]	6805	2048	80	11000	11000
[7][3]	6805	1024	80	34000	34000
[137]	8051	2021	56	0	70339
[218]	Z80	3937	164	147972	71924
[140]	6805	1056	101	0	126074

8.6.2. Sprzętowa realizacja algorytmu

Algorytm jest bardzo prosty do implementacji w układach FPGA. Z uwagi na małą złożoność wszystkie operacje wpasowują się w strukturę CLB. Zastosowane SBoxy 4x4-bitowe zajmują 4 tablice LUT, jednakże użycie 8 różnych SBoxów powoduje zwiększenie zużycia zasobów. W zestawieniach porównawczych Serpent uzyskuje najkrótszy czas szyfrowania pojedynczej rundy, najwyższe przepustowości w implementacjach potokowych, drugą najlepszą przepustowość w implementacjach iteracyjnych [229], a także drugie najlepsze wykorzystanie zasobów sprzętowych [68]. Z uwagi na permutacyjno-podstawieniową strukturę szyfru moduł szyfrowania jest odmienny od modułu deszyfrowania (wymaga dodatkowo 100% zasobów [96]). Algorytm umożliwia wyznaczanie podkluczy rundy na bieżąco i współbieżnie do szyfrowania. Dzięki zastosowaniu interfejsu JBits do modyfikacji konfiguracji FPGA w zależności od klucza, uzyskuje się przepustowości bliskie 18 Gbps [204].

Tabela 20. Realizacje algorytmu Serpent w układach ASIC i FPGA

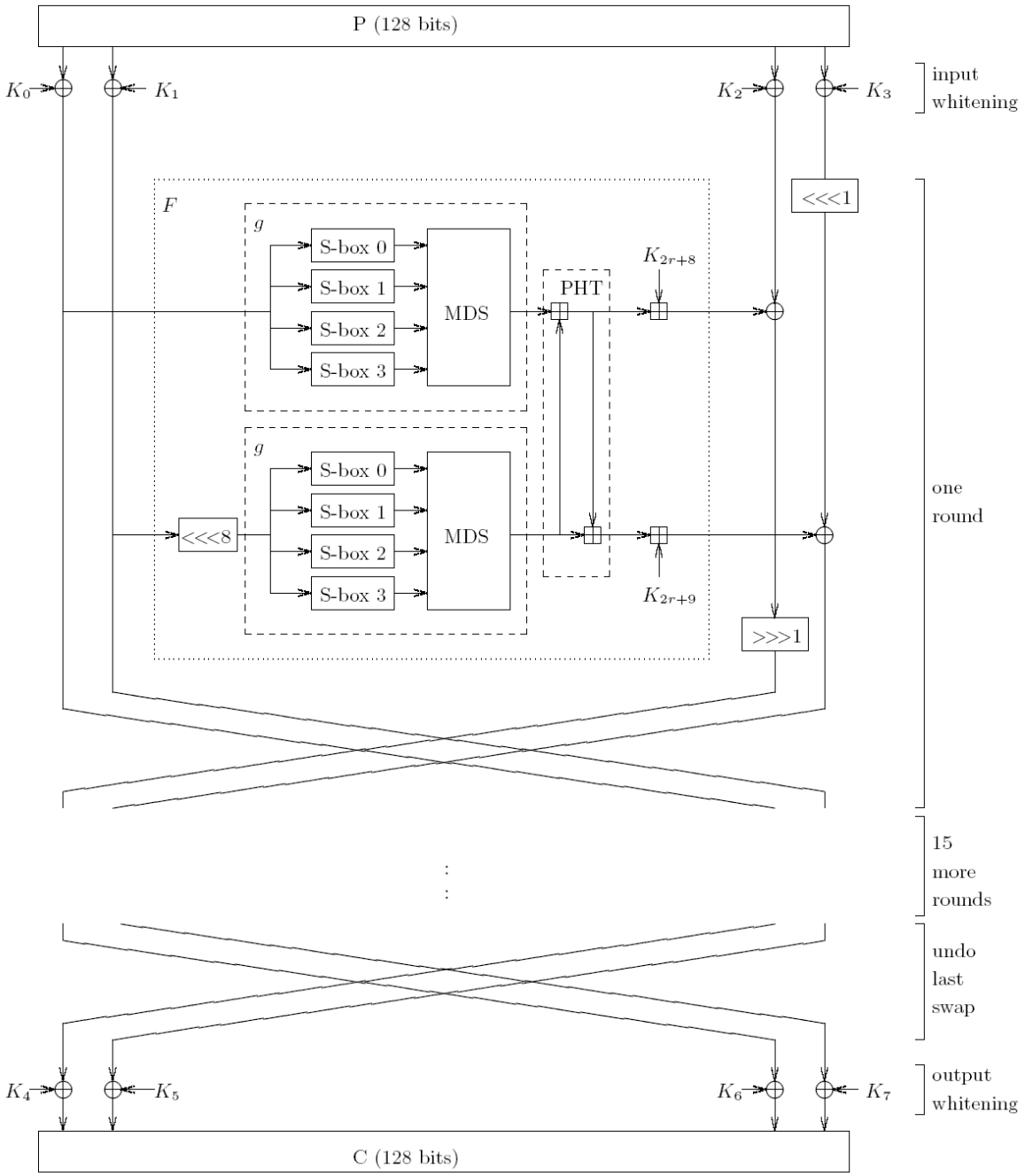
Źródło	Układ	Architektura (liczba cykli)	Całkowite zużycie zasobów	Czas generacji klucza [ns]	Przepustowość szyfrowania [Mbps]
[270]	ASIC 0.5 µm	LU 1 (32)	23 mm^2	20 / 672	202
[125]	ASIC 0.35 µm	LU 32 (1)	503770 bramek	114	932
[171]	ASIC 0.6 µm	Pipelined 4 (8)	300000 tranzystorów	171	1960
[270]	ASIC 0.5 µm	Pipelined 32 (1)	439 mm^2	19 / 213	8030
[95]	Altera Flex 10K	LU 1 (32)	3678 LEs	Brak generacji	126
[41]	Altera Flex 10K	LU 1 (32)	1976 LCs	Brak generacji	117
[41]	Altera Flex 10K	LU 8 (4)	2743 LCs	Brak generacji	301
[81]	Xilinx Virtex	LU 1 (32)	5511 CLBs	Brak generacji	62
[68]	Xilinx Virtex	LU 1 (32)	2550 CLBs	80	149
[244]	Xilinx Virtex II	LU 1 (32)	2057 CLBs	0	423
[244]	Xilinx Virtex II	LU 2 (16)	2361 CLBs	0	554
[244]	Xilinx Virtex II	LU 4 (8)	2998 CLBs	0	691
[96]	Xilinx Virtex	LU 8 (4)	4438 CLBs	0	339
[60]	Xilinx Virtex	LU 8 (4)	4507 CLBs	Brak generacji	431
[81]	Xilinx Virtex	LU 8 (4)	7964 CLBs	Brak generacji	444
[244]	Xilinx Virtex II	LU 8 (4)	4192 CLBs	0	827
[244]	Xilinx Virtex II	LU 16 (2)	7610 CLBs	0	816
[81]	Xilinx Virtex	LU 32 (1)	8103 CLBs	Brak generacji	312
[242]	Xilinx Virtex II	LU 32 (1)	13076 CLBs	0	322
[244]	Xilinx Virtex II	LU 32 (1)	13788 CLBs	0	954
[81]	Xilinx Virtex	Pipelined 32 (1)	9004 CLBs	Brak generacji	4860
[204]	Xilinx Virtex	Pipelined 32 (1)	4032 CLBs	Rekonfiguracja	17550
[60]	Xilinx Virtex	Pipelined 32/24	19700 CLBs	Brak generacji	16768

[3] W pracy [7] dokonano jedynie estymaty, nie wykonano fizycznej implementacji

8.7. Algorytm Twofish

Algorytm Twofish [224] został zaprojektowany przez zespół Bruce Schneiera jako rozwinięcie algorytmu Blowfish dopasowanego do wymagań AES. Twofish jest zmodyfikowaną siecią Feistela (patrz Rysunek 25) i przetwarza 128-bitowy blok danych w 16 rundach, przy użyciu klucza szyfrowania 128, 192 lub 256-bitowego, z którego generowanych jest 40 32-bitowych kluczy rundy. Algorytm używa następujących operacji:

- tablica lookup (SBox) 8x8-bitowa zależna od klucza,
- macierzy MDS 4x4-elementowej nad ciałem GF(2^8) – transformacja liniowa,
- przekształcenia PHT.



Rysunek 25. Schemat algorytmu Twofish [224]

Algorytm Twofish jest złożonym algorytmem, zapożyczającym przekształcenia od Square, Rijndael (macierz MDS), Khufu (SBoxy zależne od klucza) oraz Safer

(przekształcenie PHT) [149]. Twofish ma bardzo duży margines bezpieczeństwa równy 2,67 (drugi największy po Serpent) [148][229] i najsilniejszą funkcję rundy dzięki zastosowaniu SBoxów zależnych od klucza [229]. W pracy [48] wykazano, że trudno oszacować rzeczywisty stopień bezpieczeństwa, z uwagi na trudność kryptoanalizy algorytmu Twofish. Twofish jest obecnie stosowany w szerokiej gamie produktów kryptograficznych (np. Cryptix, OpenJCE i SSH Secure Shell).

8.7.1. Programowa realizacja algorytmu

Algorytm Twofish został zaprojektowany, aby uzyskać wysoką wydajność na szerokiej gamie platform programowych i sprzętowych [229]. Mimo, że algorytm nie jest najłatwiejszy do implementacji lub optymalizacji, jego modularna budowa umożliwia implementację na wiele sposobów. Twofish może być zoptymalizowany ze względu na szyfrowanie masowe, elastyczność klucza, małe zużycie zasobów sprzętowych, przepustowość, krótki czas zwłoki lub dowolną ich kombinację [229]. Dla programowych realizacji algorytmu istnieje ścisły kompromis pomiędzy czasem inicjalizacji klucza a wydajnością szyfrowania bloków [224]. W zestawieniach porównawczych Twofish jest drugim najszybszym algorytmem (po RC6) [167]. Twofish nie korzysta z instrukcji typu MMX [8], lecz używa głównie instrukcji RISC [226]. Wydajność szyfrowania jest niezależna od długości klucza, a wydajność generacji kluczy rundy maleje ze wzrostem długości klucza [226]. Efektywna równoległość algorytmu na poziomie instrukcji wynosi 3,26 [61] (162 instrukcji spośród 528 jest na ścieżce krytycznej). Wykonanie algorytmu na maszynie typu „dataflow” pozwala na przyspieszenie algorytmu Twofish o 76% [29][46][275], co wskazuje na możliwości zrównoleglenia operacji (np. przy użyciu procesorów VLIW), choć mniejsze niż dla Rijndael, z uwagi na zastosowanie transformacji PHT [8].

Tabela 21. Realizacje algorytmu Twofish na procesorze Pentium

Źródło	Rozmiar programu [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania	Liczba cykli deszyfrowania
[227]	9000	8600	258	258
[8]	---	---	282	276
[224]	8900	12700	285	285
[227]	8500	7600	315	315
[254]	---	---	371	---
[101]	---	8414	376	374
[224]	10700	4900	460	460
[224]	13600	2400	720	720
[224]	9100	1250	860	860
[30]	---	13000	937	876

Implementacje algorytmu Twofish na procesorach 8-bitowych wyznaczają klucze rundy „w locie”, nie wymagając dodatkowej pamięci RAM na ich przechowywanie. Przy 160 bajtach dostępnej pamięci danych na przechowanie kluczy rundy wydajność wzrasta dwukrotnie. Tablice q1 i q2 wymagają 512 bajtów pamięci ROM. Macierz MDS jest obliczana bezpośrednio [225]. Szeroka gama parametrów, które przedstawia Tabela 22, ilustruje elastyczność realizacji algorytmu Twofish w środowiskach o ograniczonych zasobach. Brak drugiego rejestru indeksowego w 6805 ma znaczący wpływ na rozmiar kodu i czas jego wykonania, stąd też inny procesor 8-bitowy z wieloma rejestrami indeksowymi (np. 6502) może osiągnąć lepsze wyniki [224][271].

Tabela 22. Realizacje algorytmu Twofish na procesorach 8-bitowych

Źródło	Procesor	Pamięć programu [bajty]	Pamięć danych [bajty]	Liczba cykli generacji klucza	Liczba cykli szyfrowania
[271]	6805	1000	3292	1750	11900
[271]	6805	1300	1244	1750	12700
[271]	6805	1900	220	1750	15300
[113]	8051	1903	48	0	18126
[137]	8051	2063	52	---	19819
[113]	8051	1443	68	0	24422
[140][225]	6805	2200	60	1750	26500
[218]	Z80	2808	90	28512	31877
[224]	6805	2150	60	1750	32900
[225]	6805	2000	60	1750	35000
[225]	6805	1760	60	1750	37100

8.7.2. Sprzętowa realizacja algorytmu

Twofish ma stosunkowo skomplikowaną strukturę, używa jednak operacji, które są stosunkowo łatwo realizowane w układach ASIC i FPGA. Struktura przekształcenia rundy jest identyczna z modułem generacji kluczy, co upraszcza współdzielenie zasobów [270]. Zgodnie z intencjami autorów algorytm umożliwia kompaktową implementację [269][270], a w zestawieniach porównawczych Twofish zużywa najmniejszą liczbę zasobów [96][270]. Największy czas wykonania mają SBoxy [125][270]. Ponieważ algorytm ma strukturę sieci Feistela, moduł szyfrowania jest niemal identyczny z modułem deszyfrowania (wymaga dodatkowo 6% zasobów [96]). Podklucze rundy są wyznaczane w dowolnej kolejności i mogą być wyznaczane na bieżąco współbieżnie do szyfrowania i odszyfrowania, co stanowi idealny przypadek dla implementacji sprzętowych [269][270].

Tabela 23. Realizacje algorytmu Twofish w układach ASIC i FPGA

Źródło	Układ	Architektura (liczba cykli)	Całkowite zużycie zasobów	Czas generacji klucza [ns]	Przepustowość szyfrowania [Mbps]
[224][4]	ASIC 0.35 µm	LU 1 (64)	14000 bramek	100	80
[270]	ASIC 0.5 µm	LU 1 (16)	16 mm^2	105	105
[224][4]	ASIC 0.35 µm	LU 1 (32)	19000 bramek	1000	160
[163]	ASIC 0.35 µm	LU 1 (42)	35000 bramek	0	200
[224][4]	ASIC 0.35 µm	LU 1 (16)	23000 bramek	500	320
[224][4]	ASIC 0.35 µm	LU 1 (16)	80000 bramek	3750	640
[203]	ASIC 0.13 µm	LU 1 (16)	0.109 mm^2	0	3112
[203]	ASIC 0.09 µm	LU 1 (16)	0.054 mm^2	0	3584
[125]	ASIC 0.35 µm	LU 32	431857 bramek	16	394
[224][4]	ASIC 0.35 µm	Pipelined 1/1 (16)	26000 bramek	250	640
[224][4]	ASIC 0.35 µm	Pipelined 1/2 (16)	28000 bramek	166	960
[224][4]	ASIC 0.35 µm	Pipelined 1/3 (16)	30000 bramek	133	1200
[270]	ASIC 0.5 µm	Pipelined 16 (1)	225 mm^2	92	2273
[71]	Altera Flex 10K	LU 1 (72)	57400 bramek	0	7.8
[95]	Altera Flex 10K	LU 1 (16)	1950 LEs	Brak generacji	82
[59]	Xilinx 4000	LU 1 (16)	911 CLBs	Brak generacji	86
[96]	Xilinx 4000	LU 1 (16)	907 CLBs	Brak generacji	91
[82]	Xilinx Virtex	LU 1 (16)	2666 CLBs	Brak generacji	104
[68]	Xilinx Virtex	LU 1 (16)	9363 CLBs	180	173

[4] W pracy [224] dokonano jedynie estymaty, nie wykonano fizycznej implementacji

[96][203]	Xilinx Virtex	LU 1 (16)	1076 CLBs	Brak generacji	177
[203]	Xilinx Spartan 3	LU 1 (16)	1918 CLBs	0	180
[244]	Xilinx Virtex II	LU 1 (16)	3854 CLBs	0	256
[82]	Xilinx Virtex	LU 2 (8)	3392 CLBs	Brak generacji	114
[197][5]	Xilinx Virtex	LU 2 (8)	2464 CLBs	0	139
[244]	Xilinx Virtex II	LU 2 (8)	6919 CLBs	0	241
[244]	Xilinx Virtex II	LU 4 (4)	14270 CLBs	0	201
[242]	Xilinx Virtex II	LU 16 (1)	61240 CLBs	0	206
[82]	Xilinx Virtex	Pipelined 8/1 (2)	9345 CLBs	Brak generacji	1585
[82]	Xilinx Virtex	Pipelined 8/2 (2)	10008 CLBs	Brak generacji	2396
[60]	Xilinx Virtex	Pipelined 16/23 (1)	21000 CLBs	Brak generacji	15232

8.8. Implementacja algorytmów kryptograficznych w FPGA

Implementację algorytmów kryptograficznych w układach FPGA realizowano w ramach pracy dyplomowej [242] i rozwijano w pracach [243][244][245].

8.8.1. Analiza implementacji operacji składowych

Do realizacji w układach FPGA wybrano algorytmy Rijndael, Serpent i Twofish, które zostały zakwalifikowane w procesie wyboru AES. Umożliwiają współbieżną generację podkluczy rundy w stosunku do szyfrowania. Operacje składowe są stosunkowo łatwo realizowane w układach FPGA [242][243][244][245]:

- Permutacje, stałe przesunięcia i rotacje nie wymagają użycia żadnych zasobów logicznych. Powyższe przekształcenia zmieniają jedynie połączenia pomiędzy istniejącymi blokami logicznymi.
- Dodawanie modulo 2, transformacje liniowe oraz mnożenie przez stałą w ciele Galois $GF(2^8)$ jest realizowane za pomocą wielowejściowych bramek XOR. W układach FPGA bramki takie są odwzorowywane w tablice LUT. Dla liczby wejść mniejszej niż 4 stosuje się pojedynczą tablicę LUT, natomiast przy liczbie wejść od 5 do 16 stosuje się dwie warstwy tablic LUT.
- 32-bitowe dodawanie arytmetyczne modulo 2^{32} jest realizowane jako sumator kaskadowy. W układach FPGA istnieją dedykowane układy generacji i łańcuchy propagacji przeniesień, pozwalające na wydajne wykonanie operacji.
- Podstawienie SBox może zostać przedstawione jako funkcja logiczna lub jako asynchroniczna pamięć ROM. SBox 4-bitowe są odwzorowane w 4 tablice LUT, natomiast 8-bitowe wymagają 128 tablic LUT lub 256 bajtów ROM.

8.8.2. Przyjęte założenia i strategia projektowania

Głównym celem badań eksperymentalnych było porównanie przepustowości algorytmów w trybie szyfrowania ze sprzężeniem zwrotnym (CBC, CFB, OFB) [184][242][243][244][245] zrealizowanych w układach FPGA Xilinx Virtex II w architekturach iteracyjnych. Wyznaczono następujące parametry:

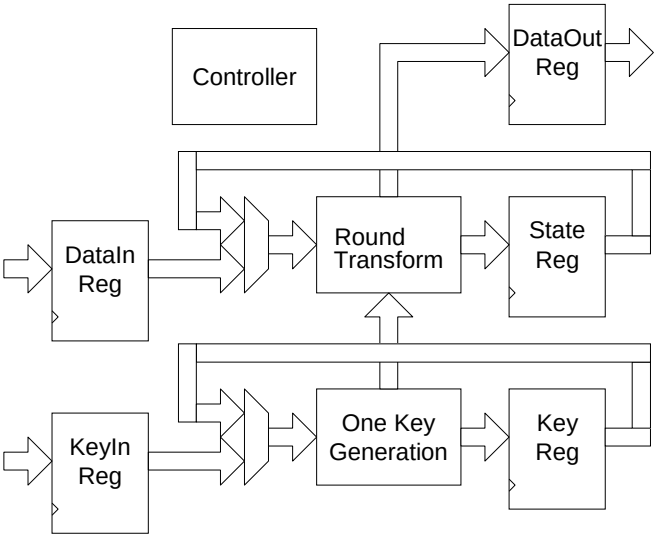
- przepustowość, czyli ilość danych przetwarzanych w jednostce czasu, wyrażaną w megabitach na sekundę (Mbps),
- zajętość zasobów sprzętowych, określającą koszt realizacji danego algorytmu w przyjętej architekturze, mierzoną w elementach CLB,

[5] W pracy [197] zaimplementowano Twofish w wersji 8-rundowej, wynik przepustowości jest przeskalowany dla wersji 16-rundowej

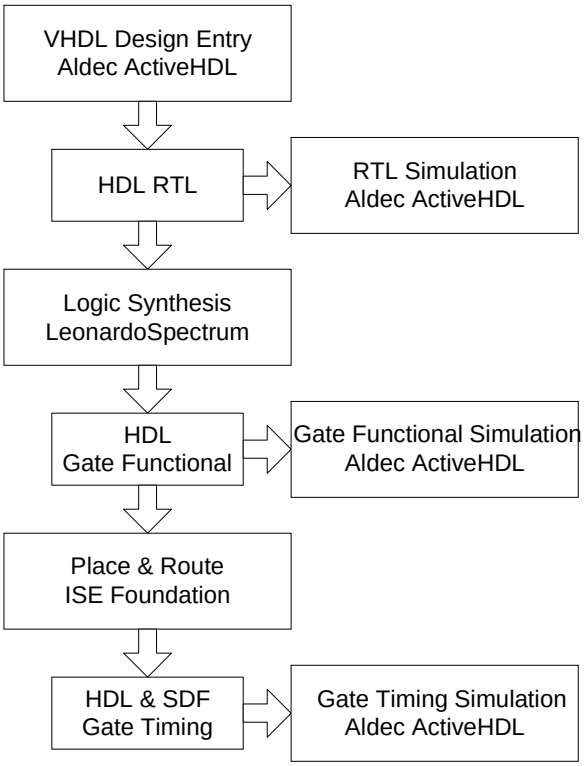
- skalowalność, charakteryzującą zależność przepustowości od zajętości zasobów sprzętowych.

Badania eksperymentalne przeprowadzono przy następujących założeniach:

- 128-bitowy synchroniczny interfejs wejściowo-wyjściowy,
- realizacja algorytmu szyfrowania i generacji podkluczy,
- współbieżne wyznaczanie podkluczy rundy w stosunku do szyfrowania,
- zerowy czas ustalania się 128-bitowego klucza szyfru,
- architektury iteracyjne dla trybów ze sprzężeniem zwrotnym,
- optymalizacja ze względu na szybkość działania układu,
- użycie układów XC2V1000 oraz XC2V6000.



Rysunek 26. Struktura algorytmów realizowanych w układach FPGA [242][243]



Rysunek 27. Zastosowana procedura projektowania algorytmów w układach FPGA [243]

Jednym z założeń była realizacja algorytmu szyfrowania z generacją podkluczy. Podklucze rundy zostały obliczone wewnątrz układu na podstawie klucza szyfru, a

nie na zewnątrz układu, co mogłoby znacznie obniżyć bezpieczeństwo systemu [68]. Rysunek 26 pokazuje realizację współbieżnej generacji podkluczy rundy w stosunku do przetwarzania rundy, przez co uzyskano zerowy czas ustalania się klucza szyfru.

Rysunek 27 przedstawia przyjętą standardową procedurę projektowania przy użyciu języka VHDL, symulacji funkcjonalnej HDL, syntezy logicznej, symulacji EDIF, syntezy fizycznej, symulacji czasowej HDL i fizycznej weryfikacji [74][172][242][244].

Zrealizowane moduły nadają się do powtórnego użycia jako gotowe makrobloki, znacznie przyspieszając proces projektowania. Są to specjalizowane produkty IPCore, które mogą zostać wykorzystane w celach komercyjnych [172].

8.8.3. Wyniki realizacji w FPGA

Tabela 24, Tabela 25 oraz Tabela 26 prezentuje wyniki realizacji algorytmów w architekturach iteracyjnych [244]. Gwiazdką oznaczono zastosowanie układu XC2V1000, gdy zużycie zasobów było mniejsze niż 5120 elementów CLB slice. W kolumnie *SBox* podano zużycie zasobów przez operację dominującą, a w kolumnie *Udział logiki* obliczono procentowy udział opóźnienia bloków logicznych w ścieżce krytycznej. Rysunek 28, Rysunek 29 i Rysunek 30 przedstawiają zależności przepustowości w funkcji używanych zasobów układu FPGA.

Tabela 24. Wyniki implementacji dla algorytmu Rijndael [243][244]

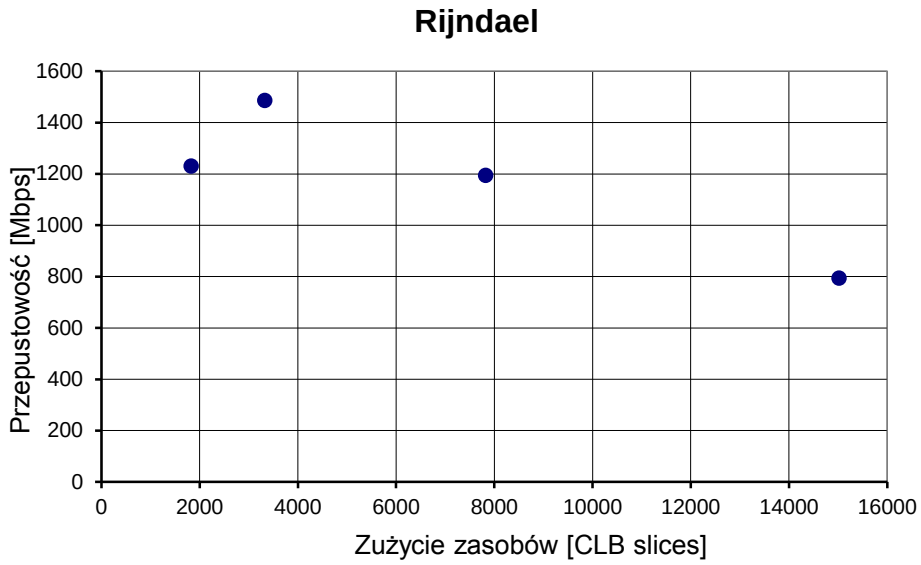
Arch	Zasoby [CLB slices]	SBox [CLB slices]	Ścieżka krytyczna [ns]	Udział logiki [%]	Maksymalna częstotliwość zegara [MHz]	Ilość cykli	Przepustowość [Mbps]
LU1*	1830	1280	10,414	37,6	96,025	10	1229,115
LU2*	3326	2560	17,233	24,5	58,028	5	1485,522
LU5	7823	6400	53,644	21,7	18,641	2	1193,050
LU10	15021	12800	161,324	14,8	6,199	1	793,434

Tabela 25. Wyniki implementacji dla algorytmu Serpent [243][244]

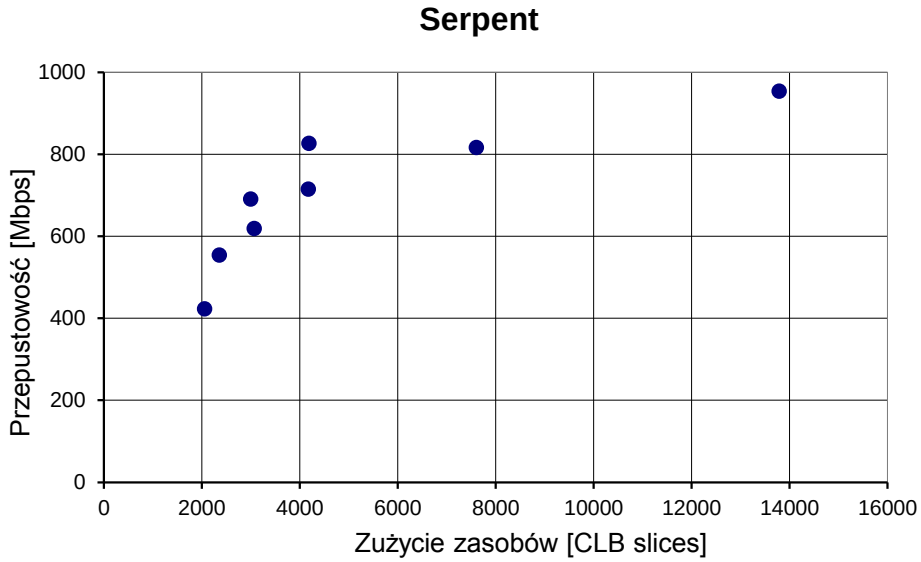
Arch	Zasoby [CLB slices]	SBox [CLB slices]	Ścieżka krytyczna [ns]	Udział logiki [%]	Maksymalna częstotliwość zegara [MHz]	Ilość cykli	Przepustowość [Mbps]
LU1*	2057	1088	9,166	31,5	109,099	33	423,171
LU2*	2361	1088	13,580	32,6	73,638	17	554,449
LU4*	2998	1088	20,586	37,4	48,577	9	690,869
LU4*	3074	1152	25,841	35,5	38,698	8	619,171
LU8*	4179	1088	35,820	33,1	27,917	5	714,685
LU8*	4192	1088	38,694	35,4	25,844	4	827,002
LU16	7610	2112	78,408	33,7	12,754	2	816,243
LU32	13788	4160	134,138	32,9	7,455	1	954,241

Tabela 26. Wyniki implementacji dla algorytmu Twofish [243][244]

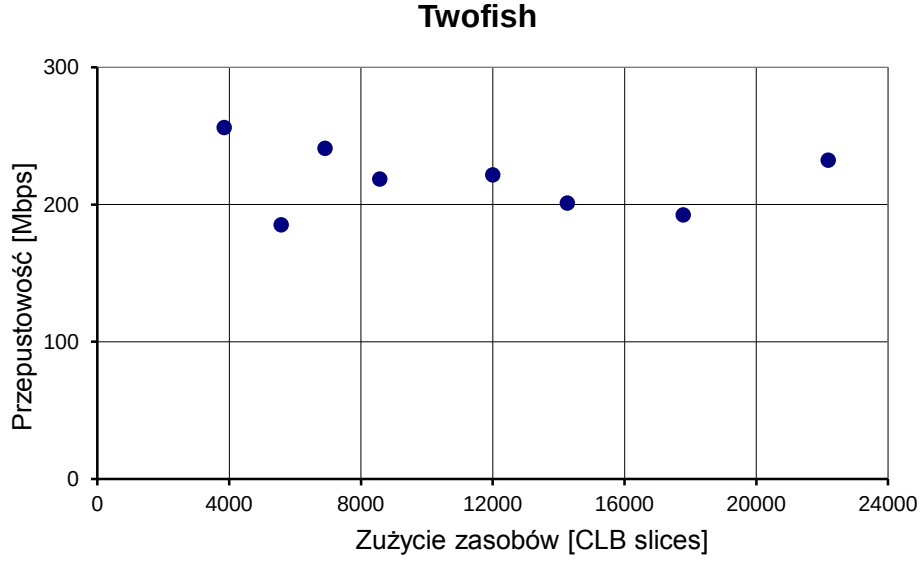
Arch	Zasoby [CLB slices]	SBox [CLB slices]	Ścieżka krytyczna [ns]	Udział logiki [%]	Maksymalna częstotliwość zegara [MHz]	Ilość cykli	Przepustowość [Mbps]
LU1*	3854	3216	24,997	50,4	40,005	20	256,031
LU1	5586	4824	38,438	29,5	26,016	18	185,002
LU2	6919	6432	53,145	34,8	18,816	10	240,851
LU2	8584	8040	65,081	31,8	15,365	9	218,531
LU3	12015	11256	96,256	33,3	10,389	6	221,631
LU4	14270	12864	127,405	33,6	7,849	5	200,934
LU5	17788	16080	166,415	30,2	6,009	4	192,290
LU6	22191	19296	183,678	32,9	5,444	3	232,291



Rysunek 28. Zależność przepustowości od zużycia zasobów dla algorytmu Rijndael [243][244]



Rysunek 29. Zależność przepustowości od zużycia zasobów dla algorytmu Serpent [243][244]



Rysunek 30. Zależność przepustowości od zużycia zasobów dla algorytmu Twofish [243][244]

8.8.4. Wnioski z realizacji w układach FPGA

Najszybszym algorytmem okazał się Rijndael – 1486 Mbps (architektura LU2), wolniejszy jest Serpent – 954 Mbps (architektura LU32), a najwolniejszy Twofish – 256 Mbps (architektura LU1) [244]. Warto zwrócić uwagę na spore dysproporcje przepustowości pomiędzy algorytmami, które nie ujawniają się przy realizacjach w architekturach potokowych [96] oraz rozwiązaniach programowych [8].

Zużycie zasobów przez moduły szyfrowania i generacji podkluczy rundy jest najmniejsze w architekturze LU1. Najmniej zasobów alokuje algorytm Rijndael – 1830, Serpent – 2057, a najwięcej Twofish – 3854 elementów CLB slice.

W przypadku Rijndael rozwijanie pętli powoduje wzrost zużywanych zasobów przy jednoczesnym spadku przepustowości (Rysunek 28). Dzieje się tak wskutek zwiększania się opóźnień programowalnych połączeń z 62% dla architektury LU1 do 85% dla architektury LU10. Dla Twofish przepustowość zależy w niewielkim stopniu od ilości rozwiniętych rund (Rysunek 30), gdyż ścieżka krytyczna jest stosunkowo długa i redukcja multiplekserów oraz przerzutników przez rozwijanie pętli wnosi niewielki zysk. Algorytmy Rijndael i Twofish charakteryzują się niską skalowalnością i powinny być stosowane z małą ilością rozwiniętych rund (architektury LU1 i LU2).

W Serpent dla architektur od LU1 do LU8 wzrost zużycia zasobów daje prawie liniowy wzrost przepustowości [41][83], natomiast w zakresie od LU8 do LU32 przyrost jest wolniejszy (Rysunek 29). Wynika to z zależności podstawienia SBox od numeru rundy, która dla architektur od LU1 do LU4 wymaga stosowania wielokrotnych bloków SBox, co wprowadza dodatkowe opóźnienia na ich przełączanie. Efektywne rozwijanie pętli jest możliwe dzięki bardzo prostej strukturze rundy [96]. Serpent charakteryzuje się najlepszą skalowalnością wśród przebadanych algorytmów.

Analiza czasowa wykazała, że dla Rijndael i Serpent operacja generacji klucza leży na ścieżce krytycznej. W pracach [68][96][242] dowiedziono, że w Rijndael generacja podkluczy wprowadza większe opóźnienie niż runda szyfrowania. Zwiększenie przepustowości jest możliwe na drodze dalszej optymalizacji modułu rozszerzania klucza. W Twofish ścieżka krytyczna przebiega przez przekształcenie rundy, a największe opóźnienie jest spowodowane obliczaniem podstawienia SBox. Wzrost przepustowości można uzyskać przez optymalizację modułu SBox.

Wnikliwe badania ujawniły, że udział bloków logicznych stanowi 15-50% całkowitego opóźnienia, a programowalne połączenia między nimi wnoszą pozostałe 50-85%. Wzrost przepustowości można uzyskać stosując układy FPGA z odmienną architekturą połączeń (np. *FastTrack Interconnect* w układach Altera [4][188]).

Operacjami zużywającymi najwięcej zasobów są podstawienia SBox: 8-bitowe dla Rijndael, 4-bitowe połączone w banki w Serpent oraz 8-bitowe zależne od klucza dla Twofish. Przeprowadzone analizy [96][125][242] wykazały, że dla Rijndael i Twofish podstawienia SBox, zajmują 60-80% zasobów, a w Serpent – około 50%. Podstawienia SBox zrealizowano za pomocą tablic LUT, gdyż układy Virtex II nie posiadają asynchronicznych pamięci ROM. Zastosowanie układów z pamięciami tego typu (np. bloki EAB w układach Altera [4][188]) zmniejszy zużycie zasobów.

W trakcie implementacji algorytmów kryptograficznych w układach FPGA ujawniło się wiele możliwości realizacji poszczególnych bloków, różniących się parametrami. Indukuje to szeroką przestrzeń projektową dla realizacji całego algorytmu.

8.9. Kryptoanaliza

W celu podniesienia poziomu bezpieczeństwa i przeciwdziałaniu modyfikacji algorytmu i odczytu kluczy sesji, algorytmy kryptograficzne są implementowane w

układach ASIC lub FPGA. Urządzenia kryptograficzne muszą spełniać wymagania NIST FIPS 140. Sprzęt realizujący szyfrowanie może zostać zaatakowany przez [27][252][253]:

- wprowadzanie błędów (ang. *fault injection*) – atak aktywny inwazyjny,
- manipulacja sygnałem zegara i zasilaniem (ang. *clock and power tampering*) – atak aktywny nieinwazyjny,
- analiza informacji z kanału bocznego (ang. *side channel information*): pobór mocy, emisja elektromagnetyczna, czas przetwarzania – atak pasywny.

W przeszłości ataki zostały przeprowadzone przy użyciu niedrogiego sprzętu [27][150]. Implementacja algorytmu kryptograficznego musi zapewniać odporność na ataki w kanale bocznym: analizę mocy, emisji elektromagnetycznej i atak czasowy. Atak przez analizę poboru mocy podczas wykonywania operacji przez układ realizujący szyfrowanie został zaproponowany w pracy [150].

W kryptografii kwantowej [173] każda próba pomiaru jest atakiem aktywnym i powoduje zmianę polaryzacji transmitowanych fotonów. Atak na kanał transmisyjny jest zatem natychmiast wykrywany.

8.9.1. Analiza mocy

Prosta analiza mocy SPA jest korelacją operacji algorytmu z poborem mocy [252][253]. Jeśli ścieżka wykonania algorytmu zależy od przetwarzanych danych wówczas SPA wykrywa sekwencję operacji i uzyskuje informację o kluczu. Przykładem jest atak na algorytm DES, gdzie różnice w poborze mocy są obserwowane dla permutacji, rotacji i instrukcji porównania, zwłaszcza w realizacjach programowych [27][150][151].

Różnicowa analiza mocy DPA jest korelacją wartości klucza z poborem mocy [252][253]. DPA działa w dwóch etapach; najpierw zapisuje się profile poboru mocy dla każdej operacji szyfrowania w tablicy T[*no_operations*][*no_samples*] oraz wartości szyfrogramu w tablicy C[*no_operations*]. Następnie wykonuje się:

- analizę korelacji [198][206],
- filtrowanie szumu przy użyciu testu odległości od średniej [150][151][180].

Oba testy weryfikują hipotezy na temat poszczególnego bajtu klucza.

Przy analizie korelacji do szyfrowania używany jest hipotetyczny klucz, następnie obliczana jest korelacja poboru mocy podczas szyfrowania hipotetycznym kluczem i zgadywanym kluczem. Jeśli bajt klucza został odgadnięty prawidłowo, korelacja osiąga maksimum. Korelację oblicza się z następującego wzoru:

$$\mathrm{Corr}(\mathrm{T},\mathrm{P}) = \frac{\mathrm{E}(\mathrm{T}*\mathrm{P})-\mathrm{E}(\mathrm{T})*\mathrm{E}(\mathrm{P})}{\sqrt{\mathrm{Var}(\mathrm{T})*\mathrm{Var}(\mathrm{P})}} \tag{32}$$

gdzie:

T – zbiór wartości poborów mocy (klucz zgadywany),
P – zbiór wartości poborów mocy (klucz hipotetyczny).

Analiza korelacji DPA 8 najstarszych bitów klucza algorytmu AES została opisana w pracach [198][206].

W teście odległości od średniej najpierw obliczane jest średnie zużycie mocy, a następnie wykonywane jest szyfrowanie za pomocą hipotetycznego klucza. Wartości zużycia mocy są przypisywane do jednego z dwóch zbiorów przez funkcję selekcji. Jeśli istnieje korelacja pomiędzy średnim poborem mocy całego zbioru i jednego z podzbiorów, wówczas dany podzbiór determinuje wartość jednego bitu [151][180].

$$\mathrm{R}[\mathrm{i}][\mathrm{j}] = \sum_{\mathrm{k}=0}^{\mathrm{k}_{\max}} \left(\mathrm{D}(\mathrm{i},\mathrm{C}[\mathrm{k}]) - \frac{1}{2}\right)(\mathrm{T}[\mathrm{k}][\mathrm{j}]) \tag{33}$$

gdzie:

R – dwuwymiarowa macierz różnicowych średnich poborów mocy,
i – n-bitowa wartość klucza, dla DES: n=6, $i \in \{0,\ldots,63\}$
j – index próbki mocy,
k – index operacji szyfrowania,
D – funkcja selekcji, dla DES: 0 - jeśli jednobitowa wartość permutacji P i odpowiadający bit funkcji L są identyczne; 1 - w przeciwnym przypadku,
C – zbiór wartości szyfrogramów,
T – zbiór wartości poborów mocy (klucz zgadywany).

Wartość R[i][j] wyznaczana za pomocą zależności (33) jest używana do dalszej analizy statystycznej. Atak na algorytm DES za pomocą testu odległości od średniej opisano w [150][151].

Wnioskowana analiza mocy IPA przebiega w dwóch etapach: profilowania i ekstrakcji klucza [93][100][252][253]. Podczas etapu profilowania wykonywane są operacje statystyczne na dużej liczbie wartości poborów mocy do nauki szczegółów implementacji, znalezienia kluczowych operacji i zidentyfikowania bitów klucza. Na etapie ekstrakcji klucza, klucz jest uzyskiwany z relatywnie małej liczby wartości poborów mocy.

8.9.2. Zapobieganie analizie mocy

Zapobieganie analizie mocy opiera się na dekorelacji wartości mocy z danymi wejściowymi i danymi klucza lub ukrywanie zmienności wartości mocy w celu zwiększenia złożoności ataku [206]. Podstawowe metody zmiany zużycia mocy [252][253]:

- wyrównywanie – wprowadzanie logiki kompensującej [27] (ang. *balancing*),
- dekorelacja – wprowadzenie dodatkowych operacji, np. maskowanie,
- randomizacja – wprowadzenie dodatkowego przypadkowego zużycia mocy.

Wyrównywanie mocy równoważy logikę kombinacyjną i przełączanie rejestrów, uniezależniając od klucza zwiększone totalne zużycie mocy i uniemożliwiając przeprowadzenie DPA. Specjalne biblioteki stosujące techniki WDDL zapewniają wyrównywanie mocy na poziomie bramek lub tranzystorów [179][206][258]. Inną koncepcją jest zastosowanie logiki prądowej [178][239], w której bramki cyfrowe charakteryzują się stałym poziomem zużywanego prądu w różnych trybach pracy.

Dekorelacja mocy maskuje liniowe operacji i modyfikuje nieliniowe [2], wymagając znaczących zmian algorytmu i spowalniając ich wykonanie [206]. SPA ujawnia tylko wagę Hamminga zamaskowanej wartości. W pracy [106] wykazano, że maskowanie addytywne może być łatwo złamane przy użyciu ataku Hamminga.

Randomizacja mocy generuje pseudolosowy szum cyfrowy i wymaga dodania elementów pobierających moc w sposób przypadkowy, zwiększając całkowity pobór mocy lub wprowadzając dodatkowe opóźnienie [34][206][232]. Nie zapobiega to atakowi DPA, ale czyni go nieefektywnym z powodu konieczności zbierania danych w długim okresie czasu [34][151].

8.10. Wyrównywanie mocy implementacji FPGA algorytmu AES

Wyrównywanie mocy implementacji algorytmu AES w układzie FPGA realizowano w pracy [250] i rozwijano w pracach [252][253].

8.10.1. Proponowana metoda projektowania

W celu wyrównania poboru mocy w układach synchronicznych należy zapewnić, że stała liczba przerzutników (ang. *flip-flop*) przełącza stan w każdym cyklu zegara. Dodatkowy przerzutnik Q_2 jest dodany równolegle do oryginalnego przerzutnika Q_1 i dokładnie jeden z nich przełącza stan w każdym cyklu zegara:

$$Q_1(t-1) \oplus Q_1(t) \oplus Q_2(t-1) \oplus Q_2(t) = '1' \tag{34}$$

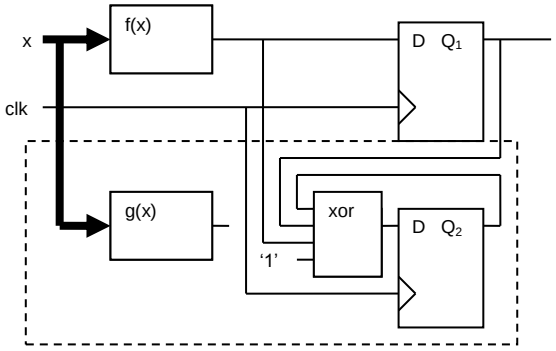
Metodę tę zaproponowano niezależnie od [180].

W celu wyrównania poboru mocy logiki kombinacyjnej należy zapewnić, że stała liczba wyjść zmienia stan dla każdej pojedynczej zmiany wektora wejściowego. Komplementarna funkcja logiczna g(x) jest dodana równolegle do oryginalnej funkcji logicznej f(x) i dokładnie jedno wyjście zmienia stan na każdą pojedynczą zmianę wektora wejściowego:

$$f(x) \oplus g(x) = XOR(x) \tag{35}$$

Funkcja f(x) może być dowolnego typu, co stanowi generalizację metod zaproponowanych w pracy [258].

Łączne zaaplikowanie obu metod opisanych zależnościami (34) i (35) prowadzi do schematu, który ilustruje Rysunek 31 [252][253]. Dodatkowe elementy są zaznaczone przerywaną linią. Ponieważ niemożliwe jest wstawienie dodatkowych bloków bezpośrednio w kodzie HDL, układ jest modyfikowany po syntezie logicznej.



Rysunek 31. Zaproponowana metoda wyrównywania mocy [252][253]

8.10.2. Przyjęte założenia

Pobór mocy jest często estymowany przez zliczanie przełączeń przerzutników w kodzie VHDL [180][198] lub mierzony w rzeczywistym środowisku [151][198]. W bieżących eksperymentach użyto narzędzia XPower, które umożliwia czasową symulację mocy na podstawie pliku wejściowego w formacie VCD. Jednakże symulator XPower został zorientowany na analizę najgorszego przypadku (ang. *worst case analysis*) w projektach małej mocy, a nie na pomiary dokładnych wartości mocy. W wyniku wartości mocy są przeestymowane, co nie ma znaczenia dla metod DPA, jeśli tylko przeestymowanie jest proporcjonalne [252][253].

Implementacja sprzętowa została wykonana w układzie Xilinx Spartan-II E FPGA. Wykonano dwie realizacje algorytmu AES: standardową (bez wyrównywania mocy) oraz z wyrównywaniem mocy, w której bloki komplementarne zostały dodane po syntezie logicznej. Wybrano iteracyjną architekturę szyfru w celu uproszczenia ataku DPA. Zaimplementowano jedną rundę AES z wewnętrznymi rejestrami potokowymi po każdej operacji. Szyfrowanie 128-bitowego bloku wejściowego przy 128-bitowym kluczu wymaga 10 iteracji (40 cykli zegara).

Atak DPA wykonano przy użyciu analizy korelacji zdefiniowanej zależnością (32), opisanej w pracy [198][206], używając przypadkowych wartości bloków wejściowych dla N=2000 operacji szyfrowania. Podczas symulacji zostały wygenerowane klucze rundy i dla każdej z N=2000 operacji moduł szyfrowania został resetowany i wykonywano pierwsze cztery cykle zegarowe (pierwszą rundę szyfrowania), gdyż podklucz pierwszej rundy jest równy kluczowi szyfrowania. Dane wygenerowane przez XPower zostały zebrane i skorelowane z wyliczoną programowo liczbą przełączeń przełączników dla hipotetycznego klucza.

8.10.3. Wyniki realizacji wyrównywania mocy

Przeprowadzone eksperymenty w środowisku programowym wykazały, że atak na AES jest możliwy tylko na operacje KeyAdd, ByteSub i ShiftRow w pierwszej rundzie [252]. Następna operacja MixColumn powoduje interakcję pomiędzy bajtami, co wprowadza dekorelacje podczas częściowego zgadywania klucza. Do wyznaczenia wartości klucza na podstawie liczby przełączeń operacji KeyAdd i ShiftRow (jednakowo dla obu operacji) należy wykonać około 500 szyfrowań (Rysunek 33). Dla operacji ByteSub wystarczy wykonać około 200 szyfrowań (Rysunek 34), gdyż jej nieliniowość powoduje szybszą dekorelację dla niewłaściwego klucza.

Zbadano wpływ dokładności wyznaczania liczby przełączeń na wyniki analizy DPA. Przeprowadzono 2000 szyfrowań, w których zastosowano dwa modele szumu: addytywny i kwantyzacyjny. Szum addytywny charakteryzuje rzeczywiste środowisko pomiarowe, natomiast szum kwantyzacyjny występuje podczas przetwarzania analogowo-cyfrowego. Analiza DPA umożliwia filtrację szumu addytywnego od SNR=8 (Rysunek 35), natomiast szum kwantyzacyjny jest filtrowany już od wartości SNR=2 (Rysunek 36) [252].

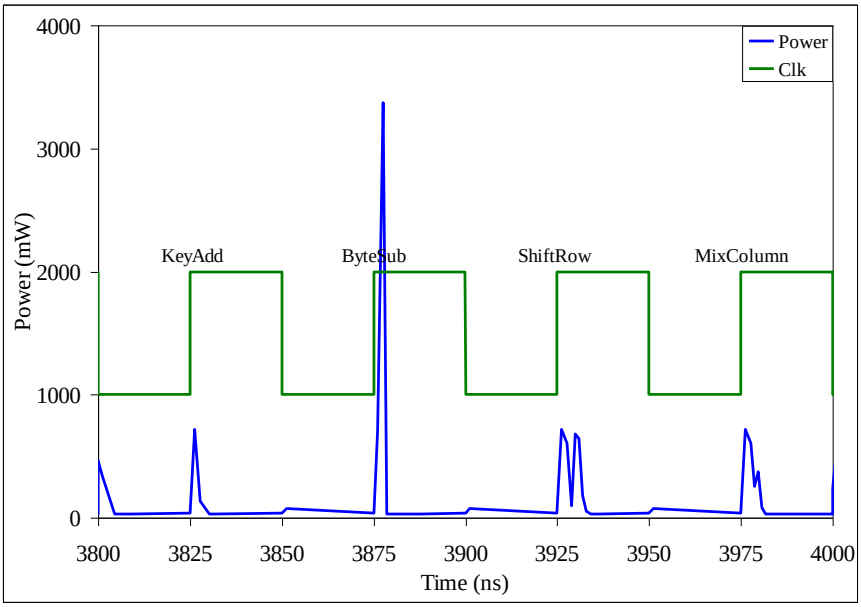
Zaimplementowano dwie wersje algorytmu AES. Standardowa wersja zużywa 3931 elementów CLB i minimalny okres sygnału zegarowego wynosi 15,5 ns. Wersja z wyrównywaniem poboru mocy wymaga 6831 elementów CLB i minimalny okres sygnału zegarowego wynosi 18 ns. Niemal dwukrotnie większe zużycie zasobów jest spowodowane generacją logiki komplementarnej dla modułu szyfrowania (dla modułu generacji kluczy rundy logika komplementarna nie została wygenerowana). Niższa maksymalna częstotliwość pracy jest efektem większej liczby połączeń programowalnych w układzie FPGA [252][253].

XPower generuje dane o poborze mocy z rozdzielczością ustaloną przez użytkownika. Zbyt niska rozdzielczość powoduje kumulację wartości mocy z wielu operacji do jednego przedziału czasowego. Zbyt wysoka rozdzielczość rozdziela wartości mocy z jednej operacji do wielu przedziałów czasowych. W wyniku eksperymentów rozdzielczość została ustalona zgodnie z wartością opóźnienia wprowadzanego przez przerzutnik (około 1 ns), co zapewnia, że wartości poboru mocy związane z przełączaniem przerzutników są akumulowane w jednym przedziale czasowym.

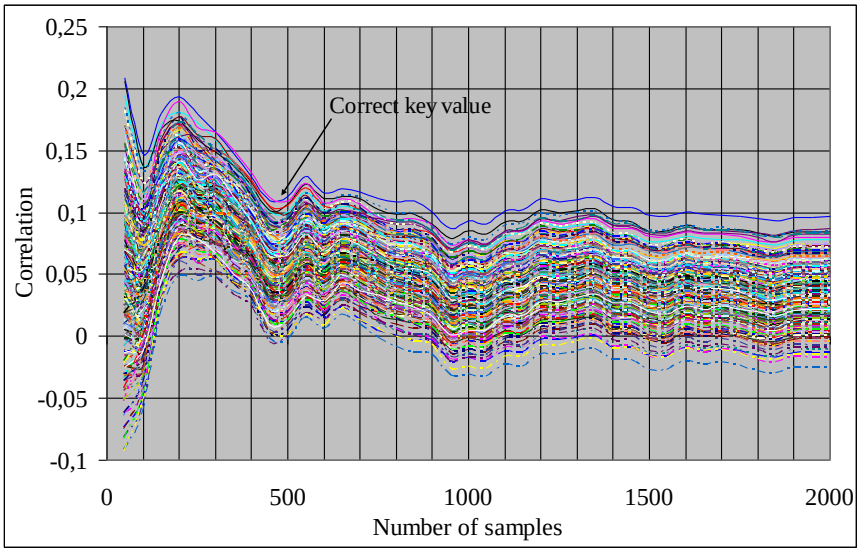
Aby wydobyć poprawne dane z przebiegu mocy w czasie, należy określić dokładne punkty w czasie dla wszystkich operacji. Wykonano dwa rodzaje symulacji mocy. Pierwsza symulacja była przeprowadzona po procesie odwzorowania (ang. *mapping*), w której brano pod uwagę opóźnienie przełączania przerzutnika (1,2 ns), druga symulacja była przeprowadzona po procesie rozmieszczania i trasowania (ang. *place and route*), w której brano pod uwagę dodatkowo opóźnienie propagacji sygnału (1,8 ns). Przykład przebiegów czasowych przedstawia Rysunek 32 oraz Rysunek 37.

W trakcie eksperymentów ujawniły się ograniczenia symulatora XPower. Maksymalny rozmiar pliku VCD z pobudzeniami wynosi 2 GB, a maksymalny czas symulacji jest równy 2,147 ms (32-bitowa liczba ze znakiem wyrażona w ps), co uniemożliwia stosowania dużej ilości szyfrowań w symulacji [252].

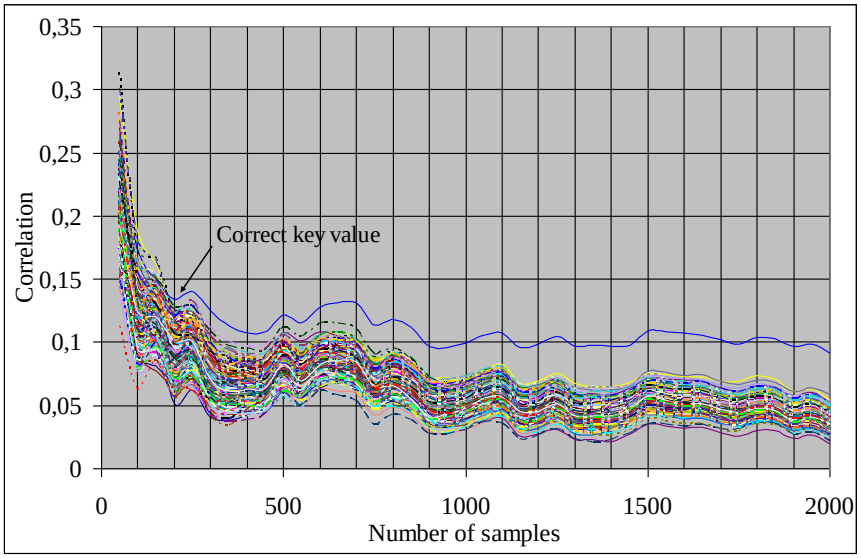
Atak DPA został przeprowadzony na obie implementacje algorytmu AES. DPA dla implementacji standardowej pokazał, że klucz szyfrowania może być określony przez wykonanie N=500 operacji szyfrowania i analizie poboru mocy operacji KeyAdd, ByteSub lub ShiftRow w pierwszej rundzie szyfrowania. Rysunek 33 przedstawia zależność wartości korelacji w funkcji ilości wykonanych operacji szyfrowania. Dla implementacji zmodyfikowanej moc pobierana podczas przełączania przerzutników była niemal stała i oscylowała wokół 5% zakresu. Analiza DPA dla N=2000 operacji szyfrowania nie wykazała żadnej korelacji pomiędzy poborem mocy i wartością klucza, co ilustruje Rysunek 38. Brak korelacji dowodzi, że zaproponowana metoda projektowania zwiększa odporność na DPA [252][253].



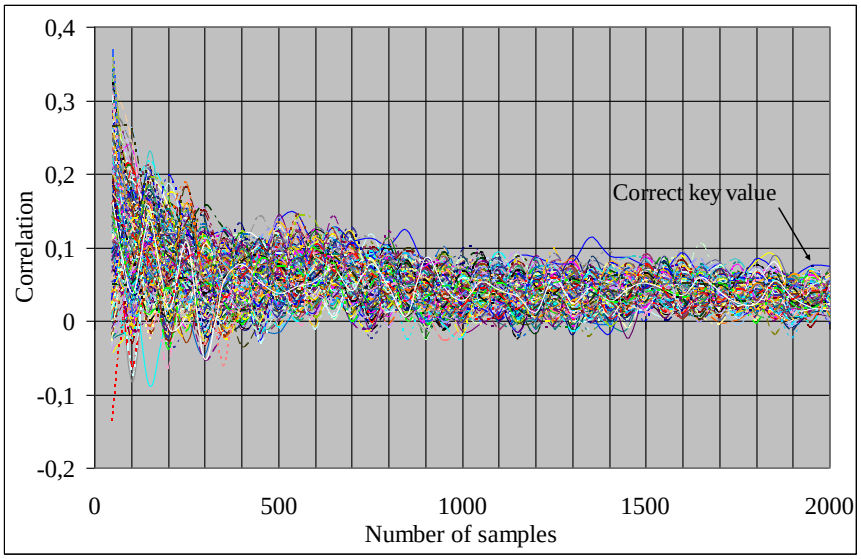
Rysunek 32. Przebieg mocy podczas 1 rundy szyfrowania AES [252][253]



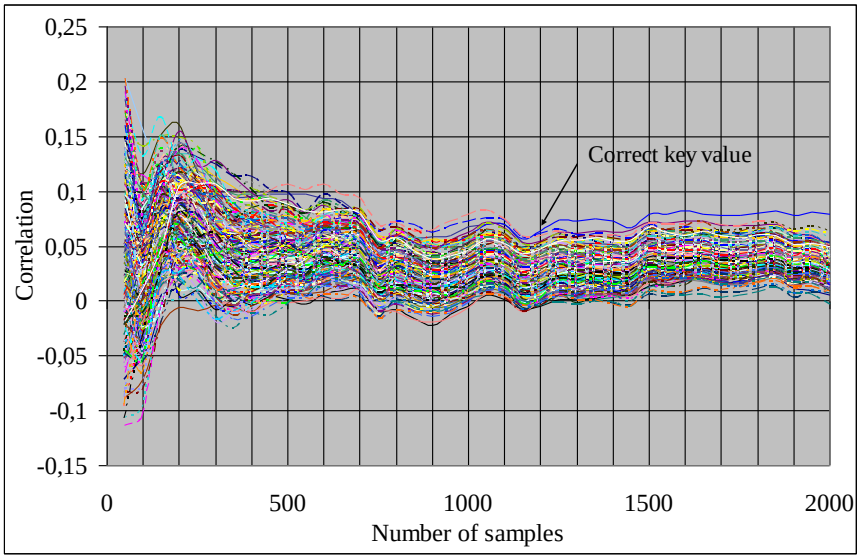
Rysunek 33. Korelacja poboru mocy operacji KeyAdd [252][253]



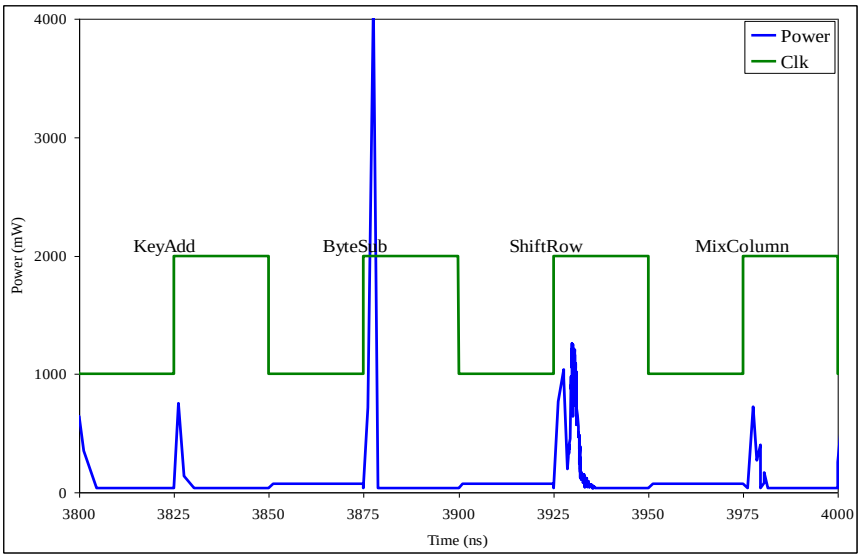
Rysunek 34. Korelacja poboru mocy operacji ByteSub [252]



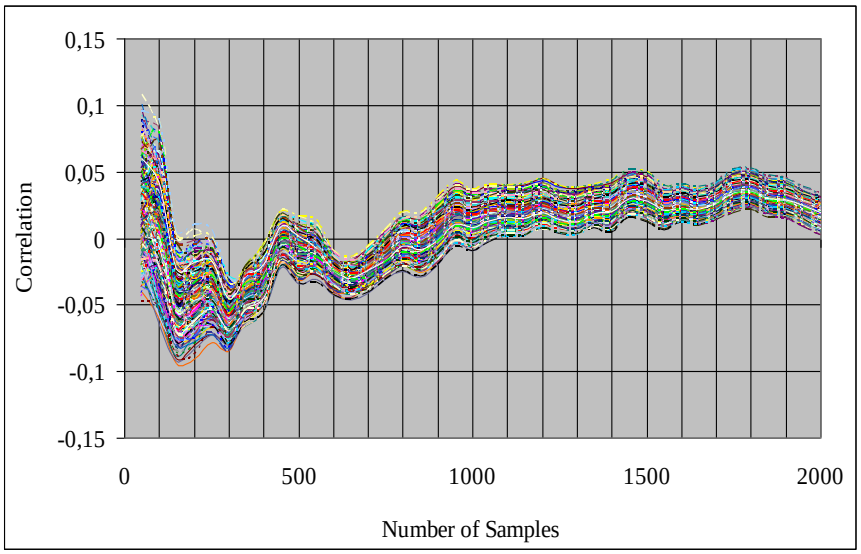
Rysunek 35. Korelacja poboru mocy operacji ByteSub (szum addytywny SNR=8) [252]



Rysunek 36. Korelacja poboru mocy operacji ByteSub (szum kwantyzacyjny SNR=4) [252]



Rysunek 37. Przebieg mocy podczas 1 rundy szyfrowania zmodyfikowanym AES [252][253]



Rysunek 38. Korelacja poboru mocy zmodyfikowanej operacji KeyAdd [252][253]

8.10.4. Wnioski z realizacji wyrównywania mocy

Wyniki eksperymentów potwierdzają, że DPA jest bardzo silnym narzędziem analizy, które potrafi dokonać filtracji szumu przy małej wartości SNR i nie wymaga dużych rozdzielczości pomiarów [252]. Wyniki pokazują również, że stosunkowo łatwo jest dokonać ekstrakcji klucza ze standardowej implementacji algorytmu Rijndael, niezabezpieczonej w odpowiedni sposób. Naraża to cały system na ujawnienie klucza szyfrowania, a przez to umożliwia dostęp do zaszyfrowanych danych przez strony nieupoważnione.

Zapobieganie atakom na podstawie informacji z kanału bocznego jest niezbędne w projektowaniu sprzętowych i programowych realizacji algorytmów szyfrowania. Spośród wielu zaprezentowanych metod przeciwdziałania atakom przez analizę DPA, zaproponowana metoda wyrównywania mocy jest relatywnie prosta do zastosowania w automatycznym projektowaniu układów cyfrowych [252].

Uzyskane wyniki wskazują także jakie zmiany należy wykonać w realizacji algorytmu Rijndael, aby utrudnić atak DPA. W celu dekorelacji mocy względem danych wejściowych należy dokonać rozproszenia mocy w czasie. Operacje KeyAdd,

ByteSub i ShiftRow powinny być realizowane asynchronicznie, a stopień rejestrowy należy zastosować jedynie po operacji MixColumn [252].

8.11. Podsumowanie

Architektura realizacji algorytmu kryptograficznego powinna być dostosowana do wymagań aplikacji. Obecne standardy szyfrowania wymuszają szyfrowanie danych za pomocą trybów ze sprzężeniem zwrotnym takich jak CBC, CFB i OFB [184][214][242][245]. W trybach ze sprzężeniem zwrotnym nie jest możliwa równoległa praca modułów szyfrujących (równoległość aplikacji), za wyjątkiem deszyfrowania w CBC, stąd nie można wykorzystać architektur potokowych do jednoczesnego przetwarzania wielu bloków. Generacja kluczy rundy powinna odbywać się wewnątrz układu, z uwagi na zapewnienie bezpieczeństwa.

Algorytmy kryptograficzne stanowią przykład systemów transformujących dane i są modelowane za pomocą grafu DFG. Większość benchmarków używa najczęściej stosowanych algorytmów z kluczem prywatnym: 3DES, Blowfish, IDEA, MARS, RC4, RC6, AES [29]. W niniejszej pracy analizowano algorytmy MARS, RC6, Rijndael, Serpent i Twofish.

9. Dodatek B: Architektury heterogenicznych implementacji algorytmów kryptograficznych

Protokoły w kryptografii z kluczem publicznym są doskonałym przykładem koncepcji HW/SW Codesign [232]. Protokół, generacja kluczy i proces zarządzania mają naturę sekwencyjną, podczas gdy algorytm (mnożenie modularne dużych liczb) może być lepiej realizowany w strukturach równoległych i potokowych. Realizacje sprzętowe algorytmu są wystarczająco szybkie, lecz nie są odpowiednie dla sekwencji protokołu i nie mogą być stosowane do zmian algorytmu. Realizacje programowe adaptują się szybciej, są jednak wolniejsze i mniej bezpieczne [68][69].

9.1. Atrybuty kryptograficznego systemu wbudowanego

Pożądanymi atrybutami systemu wbudowanego są [73][192]:

- krótki czas wytwarzania produktu (ang. *time to market)*,
- wydajność gwarantująca spełnienie ograniczeń czasu rzeczywistego,
- niski pobór mocy, lub względny pobór mocy w stosunku do wydajności,
- łatwość programowania, śledzenia i testowania (ang. *programmability and debuggability*),
- koszty projektowania, narzędzi i wytwarzania,
- dostępność konserwacji i serwisowania – możliwość naprawy uszkodzonych komponentów,
- wiarygodność i niezawodność – prawdopodobieństwo właściwej operacji nawet w obecności uszkodzeń poszczególnych komponentów,
- bezpieczeństwo – prawdopodobieństwo braku uszkodzeń katastroficznych.

W pracach [82][243] przedstawiono kryteria oceny systemu kryptograficznego:

- fizyczne bezpieczeństwo – brak możliwości nieautoryzowanej modyfikacji lub odczytu zawartości,
- rekonfigurowalność – możliwość wprowadzania zmian,
- elastyczność – możliwość pracy z wieloma algorytmami,
- przenośność – możliwość pracy w wielu środowiskach z wieloma interfejsami,
- przepustowość – ilość szyfrowanych danych w jednostce czasu,
- efektywność przetwarzania – ilość wykonywanych operacji do realizacji pożądanej funkcjonalności,
- możliwość przetwarzania równoległego – możliwość przyspieszenia przez replikację zasobów,
- koszt projektowania i wytworzenia – koszt prototypowania i weryfikacji.

Tabela 27 przedstawia porównanie parametrów implementacji systemu kryptograficznego dla realizacji programowych, ASIC i FPGA.

Tabela 27. Porównanie parametrów implementacji systemu kryptograficznego [243]

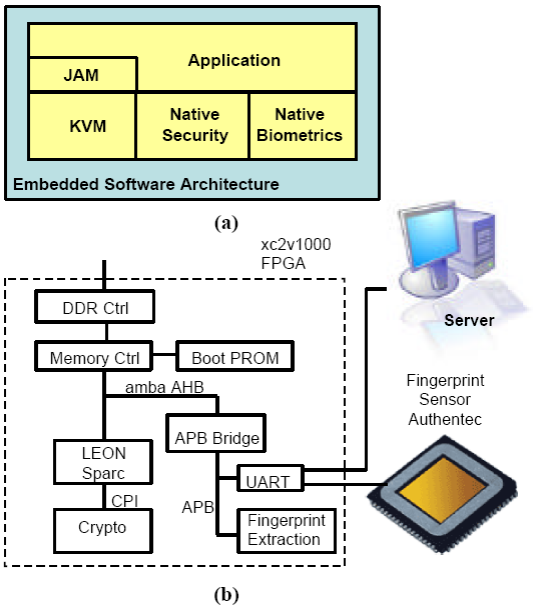
	Software	ASIC	FPGA
Bezpieczeństwo	Małe	Bardzo duże	Duże
Rekonfigurowalność	Bardzo duża	Brak	Duża
Elastyczność	Bardzo duża	Brak	Średnia
Przenośność	Bardzo duża	Mała	Średnia
Przepustowość	Średnia	Bardzo duża	Duża
Efektywność	Mała	Bardzo duża	Duża
Równoległość	Ograniczona	Tak	Tak
Koszty wytwarzania	Niskie	Bardzo wysokie	Średnie
Zużycie mocy	Bardzo duże	Niskie	Średnie

9.2. Procesor z dedykowanym akceleratorem

We wszystkich przypadkach podziału funkcjonalności na sprzęt i oprogramowanie dokonano w sposób ręczny. Na podstawie inicjalnego rozwiązania programowego identyfikowano moduły, których realizacja sprzętowa spowoduje przyspieszenie. Moduły te następnie przenoszono do sprzętu aż do wyczerpania zasobów.

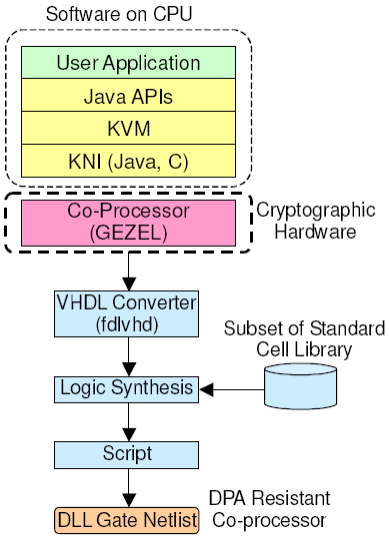
9.2.1. Akceleracja algorytmów symetrycznych

W pracach [123][222][265] przedstawiono akcelerację algorytmu AES w środowisku Java. Układ FPGA Xilinx Virtex2 zawiera 32-bitowy procesor LEON Sparc i dedykowany koprocesor kryptograficzny, co ilustruje Rysunek 39. Koprocesor jest dołączony do CPU przez magistralę CPI, która dostarcza dwa 64-bitowe porty danych i 10-bitowy port instrukcji, a także oferuje zestaw instrukcji i rejestrów dla CPU. W wyniku uzyskano 333-krotne przyspieszenie szyfrowania.



Rysunek 39. Architektura systemu z dedykowanym koprocesorem [123][265]

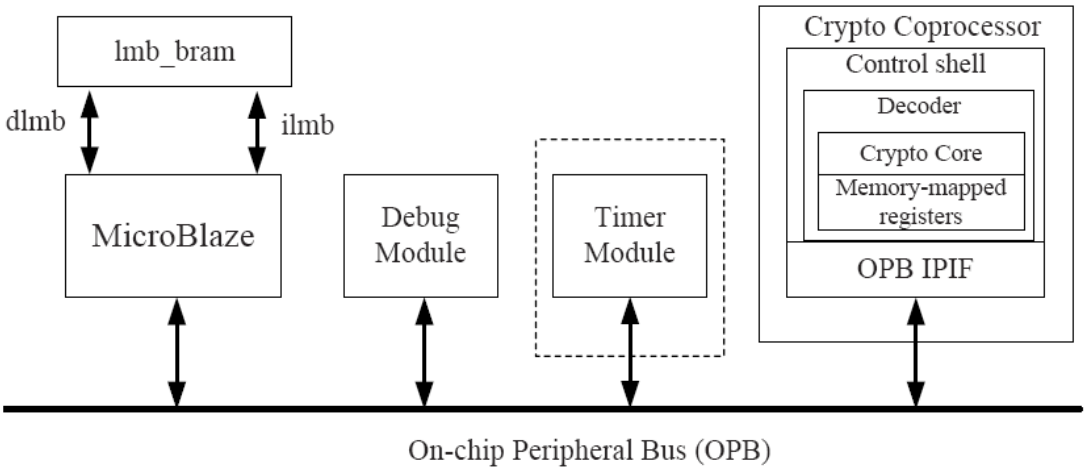
W pracy [179] przedstawiono heterogeniczną implementację algorytmu AES na platformie z 32-bitowym procesorem SH3-DSP z koprocesorem realizowanym w układzie ASIC 0.18 μm. Wykonano symulacje za pomocą symulatora ISS i środowiska GEZEL. Zaproponowano 2 architektury systemu: a) sprzętowa realizacja algorytmu, b) sprzętowe przechowywanie kluczy. Sprzęt został zaimplementowany z użyciem techniki WDDL [258] (patrz Rysunek 40) w celu przeciwdziałania analizy mocy DPA [249][252][253]. W wyniku otrzymano przyspieszenie x10 i x25 algorytmu AES względem realizacji Java.



Rysunek 40. Diagram projektowania z użyciem techniki WDDL [179]

W pracy [105] opisano implementację algorytmu RC6 wchodzącego w skład protokołu IPSec na platformie z 8-bitowym procesorem PIC i układem FPGA Altera APEX. Akcelerator komunikuje się z CPU za pomocą prostego protokołu handshake.

W pracy [110] przedstawiono implementację algorytmów AES i PRESENT na platformie z układem Xilinx Spartan III z procesorem Microblaze, co ilustruje Rysunek 41. Dla procesora StrongARM symulację wykonano za pomocą symulatora ISS i środowiska GEZEL. Akcelerator komunikuje się z procesorem za pomocą magistrali OPB. W wyniku uzyskano przyspieszenie x1,6-22 dla StrongARM i x5,6-44,6 dla MicroBlaze przy jednoczesnej redukcji mocy.

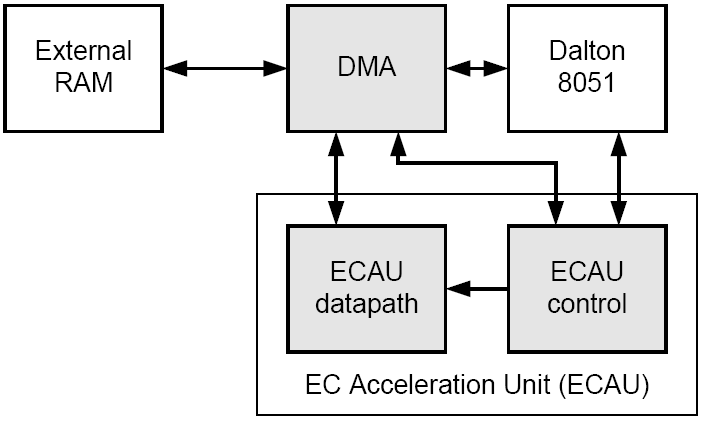


Rysunek 41. Schemat systemu realizowanego na platformie Xilinx Spartan III [110]

9.2.2. Akceleracja algorytmów asymetrycznych

Przegląd architektur i osiągniętych wyników zawierają prace [118][152].

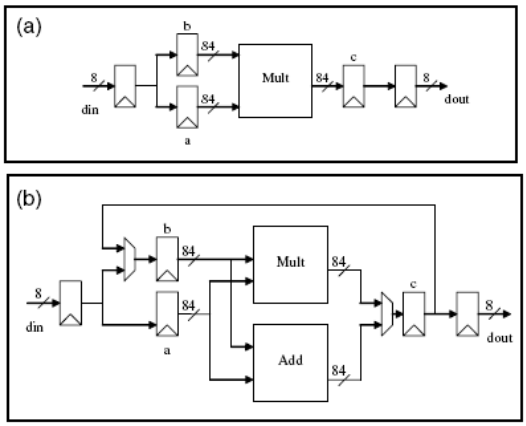
W pracy [152] przedstawiono realizację algorytmu ECC 163-bit i 191-bit na platformie z 8-bitowym procesorem 8051, co obrazuje Rysunek 42. Zaproponowano 3 architektury koprocesora: rozszerzenie listy rozkazów, arytmetyka w ciele GF, operacje grupowe. Osiągnięto przyspieszenie x50 względem rozwiązania programowego na 8051.



Rysunek 42. Schemat systemu Dalton 8051 [152]

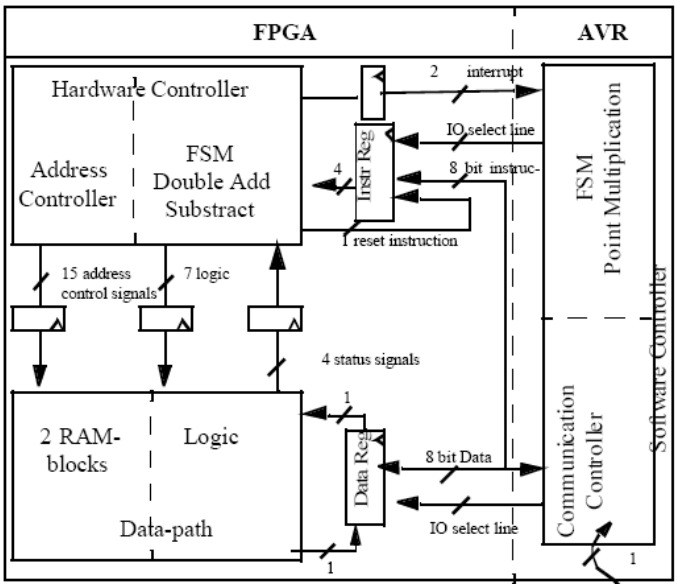
W pracy [35] opisano implementację algorytmu ECC 163-bit na platformie z 8-bitowym procesorem ATmega128 i koprocesorem realizowanym w układzie ASIC 0.18 µm, w zastosowaniu do sieci WSN, zoptymalizowane na niski pobór mocy. Zaproponowano dwie architektury koprocesora różniące się ilością wbudowanych rejestrów i w wyniku otrzymano przyspieszenie x121 i x141 i redukcję zużycia energii 110- i 182-krotną względem rozwiązania bez koprocesora.

W pracach [31][216] opisano implementację algorytmu HECC 163-bit na platformie z 8-bitowym procesorem 8051 i AVR i koprocesorem realizowanym w FPGA. Rysunek 43 przedstawia przebadane 2 opcje architektury: a) implementacja mnożnika GF, b) pełen akcelerator GF. W wyniku uzyskano przyspieszenie pracy algorytmu odpowiednio x29-38 oraz x47-60 względem rozwiązania na procesorze 8051 i AVR.



Rysunek 43. Schemat koprocesora HECC: a) mnożnik GF, b) mnożnik i sumator GF [31]

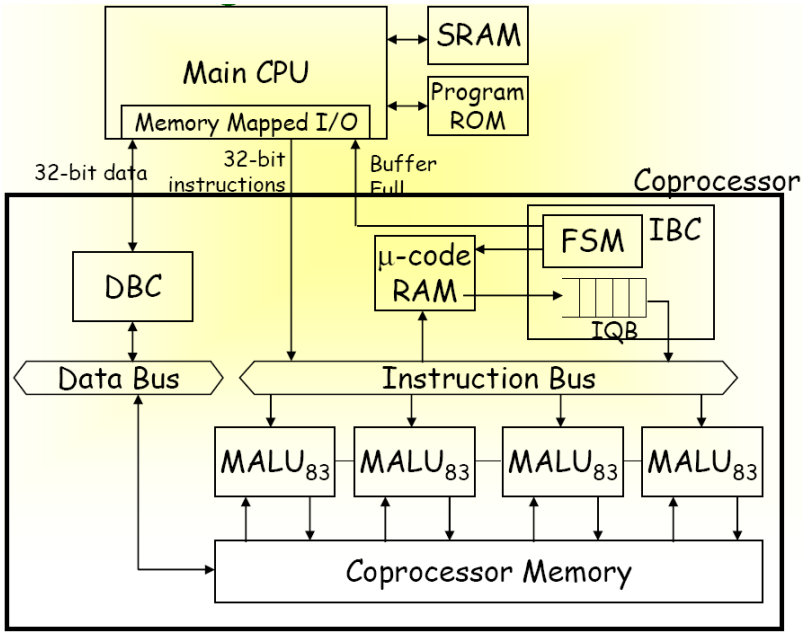
W pracach [90][128][157] przedstawiono implementacje algorytmu ECC 113-bit i 72-bit w układzie SoC Atmel FPSLIC z wbudowanym 8-bitowym procesorem AVR i układem FPGA (patrz Rysunek 44). W [90] zaproponowano 3 architektury koprocesora (z 32-bitowym mnożnikiem Karatsuba, z koprocesorem w ciele skończonym oraz rozszerzonym koprocesorem), uzyskując przyspieszenie odpowiednio x2, x33 oraz x283.



Rysunek 44. Implementacja algorytmu ECC w układzie FPSLIC [128][157]

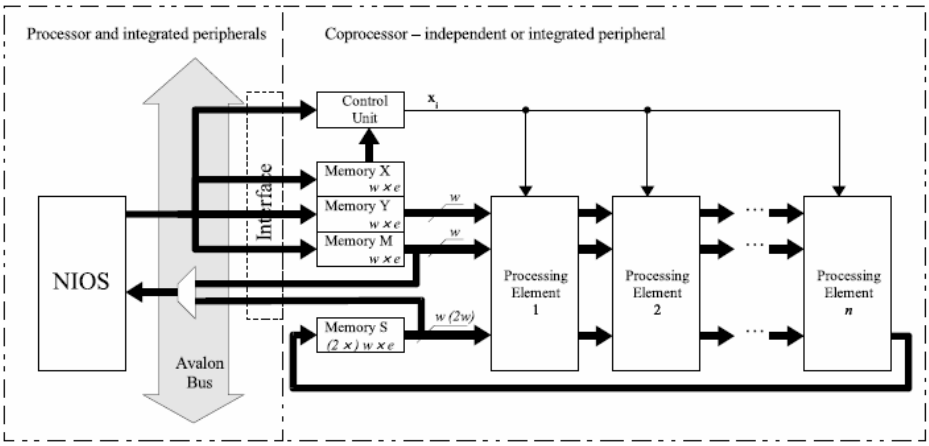
W pracy [215] zaprezentowano realizacje algorytmu ECC 160-bit w systemie z procesorem 8-bitowym 8051 i ściśle zintegrowanym koprocesorem FPGA. Symulację wykonano za pomocą symulatora ISS i środowiska GEZEL. Implementacja została zmodyfikowana przez dodanie redundantnych mnożników i sumatorów, aby operacje dodawania punktu i podwajania punktu w ciele GF(p) były wykonane w stałym czasie i z jednakowym poborem mocy. W wyniku osiągnięto odporność na ataki czasowe i analizę mocy kosztem 15% spadku wydajności i 1% wzrostu zużycia zasobów względem rozwiązania podstawowego.

W pracy [217] zaprezentowano realizację algorytmów HECC 83-bit oraz ECC 163-bit w systemie z 32-bitowym procesorem ARM i koprocesorem FPGA Xilinx Virtex II. Symulację wykonano za pomocą symulatora ISS i środowiska GEZEL. Zaproponowano 5 różnych architektur koprocesora superskalarnego (przykład pokazuje Rysunek 45) i w wyniku osiągnięto znaczne przyspieszenie działania algorytmów.



Rysunek 45. Architektura koprocesora skalarnego z przetwarzaniem równoległym [217]

W pracy [232] przedstawiono implementację algorytmu RSA w układzie SoC Altera APEX z 16/32-bitowym procesorem Nios i koprocesorem realizowanym w układzie FPGA, co ilustruje Rysunek 46. Komunikacja pomiędzy procesorem i koprocesorem odbywa się przez magistralę Avalon. Zaproponowano 3 różne architektury koprocesora: a) potokowy koprocesor modularnego mnożenia, b) 2 potokowe koprocesory modularnego mnożenia c) rozwiązanie całkowicie sprzętowe. Równoległa praca dwóch koprocesorów może zwiększyć odporność sprzętu na ataki w kanale bocznym tj. atak czasowy i analiza mocy [249][252][253]. W wyniku przeniesienia mnożnika Montgomery do sprzętu uzyskano przyspieszenie algorytmu RSA.



Rysunek 46. Diagram blokowy procesora Nios i mnożników modularnych [232]

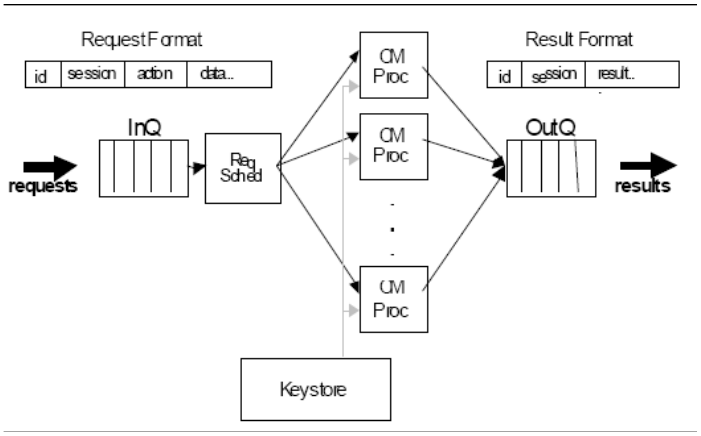
9.3. Procesor z koprocesorem rozszerzającym listę rozkazów

Porównanie zaproponowanych rozszerzeń listy rozkazów dla algorytmów kryptograficznych i dogłębny przegląd literatury w tym temacie można znaleźć w pracy [29]. Z reguły dodatkowe instrukcje są projektowane w oparciu o analizę wykonania danego algorytmu, przyspieszając nieefektywne realizacje programowe. Inne podejście zaproponowano w pracy [98], gdzie zaprezentowano metodologię automatycznej identyfikacji i selekcji rozszerzenia zbioru instrukcji. W tym celu zaproponowano algorytm klastrowania przez spiralne przeszukiwanie sąsiedztwa. Na platformie Xilinx Virtex II Pro z 32-bitowym procesorem PowerPC uzyskano przyspieszenie algorytmu Twofish x4,5.

W literaturze spotyka się projekty rozszerzeń listy rozkazów dla procesorów 8-bitowych Atmel AVR i Intel 8051 [80][118][155] oraz procesorów 32- i 64-bitowych [29][46][79][133][212][221][223][275].

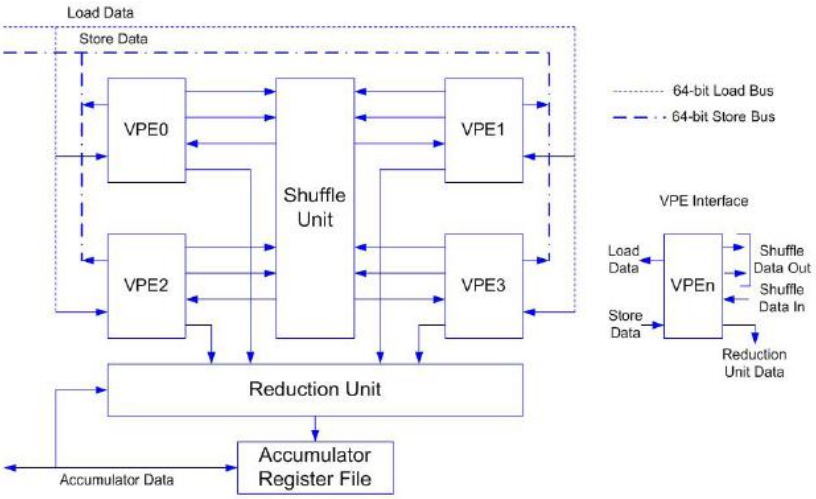
9.3.1. Koprocesor dla algorytmów symetrycznych

W pracach [46][275] do procesora Alpha 21264 zaprojektowano VLIW koprocesor CryptoManiac z nowymi instrukcjami do przyspieszania wykonania algorytmów symetrycznych. Dodano instrukcje mnożenia modularnego MULMOD, rotacje ROL, ROR, ROLX, RORX, podstawienia tablicowe SBOX, SBOXSYNC i permutację XBOX. Dodano także instrukcje łączące operacje arytmetyczne i logiczne, często występujące w algorytmach symetrycznych. Komunikacja procesora z koprocesorem odbywa się przez kolejki wejściowe i wyjściowe, co ilustruje Rysunek 47. Dzięki zastosowaniu koprocesora CryptoManiac szybkość szyfrowania algorytmów symetrycznych wzrosła nawet 2,25-krotnie (Rijndael), a średnio o 59%.



Rysunek 47. Schemat koprocesora CryptoManiac [275]

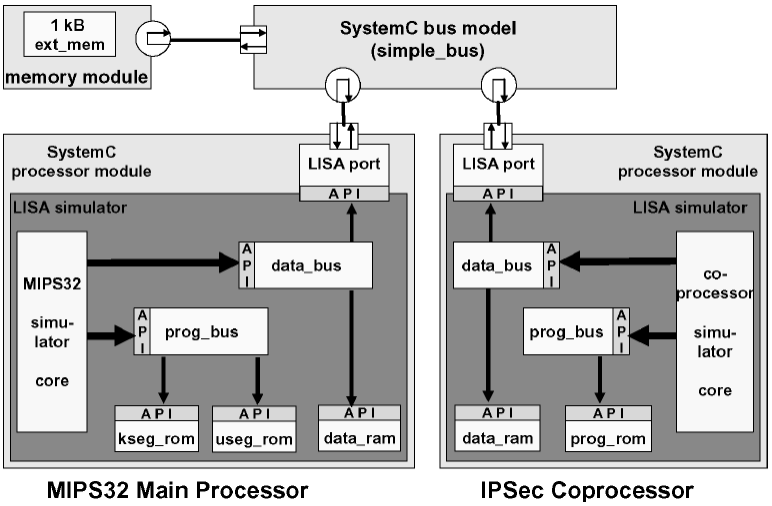
W pracy [133] do procesora SandBlaster DSP zaprojektowano koprocesor z nowymi instrukcjami do przyspieszania algorytmu AES na platformie SandBridge. Dodano instrukcje *roundXOR, shift_rows, sbox_mix_xor, sbox_xor, shift_rows_inv, sbox_mix_xor_inv, sbox_xor_inv*. Dzięki zastosowaniu koprocesora (Rysunek 48) uzyskano przyspieszenie x5,5 względem rozwiązania programowego.



Rysunek 48. Jednostka wektorowa SIMD koprocesora [133]

W pracy [29] do procesora ARM7 zaprojektowano koprocesor z nowymi instrukcjami do przyspieszania algorytmu AES. Dodano instrukcje SBox (nieliniowego podstawienia bajtu) oraz SMix (nieliniowe podstawienie bajtu i operacja MixColumn). Dzięki zastosowaniu koprocesora uzyskano przyspieszenie x2,54.

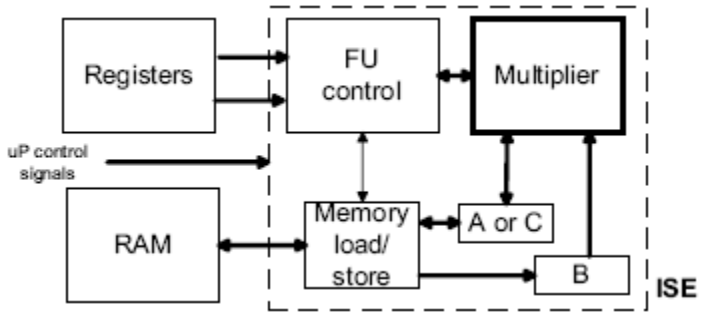
W pracy [221] przedstawiono projekt procesora ASIP opartego o procesor MIPS z rozszerzoną listą rozkazów do realizacji algorytmu Blowfish. ASIP zaprojektowano w środowisku LISATek Processor Designer (Rysunek 49) i zrealizowano w układzie ASIC 0.18 µm. Zaproponowano dwie architektury ASIP: a) SBox z dodawaniem, b) SBox i oddzielne dodawanie. W celu redukcji zużycia zasobów usunięto nieużywane instrukcje i zmniejszono liczbę rejestrów z 15 do 9. W wyniku uzyskano redukcję kodu o 56% i 50%, przyspieszenie wykonania x7,8 i x5,2 oraz redukcję zużytych bramek o 30%.



Rysunek 49. Symulacja procesora ASIP w LISA simulator [221]

9.3.2. Koprocesor dla algorytmów asymetrycznych

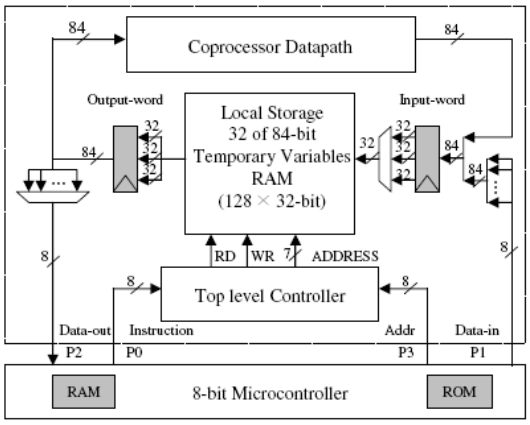
W pracy [155] w układzie Atmel FPSLIC do 8-bitowego procesora Atmel AVR zaprojektowano koprocesor FPGA z nowymi instrukcjami do przyspieszania wykonania algorytmu HECC 163-bit, co ilustruje Rysunek 50. Oprócz dodania sprzętowego mnożnika zwiększono znacznie liczbę rejestrów. Dzięki zastosowaniu koprocesora zrealizowanego na 250-500 CLBs osiągnięto przyspieszenie x30 względem rozwiązań programowych.



Rysunek 50. Struktura i interfejs koprocesora FPGA do algorytmu HECC [155]

W pracy [80] do 8-bitowego procesora Atmel AVR zaprojektowano koprocesor z nowymi instrukcjami do przyspieszania algorytmu ECC 163-bit i 233-bit oraz RSA 1024-bit. Dodano instrukcje mnożenia 8-bitowego w ciele GF(2^m) MULX oraz mnożenia z akumulacją i przeniesieniem MULACCX. Osiągnięto przyspieszenie x14 dla algorytmu ECC.

W pracy [118] do 8-bitowych procesorów Atmel AVR i Intel 8051 zaprojektowano koprocesor z nowymi instrukcjami do przyspieszania wykonania algorytmu HECC 83-bit (rozwinięcie prac [31][216]). Zastosowano ściśle zintegrowany pełen akcelerator GF z mikrokodowanymi instrukcjami, co przedstawia Rysunek 51. Dodano instrukcje transferu danych, mnożenia i dodawania w GF(2^{83}). Symulacje wykonano za pomocą symulatora ISS i środowiska GEZEL. Dzięki zastosowaniu koprocesora osiągnięto przyspieszenie x228 dla 8051 i x106 dla AVR względem rozwiązań programowych.



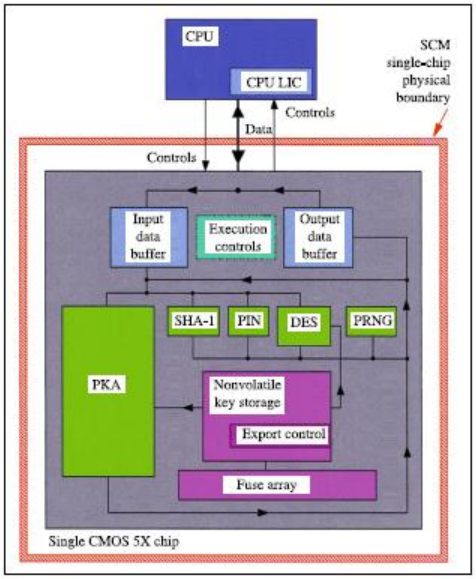
Rysunek 51. Diagram blokowy akceleratora GF do algorytmu HECC [31][118]

W pracy [212] zaprojektowano koprocesor z instrukcjami do przyspieszania wykonania algorytmów asymetrycznych. Dodano instrukcje na liczbach całkowitych wielokrotnej precyzji (ADD, SUB) oraz do mnożenia (MULM) i potęgowania modularnego (EXPM, EXPS). Komunikacja procesora z koprocesorem odbywa się przez rejestr statusu i rejestr rozkazów, a dane dla koprocesora znajdują się w pamięci. Koprocesor zrealizowano jako układ ASIC 0.6 µm osiągając 3-krotne przyspieszenie względem innych koprocesorów.

W pracy [79] do procesora w systemie S/390 zaprojektowano koprocesor do wspomagania wykonywania wszystkich funkcji kryptograficznych. Rysunek 52 przedstawia koprocesor wykonujący dwa zestawy operacji:

- DAC – wykonywane synchronicznie do CPU, realizujące protokół ICSF,
- CAP – wykonywane asynchronicznie do CPU, wspierające protokół PKSC.

Koprocesor zrealizowano jako układ ASIC CMOS 0.5 µm, spełniający wymagania bezpieczeństwa FIPS 140-1 na poziomie 4.

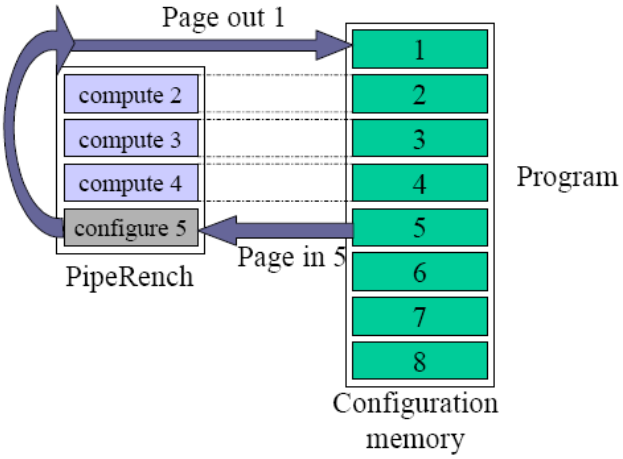


Rysunek 52. Schemat przepływu danych koprocesora w systemie S/390 [79]

9.4. Procesor z rekonfigurowalnym koprocesorem

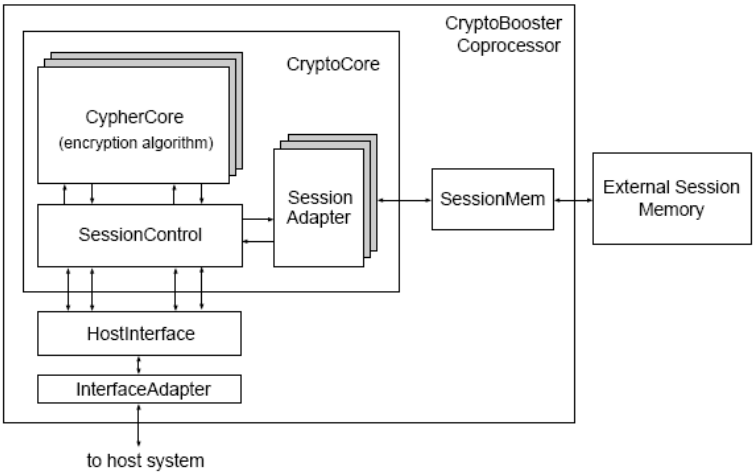
W pracy [255] zaprojektowano rekonfigurowalny potokowy koprocesor PipeRench do przyspieszania algorytmów symetrycznych. PipeRench wykorzystuje 32-bitowe potoki przetwarzania danych, wirtualizując sprzęt poprzez jego multipleksowanie w czasie, co ilustruje Rysunek 53. Koprocesor jest blokiem funkcjonalnym CPU

mającym dostęp do zbioru rejestrów oraz pamięci cache i zawiera programowalną strukturę układów przetwarzających PE. Dzięki potokowej architekturze PipeRench uzyskano znaczne przyspieszenie wykonania algorytmów IDEA, RC6 i Twofish.



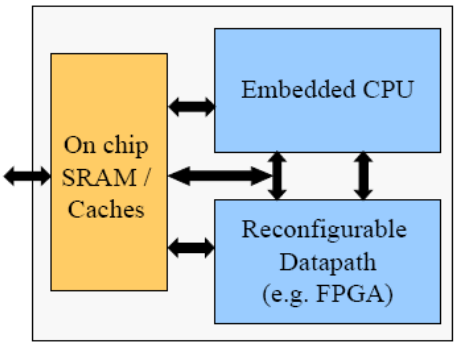
Rysunek 53. Wirtualizacja sprzętu w koprocesorze PipeRench [255]

W pracy [187] zaprojektowano rekonfigurowalny koprocesor CryptoBooster do przyspieszania wykonania algorytmu IDEA. Zastosowano częściową rekonfigurację układu FPGA z biblioteki bitstreamów konfiguracyjnych (Rysunek 54). Komunikacja hosta z koprocesorem odbywa się za pomocą rejestrów wejściowo-wyjściowych oraz linii przerwań.



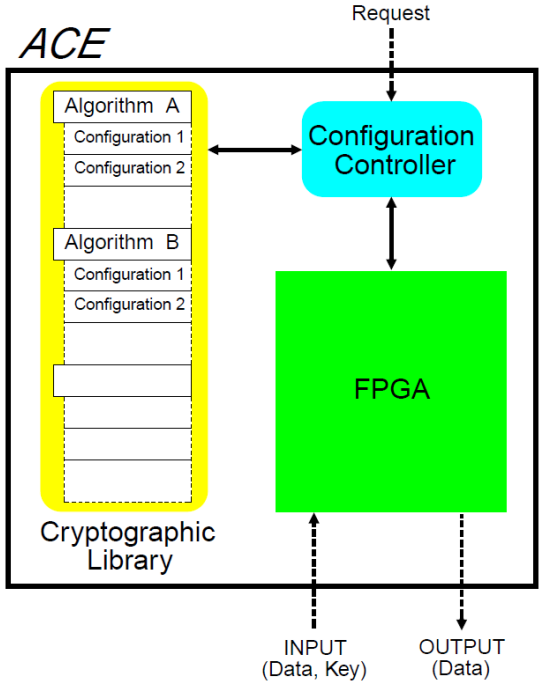
Rysunek 54. Diagram blokowy architektury CryptoBooster [187]

W pracy [166] zaprojektowano rekonfigurowalny koprocesor ściśle dołączony do CPU, co ilustruje Rysunek 55. Zaprojektowano także Nimble Compiler do kompilacji specyfikacji systemowej i algorytm podziału HW/SW wykonujący podział na drobnym stopniu granulacji, aby w pełni wykorzystać równoległość na poziomie instrukcji.



Rysunek 55. Architektura koprocesora ściśle dołączonego do CPU [166]

W pracy [69] zaprezentowano architekturę rekonfigurowalną ACE zastosowaną w IPSec. Architektura ACE adaptuje się do zmieniających się parametrów protokołu dając wyższe przepustowości względem rozwiązań programowych. Zaimplementowano algorytmy rundy finałowej AES w układzie Xilinx Virtex. ACE przechowuje w pamięci w formie biblioteki kryptograficznej skompresowane bitstreamy do programowania układów FPGA, co obrazuje Rysunek 56.



Rysunek 56. Architektura rekonfigurowalna ACE [69]

9.5. Podsumowanie

Architektura realizacji algorytmu kryptograficznego powinna być dostosowana do wymagań aplikacji. Dedykowany akcelerator jest najczęściej stosowany w systemach wbudowanych, które realizują jeden konkretny algorytm szyfrowania. Koprocesor jest rozwiązaniem generycznym do przyspieszania wielu algorytmów szyfrujących, używanym w komputerach dużej mocy obliczeniowej. Koprocesor rekonfigurowalny jest stosowany w systemach wbudowanych o ograniczonych zasobach, które mogą obsługiwać jeden z wielu algorytmów szyfrujących.