

POLITECHNIKA ŚLĄSKA
WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI
INSTYTUT ELEKTRONIKI

Kody LDPC efektywnie dekodowane w strukturach programowalnych

ROZPRAWA DOKTORSKA

AUTOR:
mgr inż. Wojciech Sułek

PROMOTOR:
dr hab. inż. Dariusz Kania, prof. Pol. Śl.

Gliwice 2008

Serdeczne podziękowania kieruję do wszystkich, którzy w jakikolwiek sposób przyczynili się do powstania niniejszej pracy. Szczególnie dziękuję prof. Dariuszowi Kani za nieocenioną pomoc w napisaniu rozprawy, jak również Kolegom z Zakładu Telekomunikacji, którzy niejednokrotnie służyli mi poradą. Serdecznie dziękuję też moim rodzicom oraz przyjaciołom, którzy nieustannie motywowali mnie i wspierali w tym czasie.

Spis treści

Wykaz oznaczeń	v
Wykaz skrótów	x
1. Wstęp	1
1.1 Geneza pracy	1
1.2 Podstawowe definicje	2
2. Algorytmy kodowania i dekodowania kodów LDPC	11
2.1 Tworzenie słów kodowych	12
2.2 Iteracyjne algorytmy dekodowania	16
2.2.1 Algorytm BP	16
2.2.2 Modyfikacje dekodowania BP	22
2.2.3 Aproksymowane reprezentacje dekodowania BP	27
2.2.4 Algorytmy z twardymi decyzjami	30
2.3 Podsumowanie	30
3. Teza i cele pracy	31
3.1 Układ pracy	32
4. Implementacja dekodera	34
4.1 Klasyfikacja architektur dekodera	36
4.2 Definicja kodu AA-LDPC	38
4.3 Przegląd implementacji dekodera znanych z literatury	39
4.4 Algorytm TDMP	42
4.4.1 Dekodowanie z zastosowaniem harmonogramu szeregowego	43
4.4.2 Dekodowanie z zastosowaniem algorytmu TDMP	46
4.5 Format wartości wiadomości w dekodерze	48
4.6 Struktura dekodera	49
4.6.1 Pamięć Q	53

4.6.2	Jednostka próbných decyzji	53
4.6.3	Jednostka SISO	55
4.6.4	Analiza błędów obciążenia wiadomości	67
4.6.5	Konfigurowalne sieci zapisu i odczytu	73
4.6.6	Optymalizacja przetwarzania potokowego	74
4.7	Środowisko symulacji systemów kodowania LDPC	79
4.8	Analiza porównawcza algorytmów dekodowania	81
4.9	Wyniki implementacji	85
4.10	Podsumowanie	91
5.	<i>Algorytmy tworzenia kodów AA-LDPC</i>	93
5.1	Wyznaczenie dystrybucji stopni wierzchołków grafu Tannera	95
5.1.1	Ewolucja gęstości prawdopodobieństwa	96
5.1.2	Optymalizacja dystrybucji stopni wierzchołków	97
5.2	Definicje pojęć związanych z grafem Tannera	101
5.3	Metody generacji kodów LDPC	106
5.3.1	Algorytm PEG	107
5.4	Algorytm tworzenia macierzy bazowej E-PEG	109
5.5	Ekspansja macierzy bazowej kodu	110
5.5.1	Definicja ekspansji grafu bazowego	111
5.5.2	Warunek usunięcia cyklu na etapie ekspansji grafu	112
5.5.3	Warunek usunięcia zbioru \mathcal{AS} na etapie ekspansji grafu	113
5.5.4	Wyszukiwanie cykli oraz zbiorów \mathcal{AS} w grafie bazowym	115
5.5.5	Algorytmy ekspansji macierzy bazowej	120
5.5.6	Weryfikacja skuteczności algorytmów ekspansji	129
5.6	Kompleksowy algorytm tworzenia kodu AA-LDPC	133
5.7	Własności korekcyjne uzyskanych kodów	135
5.8	Podsumowanie	139
6.	<i>Podsumowanie</i>	141
	<i>Bibliografia</i>	146

Wykaz oznaczeń

$ \mathcal{A} $	moc zbioru \mathcal{A}
\mathbb{R}	zbiór liczb rzeczywistych
\mathbf{u}	wektor informacyjny (słowo informacyjne)
\mathbf{x}	wektor kodowy (słowo kodowe)
\mathbf{y}	wektor odebrany
\mathcal{C}	liniowy kod blokowy
\mathbf{G}	macierz generująca kodu liniowego
\mathbf{H}	macierz kontrolna kodu liniowego
K	liczba bitów w słowie informacyjnym
N	liczba bitów w słowie kodowym
M	liczba równań kontrolnych kodu
σ^2	wariancja gaussowskiego procesu losowego skojarzonego z białym szumem
$\mathcal{M}(n)$	zbiór numerów wierszy macierzy \mathbf{H} zawierających jedynkę w kolumnie n -tej
$\mathcal{N}(m)$	zbiór numerów kolumn macierzy \mathbf{H} zawierających jedynkę w wierszu m -tym
\mathcal{G}	graf Tannera kodu
\mathcal{V}_b	zbiór wierzchołków bitowych grafu
b_n	n -ty wierzchołek bitowy

\mathcal{V}_c	zbiór wierzchołków kontrolnych grafu
c_m	m -ty wierzchołek kontrolny
\mathcal{E}	zbiór krawędzi grafu
e_i	krawędź grafu
d_b	waga kolumn macierzy kontrolnej kodu regularnego (stopień wierzchołków bitowych grafu Tannera)
d_c	waga wierszy macierzy kontrolnej kodu regularnego (stopień wierzchołków kontrolnych grafu Tannera)
\mathbf{d}_b	dystrybucja stopni wierzchołków bitowych grafu Tannera kodu
\mathbf{d}_c	dystrybucja stopni wierzchołków kontrolnych grafu Tannera kodu
$d_{b_{max}}$	maksymalny stopień wierzchołka bitowego
$d_{c_{max}}$	maksymalny stopień wierzchołka kontrolnego
\mathbf{W}	macierz bazowa kodu AA-LDPC
D	liczba wierszy macierzy bazowej kodu AA-LDPC
L	liczba kolumn macierzy bazowej kodu AA-LDPC
$\mathbf{P}_{d,l}$	kwadratowa podmacierz macierzy kontrolnej kodu AA-LDPC
P	rozmiar podmacierzy $\mathbf{P}_{d,e}$
$\mathbf{0}_{P \times P}$	macierz zerowa $P \times P$
$\mathbf{I}_{P \times P}$	macierz jednostkowa $P \times P$
$\mathbf{I}_{\theta(p)}$	permutacja macierzy jednostkowej
d_{\min}	minimalna odległość słów kodowych
h	rozmiar luki macierzy kontrolnej w postaci prawie dolnotrójkątnej
i_{\max}	maksymalna liczba iteracji procesu dekodowania
p_n^0, p_n^1	prawdopodobieństwa <i>a priori</i> bitu x_n równego odpowiednio 0 i 1

r_{mn}^0, r_{mn}^1	wiadomości z wierzchołka c_m do b_n w algorytmie BP
q_{nm}^0, q_{nm}^1	wiadomości z wierzchołka b_n do c_m w algorytmie BP
q_n^0, q_n^1	prawdopodobieństwa <i>pseudo posteriori</i> bitu x_n równego odpowiednio 0 i 1
$L(U)$	Funkcja LLR binarnej zmiennej losowej U
R_{mn}	wiadomość z wierzchołka c_m do b_n w algorytmie LLR-BP
Q_{nm}	wiadomość z wierzchołka b_n do c_m w algorytmie LLR-BP
Q_n	LLR prawdopodobieństw <i>pseudo posteriori</i> dla bitu x_n
A	stała normalizująca w algorytmie <i>Normalized-Min-Sum</i>
C	stała korygująca w algorytmie <i>Offset-Min-Sum</i>
\mathcal{C}^d	d -ty super-kod kodu \mathcal{C}
\mathbf{H}^d	d -ta podmacierz macierzy \mathbf{H}
$\mathcal{N}(d, p)$	zbiór indeksów wierzchołków bitowych sąsiadujących z wierzchołkiem p -tym super-kodu \mathcal{C}^d
$\mathcal{DP}(n)$	zbiór par $\{d, p : n \in \mathcal{N}(d, p)\}$
$\lambda_n^{d,p}$	wiadomość z wierzchołka p -tego super-kodu \mathcal{C}^d do b_n w algorytmie TDMP
Λ_n^p	wartość wiadomości $\lambda_n^{d,p}$ uaktualniona w aktualnej sub-iteracji
ρ_n^p	wiadomość z wierzchołka b_n do wierzchołka p -tego super-kodu dekodowanego w aktualnej sub-iteracji algorytmu TDMP
B	długość stałoprzecinkowego słowa wiadomości
Δ	krok kwantyzacji wiadomości
Z_{th}	próg obciążenia wartości wiadomości
$\mathcal{O}_{B, Z_{th}}$	zbiór wartości wiadomości wyrażonych za pomocą słów stałoprzecinkowych o parametrach B, Z_{th}
T_{init}	liczba cykli inicjalizacji

T_{CNU}	liczba cykli opóźnienia bloku CNU
T_{Subtr^*}	liczba cykli opóźnienia bloku Subtr*
T_{Sum^*}	liczba cykli opóźnienia bloku Sum*
T_{SISO}	liczba cykli opóźnienia bloku SISO
T_{MUX}	liczba cykli opóźnienia konfigurowalnych sieci odczytu / zapisu
T_P	liczba cykli opóźnienia przetwarzania potokowego
$X_{(m_2, m_1)}$	zmienna charakteryzująca zależność pomiędzy wierszami m_1 i m_2 macierzy bazowej, określająca liczbę cykli bezczynności
T_{idle}^m	liczba cykli bezczynności w m -tej sub-iteracji
T_{idle}	sumaryczna liczba cykli bezczynności
\mathbf{W}'	macierz bazowa przekształcona dla celów minimalizacji liczby cykli bezczynności
TH	przepustowość dekodera
$\epsilon_{(Q_n)}$	błąd obcięcia wartości wiadomości Q_n
$\epsilon_n^{(i)}$	sumaryczny błąd obcięcia w iteracji i -tej
Q_{th}	próg normalizacji wiadomości λ
A_λ	stała normalizacji wiadomości λ
\mathbf{p}_{in}	rozkład prawdopodobieństwa wartości wiadomości wejściowych
\mathbf{p}_Q	rozkład prawdopodobieństwa wartości wiadomości Q_{nm}
\mathbf{p}_R	rozkład prawdopodobieństwa wartości wiadomości R_{mn}
$p_{\text{err}}^{(i)}$	prawdopodobieństwo błędnej decyzji w iteracji i -tej
κ	wektor względnej dystrybucji krawędzi incydentnych do wierzchołków bitowych
χ	wektor względnej dystrybucji krawędzi incydentnych do wierzchołków kontrolnych

σ_{th}^2	wartość progowa wariancji dla kanału AWGN
g	obwód grafu
g_{b_n}	obwód grafu w węźle bitowym b_n
\bar{g}	wartość średnia obwodu w węzłach bitowych
$w_H(\mathbf{x})$	waga Hamminga wektora \mathbf{x} (liczba niezerowych elementów)
\mathcal{SS}_a	<i>stopping set</i>
$\mathcal{TS}_{a,b}$	<i>trapping set</i>
\mathcal{LD}_a	<i>linearly dependent set</i>
$\mathcal{AS}_{a,b}$	<i>absorbing set</i>
Φ_{b_n}	podzbiór wierzchołków kontrolnych o największej wartości odległości do b_n
θ_e	permutacja skojarzona z krawędzią e
Θ	zbiór permutacji θ_e
\mathcal{P}	zbiór P -elementowy
$\mathcal{G}^{(\mathcal{P},\Theta)}$	graf powstały w wyniku ekspansji grafu \mathcal{G}
\mathcal{S}_2	zbiór kombinacji wierzchołków bitowych
LIST_{cyc}	lista cykli
$\text{LIST}_{\mathcal{AS}}$	lista zbiorów \mathcal{AS}
s_e	wartość przesunięcia cyklicznego skojarzonego z krawędzią $e \in \mathcal{E}$
\mathbf{S}	wektor wartości s_e
CON_{e_i}	macierz ograniczeń wartości \mathbf{S} zawierających element s_{e_i}

Wykaz skrótów

AA-LDPC	Architecture Aware LDPC
ACE	Approximate Cycle EMD
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BIAWGN	Binary Input Additive White Gaussian Noise
BF	Bit Flipping
BNU	Bit Node Unit
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CNU	Check Node Unit
DE	Density Evolution
EMD	Extrinsic Message Degree
FER	Frame Error Rate
FPGA	Field Programmable Gate Array
GF	Galois Field
LD	Likelihood Difference
LDPC	Low-Density Parity-Check

LIFO	Last-In First-Out
LLR	Log-Likelihood Ratio
LR	Likelihood Ratio
LSB	Least Significant Bit
MAP	Maximum A-Posteriori
MLD	Maximum Likelihood Decoder
MSB	Most Significant Bit
PEG	Progressive Edge Growth
SLLD	Signed Log-Likelihood Difference
SNR	Signal to Noise Ratio
SPA	Sum-Product Algorithm
SPC	Single Parity Check
TDMP	Turbo Decoding Message Passing
TG	Tanner Graph
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

1. Wstęp

Rola szybkiego dostępu do informacji w postępie współczesnego świata jest nie do przecenienia. Jednym z podstawowych czynników napędzających rozwój możliwości wymiany informacji jest nieustanny progres szybkości i niezawodności systemów transmisji danych cyfrowych. Dla podtrzymania tego rozwoju konieczne jest ciągle prowadzenie prac badawczych ukierunkowanych na ulepszanie technologii transmisji danych.

Jednym z podstawowych elementów determinujących możliwości systemu transmisji jest sposób kodowania przesyłanej informacji, który umożliwia odpowiednią kompresję danych (kodowanie źródłowe) oraz detekcję i korekcję błędów (kodowanie kanałowe). Wzrost możliwości obliczeniowych współczesnych układów cyfrowych wielkiej skali integracji umożliwia wykorzystanie systemów kodowania o dużych wymaganiach obliczeniowych, niedostępnych do praktycznego zastosowania jeszcze jakiś czas temu. Niniejsza praca dotyczy jednego z najnowocześniejszych sposobów kodowania pozwalającego na korekcję błędów – kodowania LDPC (*Low-Density Parity-Check*). W szczególności praca jest ukierunkowana na rozwiązanie problemów wiążących się ze skutecznym i szybkim dekodowaniem kodów LDPC z wykorzystaniem układów programowalnych, przedstawienie efektywnej implementacji sprzętowego modułu dekodera oraz zaproponowanie i eksperymentalne zweryfikowanie metod tworzenia takich kodów, które są efektywnie dekodowane sprzętowo, a jednocześnie posiadają jak najlepsze własności korekcyjne.

1.1 Geneza pracy

Opisane w 1962r. przez Gallagera [30, 31], zapomniane przez lata i „ponownie odkryte” pod koniec lat dziewięćdziesiątych przez D.J.C. MacKaya [61] kody LDPC pozwalają osiągać dowolnie małe prawdopodobieństwo błędu (przekłamania transmitowanego bitu), przy prędkości transmisji bliskiej przepływności kanału. Pod tym względem stanowią jedną z najlepszych znanych klas kodów do korekcji błędów (ECC – *Error Correcting Codes*). Algorytmy kodowania, jak i (w szczególności) dekodowania kodów LDPC są dosyć złożone koncepcyjnie i obliczeniowo, co przez lata

ograniczało możliwości ich praktycznego wykorzystania. Aktualne zainteresowanie badaczy rozwojem metod kodowania LDPC jest spowodowane w dużej mierze wzrostem mocy obliczeniowej układów cyfrowych. Zalety tych kodów zostały docenione ostatnimi czasy przez organizacje standaryzacyjne – są one wdrażane między innymi do zastosowania w transmisji sygnału telewizji cyfrowej DVB-S2 [28], bezprzewodowych sieciach lokalnych WLAN [40] oraz sieciach WiMAX [41].

Kod LDPC charakteryzowany jest przez macierz kontrolną kodu, która jest macierzą rzadką. W celu osiągnięcia niskiej bitowej stopy błędów systemu, kody definiowane są dla dużych bloków danych (typowo wiele tysięcy bitów), stąd macierz kontrolna ma również duży rozmiar. Dla osiągnięcia dobrych własności korekcyjnych kodu konieczne jest odpowiednie skonstruowanie macierzy. Sposoby tworzenia macierzy kontrolnych definiujących kody o dobrych własnościach są zatem zagadnieniem nietrywialnym.

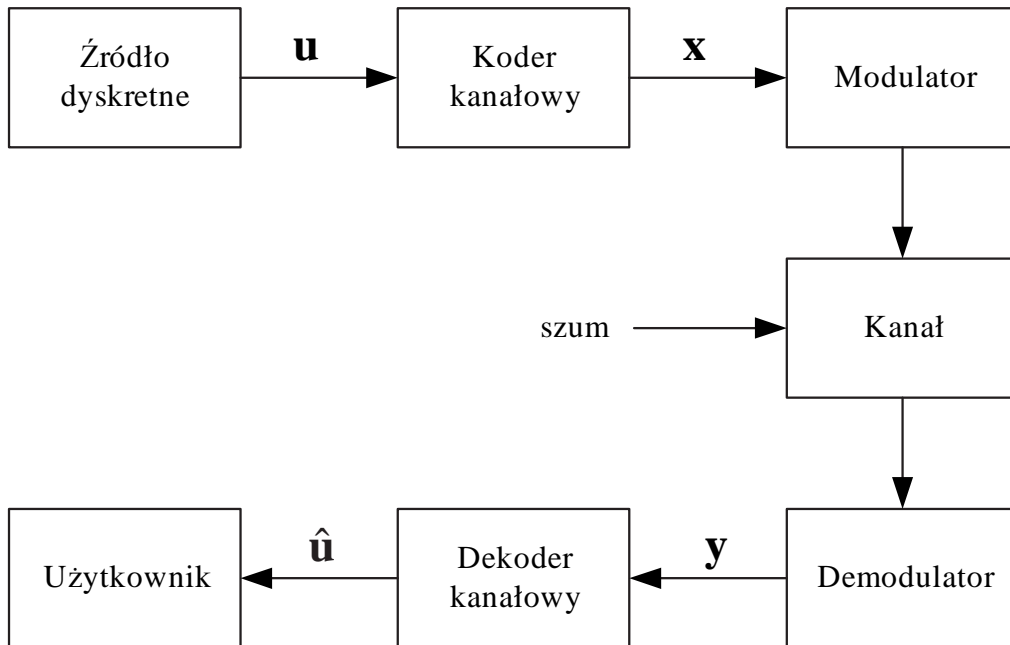
Nietrywialne jest również zadanie projektowania efektywnych modułów kodera i dekodera sprzętowego, zapewniających odpowiednią przepustowość przy ograniczonych zasobach sprzętowych. Efektywne implementacje wykorzystują pewną podklasę kodów LDPC — kody o macierzy kontrolnej w postaci blokowej. Zdecydowana większość znanych publikacji dotyczących tworzenia „dobrych” kodów LDPC nie uwzględnia ograniczeń implementacyjnych narzucanych na strukturę macierzy kontrolnej. Zagadnienie generacji macierzy kontrolnych z owymi ograniczeniami przy jednoczesnej minimalizacji bitowej stopy błędów systemu jest ciągle nie do końca rozpoznane.

Podjęto zatem zadanie opracowania specyfikacji w języku opisu sprzętu modułu uniwersalnego dekodera kodów LDPC, opracowania metod generacji „dobrych” kodów efektywnie dekodowanych sprzętowo oraz eksperymentalnego zweryfikowania własności uzyskanych kodów za pomocą środowiska symulacyjnego wykorzystującego dekodery zaimplementowany w układzie programowalnym.

1.2 Podstawowe definicje

W celu zwiększenia prawdopodobieństwa poprawnej transmisji informacji przez nieidealne kanały telekomunikacyjne stosuje się **kodowanie kanałowe** z redundancją przesyłanej informacji. Stosowane w tym celu kody nazywane są **kodami nadmiarowymi**, a nadmiar zostaje wykorzystany do detekcji i korekcji błędów. Schemat blokowy modelu takiego systemu transmisji przedstawiono na rys. 1.1 [33]. Koder przekształca sekwencję słów informacyjnych \mathbf{u} w sekwencję słów kodowych

\mathbf{x} . Elementy słowa (wektora) kodowego \mathbf{x} oraz słowa (wektora) informacyjnego \mathbf{u} przyjmują wartości z pewnego zbioru (alfabetu), przy czym w niniejszej pracy rozpatrywane są jedynie kody binarne, dla których alfabet ma postać $\{0, 1\}$. Modulator ma za zadanie przekształcenie sekwencji słów kodowych na sygnał odpowiedni dla danego kanału. Odwrotne operacje wykonywane są w demodulatorze i dekodерze, a ich wynikiem jest sekwencja słów $\hat{\mathbf{u}}$, która powinna odzwierciedlać nadaną sekwencję słów informacyjnych z jak najmniejszą stopą błędów. Elementy słowa \mathbf{y} dostarczanego do dekodera mogą mieć postać binarną $\{0, 1\}$ – wtedy mówi się o demodulatorze (dekoderze) z tzw. twardymi decyzjami, lub liczbową oznaczającą prawdopodobieństwa *a priori* dla wartości pojedynczego bitu – wtedy mówi się o demodulatorze (dekoderze) z tzw. miękkimi decyzjami.



Rys. 1.1: Uproszczony model systemu transmisyjnego

Jednym z podstawowych parametrów kodu jest **stopa kodu** R (*code rate*), nazywana też sprawnością [15], a wyznaczana zgodnie z zależnością:

$$R = \frac{K}{N} \quad (1.1)$$

gdzie N oznacza liczbę bitów przesyłanych przez kanał, natomiast K – liczbę bitów informacyjnych. Różnica między wartościami N i K to liczba bitów nadmiarowych oznaczana jako M .

Wśród kodów nadmiarowych wyróżnia się dwa rodzaje: **kody blokowe** i **splotowe**. Dla kodu blokowego istnieje jednoznaczne przyporządkowanie bloku K symboli (bitów) informacyjnych dostarczanych sekwencyjnie do kodera $\mathbf{u} = [u_1, u_2, \dots, u_K]$ do bloku N bitów słowa kodowego $\mathbf{x} = [x_1, x_2, \dots, x_N]$. Kolejne słowa informacyjne \mathbf{u} są przekształcane w słowa kodowe \mathbf{x} niezależnie od siebie. W przypadku kodów splotowych słowo kodowe powstaje na podstawie pewnej liczby słów informacyjnych, koder posiada zatem pamięć historii słów informacyjnych. W niniejszej pracy kody splotowe nie będą rozpatrywane.

Szczególne znaczenie mają **binarne liniowe kody blokowe**, których formalna definicja jest następująca [57]: kod blokowy \mathcal{C} o długości słowa N i liczbie słów kodowych 2^K jest binarnym liniowym kodem blokowym wtedy i tylko wtedy, gdy 2^K wektorów kodowych stanowi K -wymiarową podprzestrzeń N -wymiarowej przestrzeni wektorowej nad dwuelementowym ciałem skończonym (ciałem Galois) $GF(2)$.

Kod \mathcal{C} o parametrach (N, K) stanowi zatem pewien zbiór N -elementowych wektorów binarnych o liczności 2^K . Dowolny wektor N -elementowy \mathbf{x} jest poprawnym słowem kodowym, jeśli $\mathbf{x} \in \mathcal{C}$. Kod jest liniowy, jeśli suma (w ciele $GF(2)$) dwóch słów (wektorów) kodowych daje w wyniku słowo należące do kodu. Kody o parametrach N, K są skrótowo nazywane kodami (N, K) , na przykład kod $(1024, 512)$.

Istotnym parametrem, determinującym możliwości korekcyjne kodu, jest **minimalna odległość** d_{\min} . Jest to najmniejsza odległość Hamminga między dwoma dowolnymi wektorami kodowymi.

Z definicji liniowego kodu blokowego wynika, że możliwe jest wyodrębnienie K liniowo niezależnych wektorów kodowych $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_K$, a każdy wektor $\mathbf{x} \in \mathcal{C}$ jest liniową kombinacją tych wektorów. Przedstawia to zależność (1.2).

$$\mathbf{x} = \mathbf{u} \cdot \mathbf{G}, \quad \mathbf{G} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_K \end{bmatrix}_{K \times N} \quad (1.2)$$

Macierz \mathbf{G} nazywana jest macierzą generującą kodu.

Z drugiej strony, istnieje $M = N - K$ liniowo niezależnych wektorów $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_M$ generujących podprzestrzeń uzupełniającą (*null-space*) podprzestrzeni generowanej przez $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_K$, które są ortogonalne do wszystkich słów kodowych:

$$\mathbf{H} \cdot \mathbf{x}^T = \mathbf{0}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_M \end{bmatrix}_{M \times N} \quad (1.3)$$

Powyższe równanie to równanie kontrolne kodu, a macierz \mathbf{H} zwana jest macierzą parzystości lub macierzą kontrolną kodu (*parity-check matrix*). N -elementowy wektor \mathbf{x} należy do kodu \mathcal{C} o macierzy kontrolnej \mathbf{H} wtedy i tylko wtedy, gdy spełnione jest równanie (1.3), składające się z M równań kontroli parzystości. Wynik mnożenia $\mathbf{H} \cdot \mathbf{x}^T$ nazywany jest syndromem błędu (jeśli syndrom jest wektorem zerowym, to równanie kontrolne jest spełnione).

Zadaniem dekodera (rys. 1.1) jest wyznaczenie nadanego słowa informacyjnego \mathbf{u} (co ze względu na wzajemną jednoznaczność procesu kodowania jest równoważne wyznaczeniu słowa kodowego \mathbf{x}) na podstawie odebranego ciągu \mathbf{y} . Reguła decyzyjna dekodera powinna być skonstruowana w taki sposób, aby wartość wyjściowa $\hat{\mathbf{x}}$ była słowem kodowym \mathbf{x} , dla którego prawdopodobieństwo *a posteriori* $P(\mathbf{x}|\mathbf{y}, \mathbf{H})$ jest maksymalne. Dekoder działający zgodnie z tak zdefiniowaną regułą jest nazywany **dekoderem MAP** (*Maximum A-Posteriori Probability*, [57]). Prawdopodobieństwo *a posteriori* można wyznaczyć zgodnie z regułą Bayesa:

$$P(\mathbf{x}|\mathbf{y}, \mathbf{H}) = \frac{P(\mathbf{y}|\mathbf{x}, \mathbf{H}) P(\mathbf{x}|\mathbf{H})}{P(\mathbf{y}|\mathbf{H})} \quad (1.4)$$

Jeśli wszystkie słowa kodowe są jednakowo prawdopodobne, maksymalizacja (1.4) jest równoważna z maksymalizacją $P(\mathbf{y}|\mathbf{x}, \mathbf{H})$. Przy założeniu kanału bez pamięci prawdopodobieństwo $P(\mathbf{y}|\mathbf{x}, \mathbf{H})$ jest równe iloczynowi prawdopodobieństw poszczególnych bitów:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{H}) = \prod_i P(y_i|x_i, \mathbf{H}) \quad (1.5)$$

Spośród wszystkich możliwych słów kodowych \mathbf{x} jako najbardziej prawdopodobne wybierane jest zatem słowo, dla którego iloczyn (1.5) przyjmuje wartość maksymalną. Taki dekodek zwany jest **dekoderem maksymalnej wiarygodności** (MLD – *Maximum Likelihood Decoder*).

W ogólności dekodowanie MLD (MAP) liniowych kodów blokowych jest problemem NP-zupełnym [3, 60]. Liczność zbioru możliwych słów kodowych, który należy przeszukać, zależy wykładniczo od długości słowa kodowego N . Praktyczna realizacja dekodera MLD jest możliwa zatem tylko dla kodów o bardzo małych długościach słowa. Algorytmy dekodowania przedstawione w dalszej części pracy, możliwe do praktycznego zastosowania dla dowolnych długości bloków kodu, są suboptymalne, tzn. realizują dekodowanie MLD z pewnym przybliżeniem.

Dobre własności korekcyjne kodu można uzyskać dla dużych długości bloku N , co zostało wykazane przez Claude'a E. Shannona. W swoich pracach z połowy XX wieku [89, 90] określił on fundamentalne **granice możliwości transmisji** danych.

Zdefiniował pojemność informacyjną kanału C_{\max} i udowodnił, że możliwe jest skonstruowanie kodu o stopie $R < C_{\max}$, dla którego przy zastosowaniu dekodera MAP można uzyskać dowolnie małe prawdopodobieństwo błędu P_e :

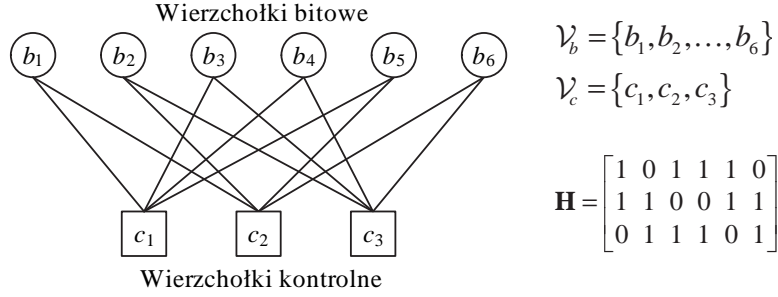
$$P_e \leq 2^{-NE_b(R)} \quad (1.6)$$

gdzie $E_b(R)$ jest funkcją określającą własności danego kanału, która przyjmuje wartości dodatnie dla $R < C_{\max}$. Istotnym wnioskiem płynącym z (1.6) jest fakt, że małe prawdopodobieństwa błędów uzyskuje się stosując kody o odpowiednio dużych długościach bloku N . Shannon nie pokazał jednak, jak skonstruować kod o dobrych własnościach.

Spośród znanych obecnie klas kodów blokowych, znakomite własności korekcyjne i dużą elastyczność doboru parametrów (długości bloku N i stopy R) mają **kody LDPC** (*Low Density Parity Check*). Odpowiednio utworzony kod LDPC o bardzo dużej długości bloku pozwala uzyskać niską bitową stopę błędów przy stosunku sygnału do szumu w kanale bliskim teoretycznej granicy wynikającej z twierdzeń Shannona [16]. Również kody o mniejszych długościach bloków cechują się znakomitymi własnościami korekcyjnymi.

Regularny kod LDPC to zbiór wektorów należących do podprzestrzeni uzupełniającej podprzestrzeń generowaną przez wiersze macierzy kontrolnej \mathbf{H} o rozmiarze $M \times N$, posiadającej następujące właściwości: (1) w każdym wierszu znajduje się d_c jedynek; (2) w każdej kolumnie znajduje się d_b jedynek; (3) macierz kontrolna jest macierzą rzadką – czyli $d_c \ll N$, $d_b \ll M$ ([30, 57, 60]). Liczba jedynek w kolumnie (wierszu) macierzy jest nazywana wagą kolumny (wiersza). Klasa (*ensemble*) regularnych kodów LDPC spełniających powyższe założenia oznaczana będzie przez $\mathcal{C}^{d_b, d_c}(N, K)$, gdzie $K = N - M$ to długość słowa informacyjnego.

Alternatywnym do macierzy kontrolnej sposobem reprezentacji kodu jest tzw. **graf Tannera** (*Tanner Graph* – TG, [105], rys. 1.2). Dla liniowego kodu blokowego o macierzy kontrolnej \mathbf{H} , graf Tannera tego kodu to graf dwudzielny, w którym jeden podzbiór wierzchołków odpowiada równaniom kontrolnym kodu (wierszom macierzy \mathbf{H}), a drugi podzbiór – bitom słowa kodowego (kolumnom macierzy \mathbf{H}). Wierzchołki te nazywane są odpowiednio **wierzchołkami kontrolnymi** (przedstawiane na rysunkach w postaci kwadratów) i **wierzchołkami bitowymi** lub symbolowymi (okręgi). Wierzchołek bitowy n jest połączony krawędzią z wierzchołkiem kontrolnym m wtedy i tylko wtedy, gdy bit n -ty występuje w m -tym równaniu kontrolnym (tzn. element h_{mn} macierzy \mathbf{H} jest jedynką). Stopień wierzchołka bitowego równy jest wadze kolumny macierzy \mathbf{H} , a stopień wierzchołka kontrolnego – wadze wiersza macierzy \mathbf{H} .



Rys. 1.2: Graf Tannera kodu regularnego

W pracy stosowane będą następujące oznaczenia: graf Tannera $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ zawiera zbiór wierzchołków $\mathcal{V} = \mathcal{V}_c \cup \mathcal{V}_b$, gdzie $\mathcal{V}_c = \{c_1, c_2, \dots, c_M\}$ stanowi zbiór wierzchołków kontrolnych, a $\mathcal{V}_b = \{b_1, b_2, \dots, b_N\}$ – zbiór wierzchołków bitowych oraz zbiór krawędzi $\mathcal{E} \subseteq V_c \times V_b$. Krawędź $e_i = (b_n, c_m) \in \mathcal{E}$ wtedy i tylko wtedy, gdy $h_{mn} \neq 0$. Dla kodów regularnych, graf Tannera jest (d_b, d_c) -regularny, co oznacza że stopień wszystkich węzłów kontrolnych jest równy d_c , a stopień wszystkich węzłów bitowych jest równy d_b . Przykład pokazany na rys. 1.2 przedstawia graf $(2,4)$ -regularny, tzn. $d_b = 2$, $d_c = 4$.

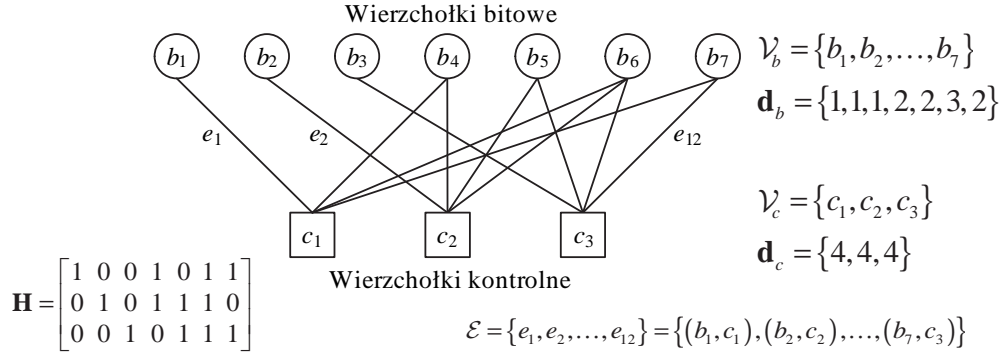
W przypadku **nieregularnych kodów LDPC** [58, 81] wagi wierszy i kolumn macierzy \mathbf{H} mogą być zróżnicowane. Definiuje się tzw. dystrybucję stopni wierzchołków (*degree distribution*) grafu Tannera $(\mathbf{d}_b, \mathbf{d}_c)$:

$$\mathbf{d}_b = \{d_{b_1}, d_{b_2}, \dots, d_{b_N}\} \quad (1.7a)$$

$$\mathbf{d}_c = \{d_{c_1}, d_{c_2}, \dots, d_{c_M}\} \quad (1.7b)$$

Wektor \mathbf{d}_b określa dystrybucję stopni wierzchołków bitowych, gdzie d_{b_n} oznacza stopień wierzchołka b_n (wagę n -tej kolumny \mathbf{H}), natomiast \mathbf{d}_c to dystrybucja stopni wierzchołków kontrolnych, gdzie d_{c_m} oznacza stopień wierzchołka c_m (wagę m -tego wiersza \mathbf{H}). Przykładowy graf nieregularny (odpowiadający kodowi Hamminga $\mathcal{C}(7, 4)$) przedstawiony na rys. 1.3 posiada dystrybucję wierzchołków bitowych $\mathbf{d}_b = \{1, 1, 1, 2, 2, 3, 2\}$ oraz regularną dystrybucję wierzchołków kontrolnych, $d_c = 4$. Kody o odpowiednio dobranej dystrybucji stopni wierzchołków grafu mają zwykle lepsze własności korekcyjne od kodów regularnych [16, 81]. Klasa nieregularnych kodów LDPC o długości słowa kodowego N i długości słowa informacyjnego K oznaczana będzie przez $\mathcal{C}^{\mathbf{d}_b, \mathbf{d}_c}(N, K)$.

Możliwość efektywnej implementacji sprzętowej modułów kodera i dekodera narzuca pewne ograniczenia na strukturę macierzy kontrolnej kodu LDPC. Klasa ko-



Rys. 1.3: Graf Tannera kodu nieregularnego

dów rozpatrywanych w niniejszej pracy, dla której został opracowany konfigurowalny moduł dekodera, nazywana będzie **kodami AA-LDPC** (*Architecture-Aware LDPC*). Kod AA-LDPC to kod o macierzy kontrolnej $\mathbf{H}_{M \times N} = \mathbf{H}_{DP \times LP}$, składającej się z $D \times L$ podmacierzy kwadratowych $\mathbf{P}_{d,l}$, $d = 1, \dots, D$, $l = 1, \dots, L$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \cdots & \mathbf{P}_{1,L} \\ \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \cdots & \mathbf{P}_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{D,1} & \mathbf{P}_{D,2} & \cdots & \mathbf{P}_{D,L} \end{bmatrix} \quad (1.8)$$

gdzie każda z podmacierzy $\mathbf{P}_{d,l}$ o rozmiarze $P \times P$ jest macierzą zerową lub też pewną permutacją kolumn macierzy jednostkowej. Uzasadnienie narzucenia takiej struktury macierzy kontrolnej będzie przedstawione w rozdziale prezentującym implementację sprzętowego dekodera kodów AA-LDPC.

Weryfikacja własności korekcyjnych kodu wymaga założenia odpowiednich modeli modulatora / demodulatora oraz kanału telekomunikacyjnego (rys. 1.1). W niniejszej pracy dla celów symulacji rozpatrywanych systemów kodowania przyjęto model kanału AWGN (*Additive White Gaussian Noise*) oraz modulację binarną BPSK (*Binary Phase Shift Keying*). Ujednolicone modele modulatora / demodulatora oraz kanału umożliwią proste porównywanie własności poszczególnych rozpatrywanych kodów.

Kanał AWGN to kanał bez pamięci, w którym do sygnału użytecznego jest dodawany biały szum gaussowski o gęstości widmowej mocy $N_0/2$ (czyli sygnał odpowiadający gaussowskiemu procesowi losowemu o impulsowej funkcji autokorelacji oraz wariancji σ^2). Modulator, kanał i demodulator traktowane łącznie odpowiadają modelowi kanału o binarnym wejściu \mathbf{x} i ciągłym wyjściu $\mathbf{y} \in \{-\infty, \infty\}$. Dla przy-

padku kanału AWGN oraz modulacji BPSK, w której sygnał transmitowany przez kanał ma jedną z dwóch postaci [116]:

$$\begin{aligned} s_1(t) &= \sqrt{\frac{2E_s}{T}} \cos(2\pi f_0 t), \quad 0 \leq t \leq T \\ s_2(t) &= \sqrt{\frac{2E_s}{T}} \cos(2\pi f_0 t + \pi), \quad 0 \leq t \leq T \end{aligned} \quad (1.9)$$

(gdzie T to czas transmisji jednego bitu, E_s to energia sygnału konieczna do transmisji jednego bitu), połączenie modulatora, kanału AWGN i demodulatora koherentnego można opisać łącznie funkcjami gęstości prawdopodobieństwa:

$$\begin{aligned} p(y|x=0) &= \frac{1}{\sqrt{\pi N_0}} e^{-\frac{y+\sqrt{E_s}}{N_0}} \\ p(y|x=1) &= \frac{1}{\sqrt{\pi N_0}} e^{-\frac{y-\sqrt{E_s}}{N_0}} \end{aligned} \quad (1.10)$$

co pozwala wykazać [78], że rozkład prawdopodobieństw *a priori* wartości bitów nadanych jest następujący:

$$\begin{aligned} P(x_n = 0|y_n) &= 0.5 \operatorname{erfc}\left(-y_n/\sqrt{N_0}\right) \\ P(x_n = 1|y_n) &= 0.5 \operatorname{erfc}\left(y_n/\sqrt{N_0}\right) \end{aligned} \quad (1.11)$$

Dla danego stosunku sygnału do szumu w kanale (*Signal to Noise Ratio*), za pomocą zależności (1.11) uzyskuje się wartości prawdopodobieństw *a priori* $P(x_n|y_n)$, które są wykorzystywane do inicjalizacji dekodera o miękkich decyzjach. Przedstawiony model jest nazywany również kanałem BIAWGN (*Binary Input Additive White Gaussian Noise*).

Sposobem zobrazowania własności systemu transmisji, który będzie stosowany w niniejszej pracy, jest wykres bitowej stopy błędów (blokowej stopy błędów) w funkcji stosunku sygnału do szumu (SNR) w kanale. **Bitowa stopa błędów** BER (*Bit Error Rate*) systemu transmisji to wartość oczekiwana stosunku liczby błędnie zdekodowanych bitów \hat{u}_n do liczby wszystkich zdekodowanych bitów. **Blokowa stopa błędów** FER (*Frame Error Rate*) to wartość oczekiwana stosunku liczby błędnie zdekodowanych bloków $\hat{\mathbf{u}}$ do liczby wszystkich bloków. Stosunek sygnału do szumu w kanale jest równy $\text{SNR} = E_b/N_0$, gdzie E_b to energia sygnału przypadająca na jeden bit informacyjny, tzn. $E_b = E_s/R$.

Wartość BER (FER) systemu zależy od kodu, a ściślej dystrybucji odległości Hamminga słów kodowych (*distance distribution*, [44]), jak również od typu kanału oraz zastosowanego algorytmu dekodowania. Analityczne wyznaczanie wartości

BER (FER) jest zadaniem bardzo złożonym. Już samo obliczenie odległości minimalnej kodu jest problemem NP-trudnym [109]. Stąd też we wszystkich znanych pracach dotyczących systemów kodowania LDPC z suboptymalnymi algorytmami dekodowania, wartości BER (FER) wyznaczone są na drodze symulacji.

Rozprawa doktorska związana jest z kodami LDPC efektywnie dekodowanymi w strukturach programowalnych. Można w niej wyróżnić dwa główne wątki: implementację dekodera sprzętowego oraz opracowanie metod tworzenia macierzy kontrolnych kodów efektywnie dekodowanych w strukturach programowalnych. Elementem łączącym owe zagadnienia są algorytmy iteracyjnego dekodowania kodów LDPC, dlatego przed prezentacją tezy i celów pracy umieszczono przegląd znanych z literatury algorytmów kodowania i dekodowania. Stanowi on podstawę do przedstawienia pomysłów związanych z problemami konstrukcji dekodera oraz tworzenia macierzy kontrolnych kodów efektywnie dekodowanych sprzętowo.

2. Algorytmy kodowania i dekodowania kodów LDPC

Jedną z podstawowych zalet kodów LDPC jest istnienie skutecznych algorytmów dekodowania o złożoności obliczeniowej liniowo zależnej od długości bloku. Dzięki temu możliwe jest wykorzystanie kodów o wielkich długościach bloku, co (jak wspomniano) jest pożądane w celu osiągnięcia dobrych własności korekcyjnych. Złożoność obliczeniowa operacji kodowania, przy zastosowaniu odpowiednio przekształconej macierzy kontrolnej, jest również do zaakceptowania.

W niniejszym rozdziale przedstawiono przegląd algorytmów dekodowania, co stanowi punkt wyjścia do dalszych rozważań, to znaczy prezentacji budowy konfigurowalnego dekodera sprzętowego. Dekoder w zależności od wymagań może realizować jeden z przedstawionych tu algorytmów. Algorytmy o większej skuteczności mają bowiem zwykle większą złożoność obliczeniową, a dekodek wymaga większych zasobów sprzętowych. Wszystkie przedstawione w rozdziale rozważania są znane z literatury, stanowią zatem rozwinięcie rozdziału wstępnego.

Skupienie uwagi na module dekodera wynika z faktu, że operacja dekodowania jest bardziej złożona koncepcyjnie i obliczeniowo. Zadanie opracowania efektywnego dekodera sprzętowego jest znacznie trudniejsze niż kodera, a ograniczenia struktury macierzy kontrolnej kodu wynikają z problemów implementacji dekodera. Oprócz tego, operacja kodowania jest całkowicie jednoznaczna, a znanych (suboptymalnych) algorytmów dekodowania i ich modyfikacji jest wiele – każdy z nich może dawać nieco inne rezultaty jednocześnie wymagając innej liczby zasobów sprzętowych. Dodatkowym uzasadnieniem skupienia uwagi na dekodерze jest znacznie większa przepustowość dekodera implementowanego w strukturze programowalnej w porównaniu z realizacją programową, co umożliwi skonstruowanie szybkiej platformy (wykorzystującej układ FPGA) służącej do eksperymentalnego badania własności kodów LDPC.

Zasadniczą częścią rozdziału jest prezentacja algorytmów dekodowania. Nie można jednak pominąć choć skrótowego przedstawienia zagadnienia kodowania. Macierz

kontrolna umożliwiająca zaproponowanie efektywnej struktury kodera powinna być bowiem wstępnie przekształcona do odpowiedniej formy, poprzez operacje zamiany wierszy i kolumn, a przekształcenie takie powinno być ostatnim etapem procesu tworzenia „dobrego” kodu. Dla kompletności przedstawianych algorytmów tworzenia kodów AA-LDPC konieczne jest zatem uwzględnienie tego przekształcenia w metodach generacji kodów, które zostaną przedstawione w niniejszej pracy.

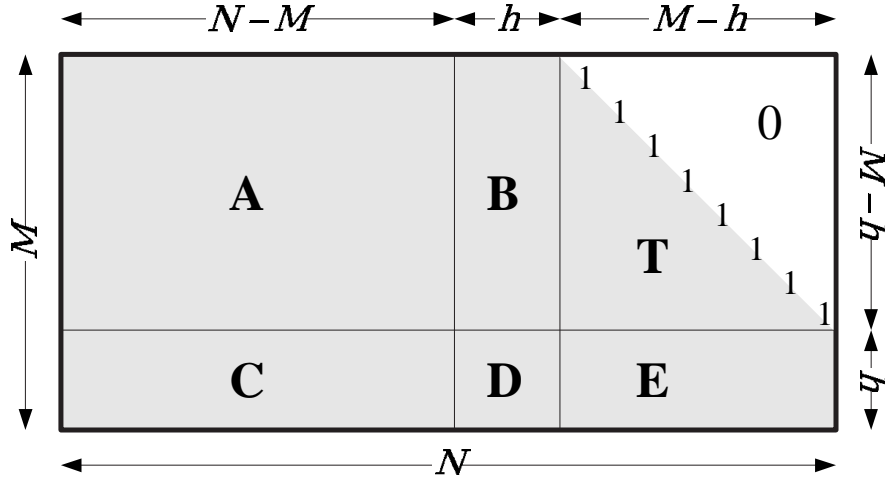
2.1 Tworzenie słów kodowych

Kodowanie to operacja polegająca na wyznaczeniu N -elementowego wektora kodowego na podstawie K -elementowego wektora informacyjnego. Koder pracuje w oparciu o wzajemnie jednoznaczne przyporządkowanie elementów zbioru wektorów informacyjnych do elementów zbioru wektorów kodowych. Możliwości takich przyporządkowań, dla kodu o danej macierzy kontrolnej \mathbf{H} , jest oczywiście bardzo wiele (dokładnie $2^K!$). Na szczególną uwagę zasługuje tzw. przyporządkowanie systematyczne, gdzie wektor kodowy składa się z dwóch części: jedna część jest równa wprost wektorowi informacyjnemu, a druga to część kontrolna dobrana w taki sposób, aby spełnione było równanie kontrolne kodu (1.3). Tak skonstruowany kod nazywany jest kodem systematycznym.

Operacja kodowania może być przeprowadzona za pomocą jednej z dwóch podstawowych metod. Pierwsza metoda polega na skorzystaniu z macierzy generującej kodu. Sposób wyznaczenia macierzy generującej przedstawiono np. w pracy [73]. Wadą tej metody jest duża złożoność obliczeniowa, wynosząca $O(N^2)$, co jest związane z faktem, że w ogólnym przypadku macierz generująca nie jest macierzą rzadką.

Richardson i Urbanke w pracy [82] zaproponowali znacznie bardziej efektywną obliczeniowo metodę, polegającą na wykorzystaniu w procesie kodowania przekształconej macierzy kontrolnej \mathbf{H} . Metoda ta jest wykorzystywana w środowisku symulacyjnym opracowanym dla realizacji celów niniejszej pracy, dlatego wymaga skrótowego przedstawienia.

W algorytmie kodowania wykorzystuje się macierz kontrolną przekształconą do postaci prawie dolnotrójkątnej (rys. 2.1). Przekształcenie to jest wykonywane za pomocą przestawiania wierszy i kolumn macierzy \mathbf{H} , co nie zmienia własności kodu, ponieważ przestawianie kolumn odpowiada zmianie kolejności (permutacji) bitów w wektorze kodowym, natomiast przestawianie wierszy odpowiada zmianie kolejności równań kontrolnych kodu. Wynikiem przekształcenia jest macierz \mathbf{H} składająca się z sześciu podmacierzy:



Rys. 2.1: Macierz kontrolna w formie bliskiej dolnotrójkątnej

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix} \quad (2.1)$$

Rozmiary tych podmacierzy można odczytać z rys. 2.1. Każda z podmacierzy jest macierzą rzadką, a podmacierz \mathbf{T} ma postać dolnotrójkątną z jedynekami na diagonalu.

Heurystyczne metody uzyskiwania przedstawionej postaci można znaleźć w pracy [82]. Istotne jest uzyskanie jak największego rozmiaru podmacierzy dolnotrójkątnej \mathbf{T} (czyli jak najmniejszej wartości h , nazywanej luką). Złożoność obliczeniowa algorytmu kodowania wynosi bowiem $O(N + h^2)$, co zostanie wyjaśnione w dalszej części podrozdziału.

Zakładając kodowanie systematyczne, wektor kodowy \mathbf{x} można przedstawić jako:

$$\mathbf{x} = [\mathbf{u} \ \mathbf{p}_1 \ \mathbf{p}_2] \quad (2.2)$$

gdzie \mathbf{u} oznacza wektor informacyjny o długości $K = N - M$, natomiast \mathbf{p}_1 i \mathbf{p}_2 to poszukiwane wektory kontrolne o długości odpowiednio h i $(M - h)$.

Jeśli obydwie strony równania kontrolnego $\mathbf{H}\mathbf{x}^T = \mathbf{0}$ zostaną pomnożone lewostronnie przez $\begin{bmatrix} \mathbf{I}_{M-h} & \mathbf{0} \\ -\mathbf{E}\mathbf{T}^{-1} & \mathbf{I}_h \end{bmatrix}$, gdzie \mathbf{I}_{M-h} i \mathbf{I}_h są macierzami jednostkowymi o rozmiarach odpowiednio $(M - h)$ i h , to równanie kontrolne przyjmie następującą

postać:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ -\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C} & -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^T \\ \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} = \mathbf{0} \quad (2.3)$$

Równanie (2.3) można przedstawić w postaci dwóch członów:

$$\mathbf{A}\mathbf{u}^T + \mathbf{B}\mathbf{p}_1^T + \mathbf{T}\mathbf{p}_2^T = \mathbf{0} \quad (2.4a)$$

$$(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C})\mathbf{u}^T + (-\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D})\mathbf{p}_1^T = \mathbf{0} \quad (2.4b)$$

Niech $\Phi_{h \times h} = -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}$. Zakładając, że Φ jest macierzą nieosobliwą i przekształcając równanie (2.4b), wektor kontrolny \mathbf{p}_1 można wyrazić jako:

$$\mathbf{p}_1^T = -\Phi^{-1}(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C})\mathbf{u}^T \quad (2.5)$$

Wektor kontrolny \mathbf{p}_2 można natomiast wyznaczyć na podstawie wektorów \mathbf{u} i \mathbf{p}_1 , za pomocą równania (2.4a) przekształconego do postaci:

$$\mathbf{p}_2^T = -\mathbf{T}^{-1}(\mathbf{A}\mathbf{u}^T + \mathbf{B}\mathbf{p}_1^T) \quad (2.6)$$

Jeśli Φ jest macierzą osobliwą, to należy za pomocą przestawiania kolumn doprowadzić macierz $\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ -\mathbf{E}\mathbf{T}^{-1}\mathbf{A} + \mathbf{C} & \Phi \end{bmatrix}$ do takiej postaci, w której podmacierz Φ jest nieosobliwa. Wyznaczenie permutacji kolumn w celu spełnienia tego warunku jest zawsze możliwe, jeśli rząd macierzy \mathbf{H} równy jest liczbie jej wierszy M [82]. Następnie wyznaczoną permutację kolumn należy zastosować do macierzy $\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix}$. W ten sposób uzyskuje się macierz kontrolną \mathbf{H} , dla której Φ jest nieosobliwa. Opisanie operacje przekształcenia wstępnego macierzy \mathbf{H} przedstawiono poniżej jako algorytm 2.1 [82].

Algorytm 2.1: Przekształcenie macierzy kontrolnej dla celów kodowania

Dane wejściowe: macierz kontrolna \mathbf{H}

Dane wyjściowe: macierz kontrolna \mathbf{H} w postaci prawie dolnotrójkątnej jak na rys. 2.1, dla której $\Phi = -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}$ jest macierzą nieosobliwą

- 1 Wykonać permutacje kolumn w celu uzyskania macierzy prawie dolnotrójkątnej $\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} & \mathbf{D} & \mathbf{E} \end{bmatrix}$ o jak najmniejszej wartości luki h
- 2 **JEŻELI** $\Phi = -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}$ jest macierzą osobliwą
- 3 $\left\{ \begin{array}{l} \text{Wykonać dalsze permutacje kolumn do uzyskania postaci, dla której } \Phi \\ \text{jest nieosobliwa} \end{array} \right.$

Algorytm kodowania (algorytm 2.2) składa się z dwóch etapów: wyznaczenia wektora \mathbf{p}_1 oraz wyznaczenia wektora \mathbf{p}_2 . Operacje wykonywane na tych etapach wyszczególniono w tabelach 2.1 i 2.2. Mnożenie przez macierz Φ , która jest macierzą gęstą to operacja o złożoności obliczeniowej $O(h^2)$. Mnożenie przez macierz \mathbf{T}^{-1} , dla macierzy \mathbf{T} będącej macierzą dolnotrójkątną z jedynkami na diagonalu, może być realizowane w sposób rekurencyjny zwany podstawianiem w tył (*back substitution*), który dla macierzy rzadkiej ma złożoność obliczeniową $O(N)$. Wszystkie pozostałe operacje konieczne do wyznaczenia słowa kodowego według przedstawionego algorytmu (mnożenie wektora przez macierz rzadką, dodawanie wektorów) mają także złożoność obliczeniową $O(N)$. Algorytm kodowania ma zatem złożoność obliczeniową $O(N + h^2)$, co precyzyjnie przedstawiają tabele 2.1 i 2.2.

Algorytm 2.2: Kodowanie metodą Richardsona-Urbanke	
Dane wejściowe: wektor informacyjny $\mathbf{u}_{1 \times K}$; macierz kontrolna jak na rys. 2.1, dla której $\Phi = -\mathbf{E}\mathbf{T}^{-1}\mathbf{B} + \mathbf{D}$ jest macierzą nieosobliwą	
Dane wyjściowe: wektor kodowy $\mathbf{x}_{1 \times N}$	
1	Wyznaczyć \mathbf{p}_1 zgodnie z zależnością (2.5) i tabelą 2.1
2	Wyznaczyć \mathbf{p}_2 zgodnie z zależnością (2.6) i tabelą 2.2
3	$\mathbf{x} = [\mathbf{u} \ \mathbf{p}_1 \ \mathbf{p}_2]$

Tab. 2.1: Operacje konieczne do wyznaczenia wektora kontrolnego \mathbf{p}_1^T według (2.5)

Operacja		Złożoność
$\mathbf{A}\mathbf{u}^T$	Mnożenie przez macierz rzadką	$O(N)$
$\mathbf{T}^{-1}(\mathbf{A}\mathbf{u}^T)$	Podstawianie w tył	
	$\mathbf{T}^{-1}(\mathbf{A}\mathbf{u}^T) = \mathbf{q}^T \Leftrightarrow (\mathbf{A}\mathbf{u}^T) = \mathbf{T}\mathbf{q}^T$	$O(N)$
$-\mathbf{E}(\mathbf{T}^{-1}\mathbf{A}\mathbf{u}^T)$	Mnożenie przez macierz rzadką	$O(N)$
$\mathbf{C}\mathbf{u}^T$	Mnożenie przez macierz rzadką	$O(N)$
$(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}\mathbf{u}^T) + (\mathbf{C}\mathbf{u}^T)$	Dodawanie wektorów	$O(N)$
$-\Phi^{-1}(-\mathbf{E}\mathbf{T}^{-1}\mathbf{A}\mathbf{u}^T + \mathbf{C}\mathbf{u}^T)$	Mnożenie przez macierz gęstą	$O(h^2)$

Tab. 2.2: Operacje konieczne do wyznaczenia wektora kontrolnego \mathbf{p}_2^T według (2.6)

Operacja		Złożoność
$\mathbf{A}\mathbf{u}^T$	Mnożenie przez macierz rzadką	$O(N)$
$\mathbf{B}\mathbf{p}_1^T$	Mnożenie przez macierz rzadką	$O(N)$
$(\mathbf{A}\mathbf{u}^T) + (\mathbf{B}\mathbf{p}_1^T)$	Dodawanie wektorów	$O(N)$
$-\mathbf{T}^{-1}(\mathbf{A}\mathbf{u}^T + \mathbf{B}\mathbf{p}_1^T)$	Podstawianie w tył	
	$-\mathbf{T}^{-1}(\mathbf{A}\mathbf{u}^T + \mathbf{B}\mathbf{p}_1^T) = \mathbf{q}^T \Leftrightarrow$	
	$\Leftrightarrow -(\mathbf{A}\mathbf{u}^T + \mathbf{B}\mathbf{p}_1^T) = \mathbf{T}\mathbf{q}^T$	$O(N)$

2.2 Iteracyjne algorytmy dekodowania

Praktyczne realizacje dekodowników LDPC opierają się na iteracyjnych algorytmach dekodowania liniowych kodów blokowych. Metody te są w ogólnym przypadku suboptymalne, jednak ich skuteczność zwykle jest tylko nieznacznie mniejsza od optymalnego dekodowania MAP. Szczególna przydatność iteracyjnych algorytmów dla kodów LDPC wynika z faktu, że dla rzadkiej macierzy kontrolnej zależność złożoności obliczeniowej procesu dekodowania od długości słowa kodowego jest liniowa.

Zdecydowana większość opisywanych i wykorzystywanych w praktyce algorytmów dekodowania należy do grupy algorytmów przekazywania wiadomości (*Message Passing*). Przekazywanie wiadomości może być zobrazowane w postaci grafu Tannera: wierzchołki grafu skojarzone są z pewnymi elementarnymi operacjami realizowanymi w procesie dekodowania, natomiast krawędzie pokazują sposób propagacji wiadomości w kolejnych iteracjach pomiędzy jednostkami obliczeniowymi skojarzonymi z wierzchołkami.

Ze względu na ścisłą zależność pomiędzy grafem Tannera a algorytmem dekodowania, w dalszej części pracy używane będą skrótowe wyrażenia „węzeł realizuje obliczenia” (zamiast „jednostka obliczeniowa skojarzona z wierzchołkiem realizuje obliczenia”) bądź też „węzeł wyznacza wartości wiadomości” (zamiast „jednostka obliczeniowa skojarzona z wierzchołkiem wyznacza wartości wiadomości”).

2.2.1 Algorytm BP

Podstawową metodą dekodowania kodów LDPC jest algorytm propagacji prawdopodobieństw BP (*Belief Propagation*) należący do grupy algorytmów przekazywania wiadomości (*Message Passing*), opracowany pierwotnie przez Gallagera [30].

Algorytm BP zwany jest też algorytmem SPA (*Sum-Product Algorithm*) ze względu na elementarne operacje, które w tym przypadku mają postać sumy iloczynów. Zdecydowana większość algorytmów przekazywania wiadomości stanowi pewną modyfikację algorytmu dekodowania BP. Zmodyfikowane algorytmy można podzielić na takie, które dokładnie odzwierciedlają algorytm BP oraz takie, w których obliczenia przeprowadzane są w sposób przybliżony. Te drugie nazywane będą aproksymowanymi reprezentacjami dekodowania BP. Prezentacja algorytmu BP będzie zatem punktem wyjścia do przedstawienia algorytmów wykorzystanych w implementacji sprzętowej dekodera.

Algorytm BP polega na iteracyjnym obliczaniu przybliżonych wartości prawdopodobieństw *a posteriori* $P(x_n|\mathbf{y}, \mathbf{H})$ dla poszczególnych bitów x_n słowa kodowego \mathbf{x} tak długo, aż twarde decyzje podjęte na podstawie wartości tych prawdopodobieństw będą wskazywały jedno z możliwych słów kodowych lub też osiągnięta zostanie założona maksymalna liczba iteracji. W tym drugim przypadku dekodowanie kończy się niepowodzeniem, w wyniku czego otrzymywana jest jedynie informacja o błędnym bloku, który nie może być skorygowany.

W procesie modyfikacji prawdopodobieństw wykorzystywane są lokalne zależności pomiędzy wartościami bitów a równaniami kontrolnymi kodu. Schemat owych zależności przedstawia graf Tannera, a propagacja prawdopodobieństw pomiędzy wierzchołkami realizującymi obliczenia jest wykonywana zgodnie ze strukturą tego grafu.

Wartości początkowe służące do inicjalizacji zmiennych algorytmu, są prawdopodobieństwami $p_n^0 = P(x_n = 0|y_n)$ oraz $p_n^1 = P(x_n = 1|y_n)$ dostarczonymi z demodulatora (przykładowo dla kanału BIAWGN wyznaczonymi zgodnie z (1.11)). Każda iteracja procesu dekodowania składa się z dwóch etapów: tzw. kroku poziomego, w którym obliczenia są wykonywane dla kolejnych wierszy macierzy \mathbf{H} (odpowiadających wierzchołkom kontrolnym grafu Tannera) oraz kroku pionowego, dla kolejnych kolumn macierzy (odpowiadających wierzchołkom symbolowym).

Opis algorytmu przedstawiony jest na stronie 19 (algorytm 2.3). Za [50,60] przyjęto następujące oznaczenia:

- $\mathcal{M}(n)$ – zbiór numerów wierszy macierzy \mathbf{H} z jedynką w kolumnie n -tej (zbiór indeksów wierzchołków kontrolnych sąsiadujących z wierzchołkiem bitowym b_n),
- $\mathcal{N}(m)$ – zbiór numerów kolumn macierzy \mathbf{H} z jedynką w wierszu m -tym (zbiór indeksów wierzchołków bitowych sąsiadujących z wierzchołkiem kontrolnym c_m),

- $\mathcal{N}(m) \setminus n$ – zbiór $\mathcal{N}(m)$ z wyłączeniem elementu n ,
- $\mathcal{M}(n) \setminus m$ – zbiór $\mathcal{M}(n)$ z wyłączeniem elementu m .

Wiadomości r_{mn}^0 oraz r_{mn}^1 wyznaczone w **kroku poziomym** oznaczają prawdopodobieństwa spełnienia m -tego równania kontrolnego przy założeniu, że symbol n -ty jest równy odpowiednio 0 lub 1, a pozostałe symbole partycypujące w tym równaniu kontrolnym są binarnymi zmiennymi losowymi o rozkładach prawdopodobieństw otrzymanych z wierzchołków bitowych (wiadomości q_{nm}^0 i q_{nm}^1). W celu wyznaczenia prawdopodobieństwa, że dane równanie kontrolne jest spełnione należy zsumować prawdopodobieństwa wszystkich słów zawierających takie kombinacje zer i jedynek, dla których spełnione jest to równanie kontrolne. Prawdopodobieństwo każdej z tych kombinacji jest równe iloczynowi prawdopodobieństw odpowiednich wartości poszczególnych bitów. Otrzymuje się zatem sumę iloczynów prawdopodobieństw po wszystkich możliwych kombinacjach spełniających równanie kontrolne, co formalnie opisano regułą sumy iloczynów przedstawioną wzorami (2.8). Funkcja $f(x_1, \dots, x_n)$ jest funkcją kontrolną – przyjmuje wartość 1, jeśli równanie kontrolne $x_1 \oplus \dots \oplus x_n = 0$ jest spełnione. Sumowanie w (2.8a)-(2.8b) odbywa się po wszystkich możliwych kombinacjach zbioru wartości bitów $\{x_{n'}\}$.

Wiadomości q_{nm}^0 oraz q_{nm}^1 wyznaczone w **kroku pionowym** oznaczają rozkład prawdopodobieństwa dla bitu n -tego, równego (odpowiednio) 0 lub 1, wyznaczony na podstawie wartości inicjalizujących (p_n^0 i p_n^1) oraz wartości prawdopodobieństw spełnienia równań kontrolnych r_{mn}^0 i r_{mn}^1 otrzymanych z wszystkich sąsiadujących węzłów kontrolnych oprócz węzła m -tego. Prawdopodobieństwa q_{nm}^0 i q_{nm}^1 są proporcjonalne do iloczynu tych prawdopodobieństw. Czynniki normalizujące α_{nm} występujący w równaniach (2.9) należy wyznaczyć w taki sposób, aby spełniony był warunek $q_{nm}^0 + q_{nm}^1 = 1$.

Wiadomości wyznaczone zarówno w kroku poziomym, jak i pionowym to tzw. wiadomości zewnętrzne (*extrinsic messages*). Oznacza to, że wartość wiadomości q_{nm} w danej iteracji jest niezależna od wiadomości r_{mn} (czyli propagowanej po tej samej krawędzi grafu Tannera). Podobnie wartość wiadomości r_{mn} jest niezależna od wartości q_{nm} wyznaczonej w poprzedniej iteracji. Ilustruje to rysunek 2.2, na którym przedstawiono sposób wyznaczania przykładowej wiadomości $q_{6,3}$ dla wierzchołka bitowego b_6 grafu Tannera pokazanego na tym rysunku. Wiadomość $r_{3,6}$ nie ma wpływu na wartość wiadomości $q_{6,3}$.

Algorytm 2.3: Algorytm BP (SPA)

Dane wejściowe: Wartości prawdopodobieństw $P(x_n = 0|y_n)$ oraz $P(x_n = 1|y_n)$, $n = 1, \dots, N$

Dane wyjściowe: Wektor kodowy $\hat{\mathbf{x}}_{1 \times N}$

1 *Inicjalizacja.* $i := 1$; Dla każdego n , dla każdego $m \in \mathcal{M}(n)$:

$$q_{nm}^0 := p_n^0 = P(x_n = 0|y_n) \quad (2.7a)$$

$$q_{nm}^1 := p_n^1 = P(x_n = 1|y_n) \quad (2.7b)$$

2 **POWTARZAJ**

3 *Krok poziomy (węzły kontrolne).* Dla każdego m , dla każdego $n \in \mathcal{N}(m)$:

$$r_{mn}^0 := \sum_{\{x_{n'}: n' \in \mathcal{N}(m) \setminus n\}} \left(f(x_n = 0, \{x_{n'}\}) \prod_{n' \in \mathcal{N}(m) \setminus n} q_{n'm}^{x_{n'}} \right) \quad (2.8a)$$

$$r_{mn}^1 := \sum_{\{x_{n'}: n' \in \mathcal{N}(m) \setminus n\}} \left(f(x_n = 1, \{x_{n'}\}) \prod_{n' \in \mathcal{N}(m) \setminus n} q_{n'm}^{x_{n'}} \right) \quad (2.8b)$$

$$f(x_1, \dots, x_n) = \begin{cases} 1 & \Leftrightarrow (x_1 \oplus \dots \oplus x_n = 0) \\ 0 & \text{w innym wypadku} \end{cases} \quad (2.8c)$$

4 *Krok pionowy (węzły bitowe).* Dla każdego n , dla każdego $m \in \mathcal{M}(n)$:

$$q_{nm}^0 := \alpha_{nm} p_n^0 \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m'n}^0 \quad (2.9a)$$

$$q_{nm}^1 := \alpha_{nm} p_n^1 \prod_{m' \in \mathcal{M}(n) \setminus m} r_{m'n}^1 \quad (2.9b)$$

5 *Prawdopodobieństwa pseudo-posteriori.* Dla każdego n :

$$q_n^0 := \alpha_n p_n^0 \prod_{m' \in \mathcal{M}(n)} r_{m'n}^0 \quad (2.10a)$$

$$q_n^1 := \alpha_n p_n^1 \prod_{m' \in \mathcal{M}(n)} r_{m'n}^1 \quad (2.10b)$$

6 *Próbne decyzje.* Dla każdego n :

$$\hat{x}_n := \begin{cases} 1 & \Leftrightarrow q_n^1 > 0.5 \\ 0 & \Leftrightarrow q_n^1 \leq 0.5 \end{cases} \quad (2.11)$$

7 $i := i + 1$

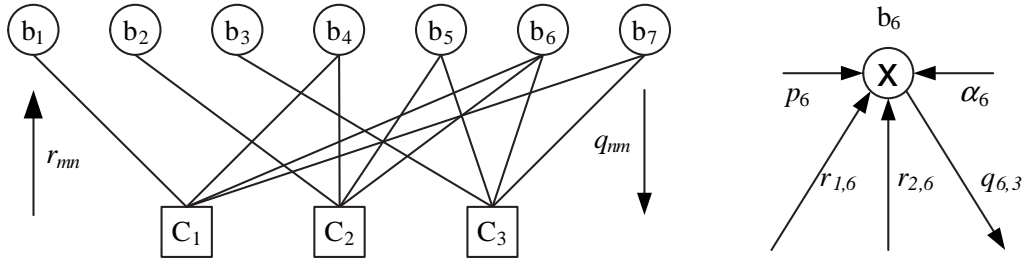
8 **AŻ** ($\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$) lub ($i = i_{\max}$)

9 **JEŻELI** $\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$

10 \lfloor Wektor $\hat{\mathbf{x}}$ jest wektorem zdekodowanym.

11 **W PRZECIWNYM WYPADKU**

12 \lfloor Dekodowanie nie zakończone sukcesem



Rys. 2.2: Przykładowa zależność wiadomości w algorytmie BP

Próbné dekodowanie (2.11) jest wykonywane na podstawie prawdopodobieństw q_n^0 i q_n^1 (2.10), gdzie uwzględnione są informacje wejściowe otrzymane z wszystkich sąsiadujących węzłów kontrolnych. Jeśli równanie kontrolne $\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$ dla twardej decyzji podjętych w danej iteracji jest spełnione, to proces dekodowania zostaje zakończony, a wynikiem jest słowo $\hat{\mathbf{x}}$. Przerwanie procesu następuje również przy osiągnięciu założonej maksymalnej liczby iteracji i_{\max} (np. $i_{\max} = 100$). W tym wypadku $\hat{\mathbf{x}}$ nie jest poprawnym słowem kodowym.

Jak wykazano [29, 50], w przypadku kodu o grafie w postaci drzewa, czyli pozbawionego cykli, wartości q_n^0 , q_n^1 (2.10) odzwierciedlają dokładne wielkości poszukiwanych prawdopodobieństw *a posteriori* $P(x_n|\mathbf{y}, \mathbf{H})$ po wykonaniu liczby iteracji równej (co najwyżej) liczbie krawędzi w najdłuższej ścieżce grafu (drzewa) kodu. Jest tak na przykład dla trywialnego kodu SPC (*Single Parity Check*), który powstaje poprzez dodawanie do słowa informacyjnego pojedynczego bitu parzystości. Macierz kontrolna takiego kodu składa się z jednego wiersza (pojedynczego równania kontrolnego). Dla kodów o grafach zawierających cykle, wartości wyznaczone w (2.10) stanowią iteracyjnie obliczane przybliżenie prawdopodobieństw *a posteriori*, stąd zwane są one prawdopodobieństwami *pseudo-posteriori*. Im mniej krótkich cykli w grafie Tannera kodu, tym dokładniejsze jest przybliżenie wartości prawdopodobieństw *a posteriori*. Fakt ten jest jedną z przyczyn poszukiwania kodów o grafach zawierających jak najmniej krótkich cykli.

Przykład 1. Dekodowanie algorytmem BP (SPA)

Dany jest liniowy kod blokowy o macierzy kontrolnej:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Niech nadany wektor kodowy będzie równy:

$$\mathbf{x} = [0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1]$$

natomiast po stronie odbiorczej niech wartości prawdopodobieństw *a priori* poszczególnych bitów (miękkich decyzji na wejściu dekodera) mają wartość:

$$p_n^1 = [0.7 \ 0.9 \ 0.9 \ 0.9 \ 0.5 \ 0.9 \ 0.1 \ 0.9 \ 0.9 \ 0.1 \ 0.1 \ 0.3]$$

Twarde decyzje w tym przypadku wskazują niepoprawne słowo: przekłamanie są bity pierwszy i ostatni, jak również jest niepewność co do bitu piątego (wartości te wyróżniono).

Inicjalizacja dekodera polega na przypisaniu wartości do wiadomości q_{nm}^0 oraz q_{nm}^1 równych prawdopodobieństwom *a priori*:

$$q_{nm}^1 = \begin{bmatrix} 0.7 & 0.9 & 0.9 & 0.9 & - & - & 0.1 & 0.9 & - & - & - & - \\ - & 0.9 & - & - & 0.5 & 0.9 & - & 0.9 & 0.9 & - & - & - \\ - & - & 0.9 & 0.9 & - & 0.9 & - & - & 0.9 & 0.1 & - & - \\ 0.7 & - & - & - & 0.5 & - & 0.1 & - & - & 0.1 & 0.1 & - \\ - & 0.9 & - & 0.9 & - & 0.9 & - & - & - & - & 0.1 & 0.3 \\ 0.7 & - & 0.9 & - & 0.5 & - & 0.1 & - & - & - & - & 0.3 \end{bmatrix}$$

Następuje etap obliczeń dla **węzłów kontrolnych**. Obliczane są sumy iloczynów odpowiednich wartości prawdopodobieństw r_{mn} . Przykładowo $r_{6,1}^1$

$$\begin{aligned} r_{6,1}^1 &= q_{6,3}^1 q_{6,5}^0 q_{6,7}^0 q_{6,12}^0 + q_{6,3}^0 q_{6,5}^1 q_{6,7}^0 q_{6,12}^0 + q_{6,3}^0 q_{6,5}^0 q_{6,7}^1 q_{6,12}^0 + q_{6,3}^0 q_{6,5}^0 q_{6,7}^0 q_{6,12}^1 \\ &+ q_{6,3}^0 q_{6,5}^1 q_{6,7}^1 q_{6,12}^1 + q_{6,3}^1 q_{6,5}^0 q_{6,7}^1 q_{6,12}^1 + q_{6,3}^1 q_{6,5}^1 q_{6,7}^0 q_{6,12}^1 + q_{6,3}^1 q_{6,5}^1 q_{6,7}^1 q_{6,12}^0 \\ &= 0.9 \cdot 0.5 \cdot 0.9 \cdot 0.7 + 0.1 \cdot 0.5 \cdot 0.9 \cdot 0.7 + 0.1 \cdot 0.5 \cdot 0.1 \cdot 0.7 + 0.1 \cdot 0.5 \cdot 0.9 \cdot 0.3 \\ &+ 0.1 \cdot 0.5 \cdot 0.1 \cdot 0.3 + 0.9 \cdot 0.5 \cdot 0.1 \cdot 0.3 + 0.9 \cdot 0.5 \cdot 0.9 \cdot 0.3 + 0.9 \cdot 0.5 \cdot 0.1 \cdot 0.7 \\ &= 0.5 \end{aligned}$$

Po powtórzeniu tej procedury dla wszystkich krawędzi grafu Tannera otrzymuje się wszystkie wymagane wartości prawdopodobieństw r_{mn} :

$$r_{mn}^1 = \begin{bmatrix} 0.336 & 0.418 & 0.418 & 0.418 & - & - & 0.582 & 0.418 & - & - & - & - \\ - & 0.5 & - & - & 0.295 & 0.5 & - & 0.5 & 0.5 & - & - & - \\ - & - & 0.705 & 0.705 & - & 0.705 & - & - & 0.705 & 0.295 & - & - \\ 0.5 & - & - & - & 0.602 & - & 0.5 & - & - & 0.5 & 0.5 & - \\ - & 0.398 & - & 0.398 & - & 0.398 & - & - & - & - & 0.602 & 0.705 \\ 0.5 & - & 0.5 & - & 0.449 & - & 0.5 & - & - & - & - & 0.5 \end{bmatrix}$$

W etapie **węzłów bitowych** obliczane są iloczyny odpowiednich prawdopodobieństw wraz z wagami normalizującymi α . Przykładowo $q_{1,6}^1$ (pierwszy węzeł bitowy - pierwsza kolumna, szósty węzeł kontrolny - szósty wiersz)

$$q_{1,6}^1 = \alpha_{1,6} p_1^1 r_{1,1}^1 r_{4,1}^1 = \alpha_{1,6} \cdot 0.7 \cdot 0.336 \cdot 0.5 = 0.542$$

$$q_{1,6}^0 = \alpha_{1,6} p_1^0 r_{1,1}^0 r_{4,1}^0 = \alpha_{1,6} \cdot 0.3 \cdot 0.664 \cdot 0.5 = 0.458$$

Po powtórzeniu tej procedury dla wszystkich krawędzi grafu Tannera otrzymuje się wszystkie wymagane wartości prawdopodobieństw q_{nm} :

$$q_{nm}^1 = \begin{bmatrix} 0.7 & 0.856 & 0.956 & 0.934 & - & - & 0.1 & 0.9 & - & - & - & - \\ - & 0.81 & - & - & 0.552 & 0.934 & - & 0.866 & 0.956 & - & - & - \\ - & - & 0.866 & 0.81 & - & 0.856 & - & - & 0.9 & 0.1 & - & - \\ 0.542 & - & - & - & 0.254 & - & 0.134 & - & - & 0.044 & 0.144 & - \\ - & 0.866 & - & 0.939 & - & 0.956 & - & - & - & - & 0.1 & 0.3 \\ 0.542 & - & 0.939 & - & 0.388 & - & 0.134 & - & - & - & - & 0.506 \end{bmatrix}$$

Obliczane są również prawdopodobieństwa *pseudo-posteriori*, przykładowo q_1^1 :

$$q_1^1 = \alpha_1 p_1^1 r_{1,1}^1 r_{4,1}^1 r_{6,1}^1 = \alpha_1 \cdot 0.7 \cdot 0.336 \cdot 0.5 \cdot 0.5 = 0.542$$

$$q_1^0 = \alpha_1 p_1^0 r_{1,1}^0 r_{4,1}^0 r_{6,1}^0 = \alpha_1 \cdot 0.3 \cdot 0.664 \cdot 0.5 \cdot 0.5 = 0.458$$

Na podstawie wektora prawdopodobieństw *pseudo-posteriori* po 1 iteracji:

$$q_n^1 = \begin{bmatrix} \mathbf{0.542} & 0.81 & 0.939 & 0.911 & 0.341 & 0.934 & 0.134 & 0.866 & 0.956 & 0.0445 & 0.144 & 0.506 \end{bmatrix}$$

podejmowane są **próbne decyzje**:

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{1} & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Ponieważ $\mathbf{H}\hat{\mathbf{x}}^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}^T$ (trzy równania kontrolne niespełnione), słowo $\hat{\mathbf{x}}$ nie jest słowem kodowym. Wykonywana jest zatem kolejna iteracja algorytmu.

Po drugiej iteracji otrzymuje się wartości prawdopodobieństw:

$$q_n^1 = \begin{bmatrix} 0.405 & 0.758 & 0.921 & 0.893 & 0.338 & 0.908 & 0.144 & 0.849 & 0.934 & 0.064 & 0.159 & 0.536 \end{bmatrix}$$

które odpowiadają poprawnemu słowu kodowemu

$$\hat{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Proces dekodowania jest zatem kończony, a jego wynikiem jest słowo równe założonemu na wstępie słowu nadanemu.

2.2.2 Modyfikacje dekodowania BP

Zamiast bezpośrednich wartości prawdopodobieństw, lepiej nadające się do sprzętowej implementacji dekodera jest zastosowanie jako wiadomości pewnych przekształceń (funkcji) wartości prawdopodobieństw [42, 50], takich jak:

- stosunek prawdopodobieństw (LR, *Likelihood Ratio*) $f_\lambda(p^0, p^1) = (p^0/p^1)$
- logarytmiczny stosunek prawdopodobieństw (LLR, *Log-Likelihood Ratio*)
 $f_\Lambda(p^0, p^1) = \ln(p^0/p^1)$
- różnica prawdopodobieństw (LD, *Likelihood Difference*) $f_\delta(p^0, p^1) = p^0 - p^1$
- logarytmiczna różnica prawdopodobieństw (SLLD, *Signed Log-Likelihood Difference*) $f_\Delta(p^0, p^1) = \text{sgn}(p^0 - p^1) \ln |p^0 - p^1|$

Spośród wyżej wymienionych przekształceń zdecydowanie największe znaczenie ma logarytmiczny stosunek prawdopodobieństw LLR. Większość prac dotyczących symulacji i implementacji układów dekodowania LDPC opiera się na zastosowaniu przekształcenia LLR. Pozostałe funkcje mają znacznie mniejsze znaczenie praktyczne. Zmodyfikowany algorytm BP, wykorzystujący wiadomości w postaci logarytmicznego stosunku prawdopodobieństw, nazywany będzie algorytmem LLR-BP.

Algorytm LLR-BP

Korzyści związane z zastosowaniem algorytmu LLR-BP w porównaniu z BP są znaczące. Operacje wyznaczania iloczynów (BP) odpowiadają operacjom sumowania w LLR-BP, nie jest wymagane wyznaczanie współczynników normalizujących α_{nm} w kroku pionowym (2.9), wreszcie algorytm ma mniejszą wrażliwość na błędy zaokrągleń wiadomości [77]. Dzięki temu podobny efekt dekodowania można uzyskać przy zastosowaniu krótszych stałoprzecinkowych słów wiadomości w postaci LLR.

Funkcja LLR binarnej zmiennej losowej U jest definiowana jako:

$$L(U) = \ln \left(\frac{P_U(u=0)}{P_U(u=1)} \right) \quad (2.12)$$

gdzie $P_U(u)$ oznacza prawdopodobieństwo, że zmienna losowa U przyjmuje wartość u . Znak wartości funkcji $L(U)$ informuje, która wartość zmiennej U jest bardziej prawdopodobna, a moduł zależy od poziomu wiarygodności informacji o wartości tej zmiennej.

Można wykazać, że dla dwóch statystycznie niezależnych zmiennych losowych U i V , funkcja LLR sumy modulo-2 tych zmiennych jest równa [32]:

$$L(U \oplus V) = \ln \frac{1 + e^{L(U)+L(V)}}{e^{L(U)} + e^{L(V)}} = 2 \tanh^{-1} \left(\tanh \left(\frac{L(U)}{2} \right) \tanh \left(\frac{L(V)}{2} \right) \right) \quad (2.13)$$

Funkcję $\tanh(x)$ będącą monotonicznie rosnącą funkcją niesymetryczną, można też przedstawić jako $\tanh(x) = \text{sgn}(x) \tanh(|x|)$. Podobnie można wyrazić $\tanh^{-1}(x)$.

Funkcja LLR sumy modulo-2 dwóch zmiennych, wyznaczana na podstawie (2.13), będzie oznaczana w dalszej części pracy (za [32]) jako:

$$L(U \oplus V) \triangleq L(U) \boxplus L(V) \quad (2.14)$$

Operacja \boxplus jest przemienne i łączna.

Po przekształceniu wiadomości w algorytmie BP do postaci: $Q_{nm} = \ln(q_{nm}^0/q_{nm}^1)$, $R_{mn} = \ln(r_{mn}^0/r_{mn}^1)$, otrzymuje się algorytm LLR-BP, przedstawiony formalnie na stronie 25 (algorytm 2.4). Z zależności (2.13) wynika, że suma iloczynów wyliczana w kroku poziomym algorytmu BP, po przekształceniu wiadomości do postaci LLR, uzyskuje postać iloczynu tangensów hiperbolicznych. Iloczynowi wyznaczanemu w kroku pionowym BP odpowiada natomiast suma wiadomości LLR.

Jest to podstawowa wersja algorytmu LLR-BP. Iloczyn występujący w równaniu dla wierzchołków kontrolnych (2.19) można zastąpić sumą, gdyż operacja $L(U) \boxplus L(V)$ może być alternatywnie przedstawiona jako [9]:

$$L(U) \boxplus L(V) = \operatorname{sgn}(L(U)) \operatorname{sgn}(L(V)) \times \psi^{-1}(\psi(|L(U)|) + \psi(|L(V)|)) \quad (2.15)$$

$$\psi(x) = \psi^{-1}(x) = -\ln(\tanh(x/2)) \quad (2.16)$$

Z (2.15) wynika alternatywna postać równania (2.19) określającego wiadomości w kroku poziomym:

$$R_{mn} := \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \operatorname{sgn}(Q_{n'm}) \right) \times \psi^{-1} \left(\sum_{n' \in \mathcal{N}(m) \setminus n} \psi(|Q_{n'm}|) \right) \quad (2.17)$$

gdzie $\psi(x)$ zdefiniowano w (2.16).

Schemat podwójnej rekurencji

W celu wyliczenia sum (2.20) niezależnie dla wszystkich krawędzi $m \in \mathcal{M}(n)$ danego wierzchołka, konieczne jest wykonanie $(d_b - 1)d_b$ dwuargumentowych operacji sumowania (gdzie $d_b = |\mathcal{M}(n)|$ to stopień wierzchołka symbolowego). Łatwo jednak zauważyć, że wiele z tych dwuargumentowych sumowań jest powtarzanych. W celu pozbycia się tego nadmiaru, można przykładowo wyznaczyć pełną sumę (2.21), a następnie wyliczyć wiadomości wyjściowe (2.20) poprzez odejmowanie odpowiednich składników. W ten sposób wyznaczenie wszystkich wiadomości wyjściowych oraz prawdopodobieństw *pseudo-posteriori* wymaga jedynie $2d_b$ dwuargumentowych operacji (d_b dodawań i d_b odejmowań).

Algorytm 2.4: Algorytm LLR-BP

Dane wejściowe: Wartości prawdopodobieństw $P(x_n = 0|y_n)$ oraz
 $P(x_n = 1|y_n)$, $n = 1, \dots, N$

Dane wyjściowe: Wektor kodowy $\hat{\mathbf{x}}_{1 \times N}$

1 *Inicjalizacja.* $i := 1$; Dla każdego n , dla każdego $m \in \mathcal{M}(n)$:

$$Q_{nm} := L(x_n|y_n) = \ln \left(\frac{P(x_n = 0|y_n)}{P(x_n = 1|y_n)} \right) \quad (2.18)$$

2 **POWTARZAJ**

3 *Krok poziomy (węzły kontrolne).* Dla każdego m , dla każdego $n \in \mathcal{N}(m)$:

$$R_{mn} := \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \text{sgn}(Q_{n'm}) \right) \times 2 \tanh^{-1} \left(\prod_{n' \in \mathcal{N}(m) \setminus n} \tanh \left(\frac{|Q_{n'm}|}{2} \right) \right) \quad (2.19)$$

4 *Krok pionowy (węzły bitowe).* Dla każdego n , dla każdego $m \in \mathcal{M}(n)$:

$$Q_{nm} := L(x_n|y_n) + \sum_{m' \in \mathcal{M}(n) \setminus m} R_{m'n} \quad (2.20)$$

5 *LLR prawdopodobieństw pseudo-posteriori.* Dla każdego n :

$$Q_n := L(x_n|y_n) + \sum_{m \in \mathcal{M}(n)} R_{mn} \quad (2.21)$$

6 *Próbnne decyzje.* Dla każdego n :

$$\hat{x}_n := \begin{cases} 1 & \Leftrightarrow Q_n < 0 \\ 0 & \Leftrightarrow Q_n \geq 0 \end{cases} \quad (2.22)$$

7 $i := i + 1$

8 **AŻ** ($\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$) lub ($i = i_{\max}$)

9 **JEŻELI** $\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$

10 \lfloor Wektor $\hat{\mathbf{x}}$ jest wektorem zdekodowanym.

11 **W PRZECIWNYM WYPADKU**

12 \lfloor Dekodowanie nie zakończone sukcesem

Podobny nadmiar operacji występuje w przypadku niezależnego wyznaczania wiadomości w wierzchołkach kontrolnych. W celu lepszej organizacji obliczeń stosuje się tzw. algorytm „wprzód - w tył” (*forward-backward*), nazywany też schematem podwójnej rekurencji. Algorytm ten pierwotnie został przedstawiony dla kodów liniowych w [1], a na gruncie kodów LDPC opisany został w [9, 39].

Rozpatrzmy wierzchołek kontrolny c_m stopnia d_c , połączony z wierzchołkami symbolowymi $\mathcal{N}(m) = (b_{n_1}, b_{n_2}, \dots, b_{n_{d_c}})$. Wiadomościami wejściowymi są funkcje LLR $Q_{n_1 m} = L(x_{n_1}), Q_{n_2 m} = L(x_{n_2}), \dots, Q_{n_{d_c} m} = L(x_{n_{d_c}})$. Zdefiniujmy dwa zbiory pomocniczych zmiennych: dla rekurencji wprzód $\{\alpha_1, \alpha_2, \dots, \alpha_{d_c}\}$ i w tył $\{\beta_1, \beta_2, \dots, \beta_{d_c}\}$:

$$\begin{aligned} \alpha_1 &= x_{n_1}, \alpha_2 = \alpha_1 \oplus x_{n_2}, \alpha_3 = \alpha_2 \oplus x_{n_3}, \dots, \alpha_{d_c} = \alpha_{d_c-1} \oplus x_{n_{d_c}} \\ \beta_{d_c} &= x_{n_{d_c}}, \beta_{d_c-1} = b_{d_c} \oplus x_{n_{d_c-1}}, \beta_{d_c-2} = \beta_{d_c-1} \oplus x_{n_{d_c-2}}, \dots, \beta_1 = \beta_2 \oplus x_{n_1} \end{aligned}$$

Powtarzając operację $L(U) \boxplus L(V)$, według (2.13) w sposób rekurencyjny, otrzymuje się wartości $L(\alpha_1), L(\alpha_2), \dots, L(\alpha_{d_c})$ (rekurencja „wprzód”) oraz $L(\beta_{d_c}), L(\beta_{d_c-1}), \dots, L(\beta_1)$ (rekurencja „w tył”). Ponieważ oczekujemy spełnienia równania kontrolnego $(x_{n_1} \oplus x_{n_2} \oplus \dots \oplus x_{n_{d_c}}) = 0$, to x_{n_i} można wyrazić jako $x_{n_i} = (\alpha_{i-1} \oplus \beta_{i+1})$ dla każdego $i \in \{2, 3, \dots, d_c - 1\}$. Sposób wyznaczania wiadomości dla kroku poziomego algorytmu LLR-BP (węzły kontrolne) ze schematem podwójnej rekurencji można zatem formalnie zapisać następująco:

$$\begin{aligned} L(\alpha_1) &= Q_{n_1 m} \\ L(\alpha_i) &= L(\alpha_{i-1}) \boxplus Q_{n_i m}, \quad i = 2, 3, \dots, d_c \end{aligned} \quad (2.23a)$$

$$\begin{aligned} L(\beta_{d_c}) &= Q_{n_{d_c} m} \\ L(\beta_i) &= L(\beta_{i+1}) \boxplus Q_{n_i m}, \quad i = d_c - 1, d_c - 2, \dots, 1 \end{aligned} \quad (2.23b)$$

$$R_{mn_i} = \begin{cases} L(\beta_2), & i = 1 \\ L(\alpha_{i-1}) \boxplus L(\beta_{i+1}), & i = 2, 3, \dots, d_c - 1 \\ L(\alpha_{d_c-1}), & i = d_c \end{cases} \quad (2.23c)$$

Obliczenia dla węzła kontrolnego, składające się z rekurencyjnego wyznaczania $L(\alpha_i), L(\beta_i)$ oraz finalnego parowania wartości w (2.23c), wymagają wykonania sumarycznie $(3d_c - 4)$ dwuargumentowych operacji \boxplus . Obliczenia realizowane zgodnie z (2.17) wymagają d_c^2 operacji $\psi(x)$ oraz $d_c(d_c - 1)$ sumowań lub alternatywnie $d_c(d_c - 2)$ dwuargumentowych operacji \boxplus . Przykładowo dla $d_c = 8$ daje to 48 operacji \boxplus , podczas gdy zastosowanie schematu podwójnej rekurencji wymaga wykonania jedynie 20 operacji \boxplus .

Dwuargumentowa operacja $L(U) \boxplus L(V)$ może być realizowana na szereg sposobów. Jednym z nich jest bezpośrednio zastosowanie (2.13) lub (2.15), co jednak nie jest najefektywniejszym rozwiązaniem. Wzór (2.13) można przekształcić korzystając z zależności [86]: $\ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$. Na jej podstawie łatwo wykazać, że:

$$\begin{aligned} L(U) \boxplus L(V) &= \ln \frac{1 + e^{L(U)+L(V)}}{e^{L(U)} + e^{L(V)}} = \operatorname{sgn}(L(U)) \operatorname{sgn}(L(V)) \min(|L(U)|, |L(V)|) \\ &\quad + \ln \left(1 + e^{-|L(U)+L(V)|}\right) - \ln \left(1 + e^{-|L(U)-L(V)|}\right) \end{aligned} \quad (2.24)$$

Nieliniowe operacje $\ln(1 + e^{-|L(U)+L(V)|})$ oraz $\ln(1 + e^{-|L(U)-L(V)|})$ mogą być zrealizowane sprzętowo w postaci tablicy wartości (*look-up table*) lub w postaci aproksymacji odcinkowo-liniowej [39].

2.2.3 Aproksymowane reprezentacje dekodowania BP

Przedstawione w poprzednim podrozdziale zmodyfikowane i uproszczone (pod względem złożoności obliczeń) wersje algorytmu BP, były wierną reprezentacją tego algorytmu. Możliwe jest dalsze upraszczanie algorytmu dekodowania, poprzez realizację obliczeń w sposób przybliżony. Zmniejszenie nakładów obliczeniowych odbywa się zatem kosztem pewnego (niewielkiego) pogorszenia skuteczności algorytmu, a w konsekwencji zwiększenia bitowej stopy błędów systemu. Rozpatrywany będzie jedynie krok poziomy algorytmu, gdyż dla kroku pionowego suma (2.20) zasadniczo nie daje się już uprościć. Krok próbnych decyzji również pozostaje bez zmian. Znaki wiadomości wyjściowych węzła kontrolnego wyznaczane są zawsze w ten sam sposób (niezależnie od modułów, jak w (2.19)), stąd też w niniejszym podrozdziale rozpatrywane jest tylko wyznaczanie modułów wartości wiadomości.

Algorytm Min-Sum

Na podstawie (2.24), najprostsza aproksymacja operacji \boxplus ma postać:

$$|L(U) \boxplus L(V)| \approx \min(|L(U)|, |L(V)|) \quad (2.25)$$

Korzystając z tej aproksymacji definiuje się algorytm Min-Sum, dla którego moduły wiadomości wyjściowych węzłów kontrolnych wyznaczane są zgodnie z:

$$|R_{mn}| := \min_{n' \in \mathcal{N}(m) \setminus n} (|Q_{n'm}|) \quad (2.26)$$

a pozostałe operacje realizowane tak, jak w algorytmie LLR-BP. Przy wyznaczaniu wiadomości można skorzystać ze schematu podwójnej rekurencji (2.23).

Moduł wiadomości wyjściowej wierzchołka kontrolnego (a więc poziom wiarygodności informacji dostarczanej przez wierzchołek) jest nie większy niż najmniejszy moduł wiadomości wejściowych [11], lub inaczej $|L(U) \boxplus L(V)| \leq \min(|L(U)|, |L(V)|)$. Można zatem zaproponować proste metody korekcji wiadomości wyznaczonych algorytmem Min-Sum. Polegają one na normalizacji wyniku (2.26) pewną stałą A bądź też odjęciu dodatniej stałej C .

Algorytm Normalized-Min-Sum

W algorytmie tym wiadomość wyznaczona zgodnie z algorytmem Min-Sum jest normalizowana stałą A :

$$|R_{mn}| := \frac{\min_{n' \in \mathcal{N}(m) \setminus n} (|Q_{n'm}|)}{A}, \quad A > 1 \quad (2.27)$$

Sposób wyznaczania parametru A prowadzący do minimalizacji błędu aproksymacji został przedstawiony w [11]. Alternatywnie można skorzystać z symulacji lub procedury ewolucji gęstości prawdopodobieństwa (DE - *Density Evolution*) [12, 122]. Jakkolwiek wartość A wyznaczona jedną z tych metod generalnie zależy od kodu oraz od poziomu SNR, często jednak satysfakcjonujące rezultaty otrzymuje się przyjmując stałą wartość A wynoszącą np. $A = 1.33$ (przykład nieprzypadkowy, gdyż dzielenie przez 1.33 odpowiada mnożeniu przez 0.75, co jest łatwo realizowalne sprzętowo za pomocą pojedynczego sumatora). Kwestia doboru wartości A zostanie bliżej przedstawiona w rozdziale dotyczącym implementacji modułów realizujących funkcje wierzchołka kontrolnego.

Algorytm Offset-Min-Sum

W tym przypadku wiadomość wyznaczona zgodnie z algorytmem Min-Sum jest korygowana stałą C :

$$|R_{mn}| := \max \left(\min_{n' \in \mathcal{N}(m) \setminus n} (|Q_{n'm}|) - C, 0 \right), \quad C > 0 \quad (2.28)$$

Algorytm Offset Min-Sum opisywany jest w wielu publikacjach. Sposób wyznaczenia składnika korekcyjnego C za pomocą ewolucji gęstości prawdopodobieństwa zaproponowano między innymi w [9, 10, 12]. Oprócz najprostszego przypadku, w którym wartość C jest przyjmowana jako stała, istnieją wersje algorytmu, gdzie C jest uzależniane od wartości $\min(|Q_{n'm}|)$ [43] jak również od stopnia wierzchołka kontrolnego [35].

W prosty sposób można wykorzystać w powyższych algorytmach schemat podwójnej rekurencji. Wyznaczanie $L(\alpha_i)$, $L(\beta_i)$ należy oprzeć na zależności $|L(U) \boxplus L(V)| \approx \min(|L(U)|, |L(V)|)$, natomiast w (2.23c) należy dodatkowo uwzględnić czynnik A lub składnik C .

Aproksymowana reprezentacja algorytmu LLR-BP z podwójną rekurencją

Nieco odmienne podejście polega na uwzględnieniu korekcji funkcji $\min()$ w kolejnych krokach rekurencji. W ogólny sposób sformułowano to w postaci zależności (2.29).

$$|L(U) \boxplus L(V)| = \max \{ \min(|L(U)|, |L(V)|) + f(|L(U)|, |L(V)|), 0 \} \quad (2.29)$$

Funkcja $f(x, y)$ to człon korekcyjny, który może być wyliczany z większą lub mniejszą precyzją:

- Dokładna wartość funkcji (na podstawie (2.24)) wynosi:

$$f(x, y) = \ln(1 + e^{-|x+y|}) - \ln(1 + e^{-|x-y|}) \quad (2.30)$$

- Zakładając, że $e^{-|x+y|} \ll 1$, otrzymuje się przybliżoną wartość [125]:

$$f(x, y) = -\ln(1 + e^{-|x-y|}) \quad (2.31)$$

- Najprostsza postać członu korekcyjnego daje Algorytm Offset-Min-Sum:

$$f(x, y) = -C, \quad C > 0 \quad (2.32)$$

- W licznych publikacjach [27, 119, 120] proponowana jest warunkowa korekcja stałą (*Conditional Offset-Min-Sum*), która może być zdefiniowana następująco [27]:

$$f(x, y) = \begin{cases} -C, & (|x+y| > 2|x-y|) \text{ oraz } (|x-y| < 2) \\ 0, & \text{w innym wypadku} \end{cases} \quad (2.33)$$

- Odcinkowo liniowa aproksymacja (2.30) zaproponowana w [67] ma postać:

$$f(x, y) = \max\left(\frac{5}{8} - \frac{|x-y|}{4}, 0\right) - \max\left(\frac{5}{8} - \frac{|x+y|}{4}, 0\right) \quad (2.34)$$

2.2.4 Algorytmy z twardymi decyzjami

W skrajnie uproszczonej metodzie dekodowania, wiadomości propagowane w dekodерze są słowami 1-bitowymi ($Q_{nm} \in \{0, 1\}$, $R_{mn} \in \{0, 1\}$). Można wówczas mówić o dekodерze z twardymi decyzjami, gdyż iteracyjnie modyfikowane decyzje co do wartości poszczególnych bitów, są zerojedynkowe. Informacja dostarczana z demodulatora jest również w postaci „twardych” wartości, tzn. $y = [y_1, y_2, \dots, y_N]$, $y_i \in \{0, 1\}$. Podstawowe algorytmy z twardymi decyzjami zwane są algorytmami Gallagera A oraz B [2, 59], dla których przekazywanie 1-bitowych wiadomości jest wykonywane zgodnie ze strukturą grafu Tannera. Do rodziny algorytmów dekodowania o twardych decyzjach zaliczany jest również Algorytm E [83], w którym alfabet wiadomości jest postaci $\{0, 1, e\}$, gdzie e (tzw. *erasure*) oznacza stan niepewności. Przykładowo wiadomość z węzła bitowego równa jest e , jeśli około połowa wiadomości wejściowych ma wartość 0, a druga połowa 1. Nieco odmiennym algorytmem o twardych decyzjach jest algorytm BF (*Bit Flipping*, [49]), dla którego w kroku poziomym wyznaczane są numery niespełnionych równań kontrolnych, natomiast w kroku pionowym wykonywana jest operacja negacji tych bitów, które są zawarte w największej liczbie niespełnionych równań kontrolnych. Nie występuje tu propagacja wiadomości zgodnie z grafem Tannera, więc algorytmu BF nie można zaliczyć do rodziny algorytmów przekazywania wiadomości.

2.3 Podsumowanie

Przedstawione w niniejszym rozdziale algorytmy kodowania i dekodowania są znanymi z literatury zagadnieniami dotyczącymi systemów kodowania LDPC. Stanowią one wprowadzenie do dalszych rozważań: przedstawienia opracowanej efektywnej implementacji modułu dekodera, jak również zaproponowania algorytmów tworzenia macierzy kontrolnych kodów, które są efektywnie dekodowane sprzętowo.

3. Teza i cele pracy

Zasadniczym wkładem naukowym pracy są algorytmy generacji kodów LDPC o strukturze macierzy kontrolnej umożliwiającej efektywną implementację sprzętowych modułów dekodera, konkurencyjnych dla znanych z literatury kodów nie zorientowanych na implementację. Sformułowano zatem następującą tezę pracy:

Istnieje możliwość generowania macierzy kontrolnych kodów LDPC efektywnie dekodowanych w strukturach programowalnych, konkurencyjnych do znanych kodów LDPC pod względem własności korekcyjnych.

Celem pracy jest opracowanie konfigurowalnego modułu dekodera kodów LDPC, efektywnego pod względem przepustowości, wymaganych zasobów sprzętowych oraz jakości dekodowania, który może realizować swoje funkcje dla dowolnego kodu AA-LDPC, implementacja dekodera w strukturach programowalnych typu FPGA oraz zaproponowanie algorytmów generacji kodów AA-LDPC zapewniających niską bitową stopę błędów systemu transmisji.

W procesie prowadzącym do osiągnięcia celów pracy zostały wyodrębnione następujące zadania cząstkowe:

1. Opracowanie synteżowalnego opisu w języku VHDL efektywnych modułów dekodera regularnych, jak i nieregularnych kodów AA-LDPC, działających zgodnie z iteracyjnymi algorytmami przekazywania wiadomości, przy wykorzystaniu różnych uproszczeń algorytmu dekodowania.
2. Opracowanie metod automatycznego dostosowania synteżowalnego opisu modułu dekodera do kodu o danej macierzy kontrolnej.
3. Stworzenie zintegrowanego środowiska projektowania, testowania i symulacji systemów kodowania LDPC, opartego na implementacji dekodera w układzie typu FPGA oraz modelach pozostałych elementów systemu transmisyjnego (koder, modulator, demodulator i kanał transmisyjny) w środowisku MATLAB.
4. Weryfikacja skuteczności działania modułu dekodera sprzętowego wykorzystującego wiadomości o ograniczonej precyzji. Przeprowadzenie analizy i porówna-

nia własności systemów kodowania LDPC w warunkach dekodowania o ograniczonej precyzji. Porównanie przepustowości, wymaganych zasobów układu programowalnego oraz bitowej stopy błędów systemu w zależności od zaimplementowanego algorytmu dekodowania.

5. Opracowanie algorytmów generacji kodów LDPC efektywnie dekodowanych sprzętowo, a jednocześnie konkurencyjnych ze względu na własności korekcyjne do kodów LDPC utworzonych za pomocą znanych z literatury metod. Algorytmy powinny umożliwiać generację macierzy kontrolnych kodów regularnych, jak i nieregularnych, o dowolnej stopie R , długościach bloku N oraz rozmiarze podmacierzy P .
6. Przeprowadzenie badań eksperymentalnych własności kodów uzyskanych za pomocą opracowanych algorytmów, przy wykorzystaniu stworzonego środowiska symulacji systemów kodowania LDPC. Porównanie uzyskanych kodów z kodami utworzonymi za pomocą innych znanych z literatury metod.

3.1 Układ pracy

Rozważania prezentowane w pracy można podzielić na dwa zasadnicze wątki. Pierwszy dotyczy implementacji dekodera w strukturach programowalnych (rozdział 4), natomiast drugi – algorytmów generacji macierzy kontrolnych kodów efektywnie dekodowanych (rozdział 5).

Rozdział 4 odnosi się zatem do realizacji zadań cząstkowych 1-4. Przedstawiona w nim zostanie budowa uniwersalnego dekodera, którego specyfikacja została opracowana w postaci syntezy opisu w języku opisu sprzętu (VHDL). Jego uniwersalność polega na możliwości dostosowania do dowolnego kodu AA-LDPC z jednej strony, a z drugiej strony – możliwości uproszczenia i konfiguracji elementów modułu dekodera, pozwalającej na dostosowanie przepustowości, wymaganych zasobów i jakości dekodowania do wymagań użytkownika. Przedstawione zostaną zatem różnorodne propozycje budowy elementów dekodera. Na etapie projektowania efektywnego dekodera sprzętowego pojawiły się pewne problemy, które wymagały skutecznego rozwiązania. Istotnym elementem rozdziału 4 będzie propozycja rozwiązania problemu negatywnego wpływu reprezentacji danych w postaci słów stałoprzebiegowych o silnie ograniczonej długości na jakość dekodowania. Zaprezentowany zostanie również opracowany sposób optymalizacji przetwarzania potokowego w dekodерze prowadzący do zwiększenia jego przepustowości. Na bazie zaprojektowanego

dekodera, implementowanego w zestawie uruchomieniowym z układem FPGA stworzono środowisko testowania i symulacji systemów kodowania LDPC, które zostanie przedstawione w końcowej części rozdziału. Wszystkie wyniki symulacji zamieszczone w pracy uzyskano wykorzystując wspomniane środowisko.

W rozdziale 5 będą przedstawione metody tworzenia macierzy kontrolnych kodów LDPC należących do klasy AA-LDPC, które mają dobre własności korekcyjne, a jednocześnie mogą być efektywnie dekodowane w zaprojektowanym dekodery sprzętowym. Metody, które zostaną zaprezentowane, są uniwersalne, tzn. umożliwiają tworzenie kodów regularnych, jak i nieregularnych, o dowolnej długości bloku N , stopie kodu R oraz rozmiarze podmacierzy P . Utworzony kod może zostać natychmiast przetestowany, gdyż proces generacji fragmentów opisu dekodera w języku VHDL odpowiadających danemu kodowi AA-LDPC odbywa się automatycznie, za pomocą odpowiednich procedur oprogramowania MATLAB, wykorzystywanego do integracji poszczególnych elementów środowiska testowania i symulacji systemów kodowania LDPC. W końcowej części rozdziału zostanie przedstawiona analiza własności kodów AA-LDPC uzyskanych za pomocą opracowanych algorytmów. Zaprezentowane zostaną wyniki eksperymentów dla kodów o różnych długościach bloku N , stopach R i dystrybucjach stopni wierzchołków grafu oraz porównanie z odpowiednimi kodami LDPC (nie zorientowanymi na implementację) pochodzącymi ze źródeł literaturowych i internetowych oraz kodami otrzymanymi przy wykorzystaniu znanych z literatury metod. Pozwoli to na wyselekcjonowanie najlepszego spośród zaproponowanych algorytmów tworzenia macierzy kontrolnej kodu AA-LDPC oraz udowodnienie tezy pracy.

W rozdziale 6 stanowiącym podsumowanie pracy, zostaną zawarte również możliwe kierunki dalszych prac.

4. Implementacja dekodera

Specyfikacja sprzętowego dekodera kodów LDPC jest zadaniem nietrywialnym i wymaga sporego wysiłku projektanta. Dla uzyskania odpowiedniego efektu konieczne jest staranne rozwiązanie szeregu problemów i dokonanie kilku istotnych wyborów dotyczących budowy dekodera. Dołożono starań, aby zaprojektowany dekodery był uniwersalny z jednej strony, a konkurencyjny do rozwiązań znanych z literatury pod względem przepustowości oraz wymaganych zasobów układu programowalnego z drugiej strony.

Specyfikacja konfigurowalnego dekodera została opracowana w postaci syntezowalnego opisu w języku VHDL. Założono, że moduł dekodera powinien być przystosowany do natychmiastowego wykorzystania w systemie transmisji danych z kodowaniem LDPC. Powinien być również elementem środowiska testowania i symulacji systemów kodowania LDPC, które zostało stworzone dla realizacji celów pracy oraz eksperymentalnego wykazania prawdziwości tezy. Opis modułu dekodera w języku VHDL został zatem opracowany w taki sposób, aby dekodery mógł funkcjonować dla kodów z szerokiej rodziny kodów AA-LDPC, tzn. regularnych i nieregularnych, o dowolnej długości bloku N oraz stopie R . Ważna jest również możliwość wyboru długości słowa oraz jednego spośród wielu algorytmów wyznaczania wiadomości.

W dalszej części rozdziału, po prezentacji klasyfikacji architektur dekodery sprzętowych oraz przeglądu implementacji znanych z literatury, przedstawiono definicję pojęć związanych z kodami AA-LDPC i odpowiadającym im grafem Tannera oraz uzasadnienie celowości ograniczenia rozpatrywanej klasy kodów LDPC do kodów AA-LDPC. W implementacji dekodera zdecydowano się na wykorzystanie algorytmu TDMP (*Turbo Decoding Message Passing*), który jest pewną modyfikacją klasycznych algorytmów przekazywania wiadomości, dzięki której można uzyskać nieco szybszą zbieżność obliczeń. Przedstawiona zostanie architektura dekodera TDMP oraz budowa jednostek obliczeniowych przy zastosowaniu różnych algorytmów wyznaczania wiadomości z wierzchołków kontrolnych. Jak wykazały eksperymenty, poszczególne algorytmy różnią się pod względem wymaganych zasobów i maksymalnej częstotliwości sygnału zegarowego z jednej strony, a jakością dekodowania (bitową stopą błędów systemu) z drugiej strony. Dołożono zatem starań, aby możliwy był

wybór spośród szerokiej gamy algorytmów w zależności od wymagań dla danego systemu. Wybór ten jest wykonywany z poziomu środowiska MATLAB, w którym łączone są odpowiednie fragmenty opisu dekodera w języku VHDL.

W implementacji sprzętowej dekodera, wiadomości reprezentowane są przez liczby stałoprzecinkowe o ograniczonej długości słowa, co jest przyczyną błędów zaokrąglenia oraz błędów obcięcia wartości. Jak się okazało w trakcie prowadzonych prac, błędy te są dosyć znaczące. Dla poprawy skuteczności dekodowania zaproponowano wprowadzenie pewnych modyfikacji struktury jednostek obliczeniowych dekodera, mających na celu zmniejszenie błędów obcięcia wartości wiadomości przesyłanych pomiędzy jednostkami obliczeniowymi w formacie stałoprzecinkowym o małej długości słowa.

Położono również nacisk na uzyskanie jak największej przepustowości dekodera. Wszystkie bloki dekodera sterowane są jednym sygnałem zegarowym. Dla poprawy czytelności sygnał ten nie jest pokazywany na rysunkach, a wszystkie rejestry – typu D – sterowane tym sygnałem oznaczane są poprzez szare prostokąty. W celu zwiększenia maksymalnej częstotliwości sygnału zegarowego w blokach dekodera zastosowano rejestry przetwarzania potokowego. Zastosowanie tych rejestrów wiąże się z powstaniem pewnej liczby cykli bezczynności, dla których dekodery są w stanie oczekiwania na zakończenie iteracji (pojawienia się danych na wyjściu potoku). Dla algorytmu TDMP liczba cykli bezczynności zależy od pewnych zależności pomiędzy kolejnymi wierszami macierzy kontrolnej kodu. W celu minimalizacji liczby tych cykli (a co za tym idzie maksymalizacji przepustowości), opracowano algorytm sortowania wierszy (i kolumn) macierzy bazowej kodu AA-LDPC. Jak zostanie pokazane na przykładach, przekształcenie macierzy z wykorzystaniem tego algorytmu umożliwi znaczne zmniejszenie liczby cykli bezczynności, w niektórych przypadkach prawie do zera.

Aby umożliwić szybką weryfikację układu dekodera oraz eksperymentalne wyznaczenie własności kodów AA-LDPC stworzono środowisko testowania i symulacji systemów kodowania LDPC. Dekoder implementowany jest w zestawie uruchomieniowym z układem rodziny VirtexII firmy Xilinx, a pozostałe elementy systemu transmisyjnego (koder, modulator, demodulator i kanał transmisyjny) modelowane są w środowisku MATLAB. Opracowane środowisko umożliwia wykonywanie symulacji ze znacznie większą szybkością niż przy wykorzystaniu programowego modelu dekodera pracującego np. na komputerze PC. Wszystkie wyniki symulacji zamieszczone w pracy pochodzą z opracowanego środowiska.

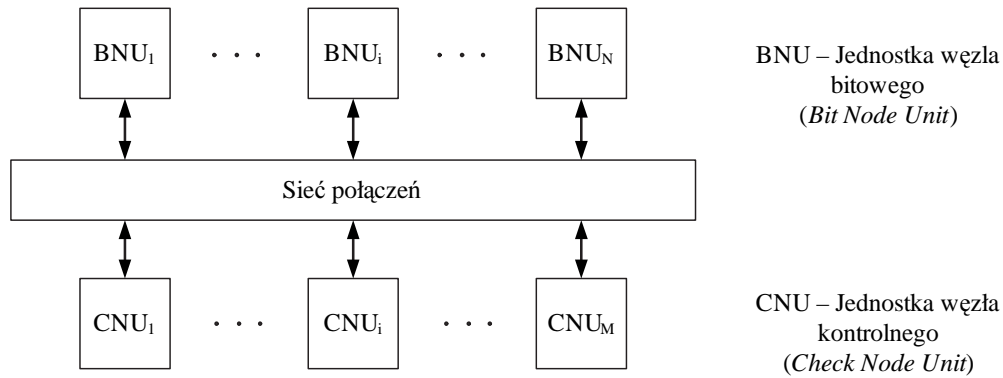
Dla wykazania konkurencyjności dekodera do rozwiązań znanych z literatury, na końcu rozdziału przedstawione zostaną wyniki implementacji (przepustowość, wy-

magane zasoby) dla przykładowych kodów oraz porównanie z implementacjami znanymi z literatury. Na podstawie zgromadzonych wyników eksperymentów, wybrano strukturę jednostek obliczeniowych dekodera do eksperymentów przeprowadzanych dla celów wyznaczania własności korekcyjnych kodów AA-LDPC, prezentowanych w kolejnym rozdziale.

4.1 Klasyfikacja architektur dekodera

Zasadniczymi elementami sprzętowej implementacji dekodera są jednostki obliczeniowe realizujące funkcje wierzchołków grafu Tannera. Ze względu na sposób alokacji obliczeń do konkretnych jednostek można wyróżnić 3 podstawowe architektury dekodera: równoległą, szeregową i szeregowo-równoległą.

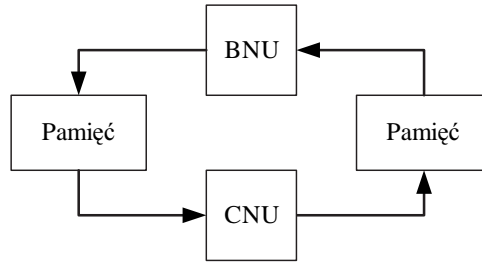
Architektura równoległa (rys. 4.1) odpowiada wprost grafowi Tannera danego kodu. Dekoder składa się z N jednostek obliczeniowych realizujących operacje węzła bitowego (BNU - *Bit Node Unit*) oraz M jednostek obliczeniowych realizujących operacje węzła kontrolnego (CNU - *Check Node Unit*). Komunikacja pomiędzy jednostkami odbywa się za pomocą sieci połączeń odpowiadającej bezpośrednio strukturze grafu Tannera kodu.



Rys. 4.1: Równoległa architektura dekodera

W **architekturze szeregowej** (rys. 4.2) do realizacji obliczeń wykorzystywana jest jedna jednostka BNU i jedna jednostka CNU. Operacje odpowiadające kolejnym węzłom bitowym (kontrolnym) wykonywane są szeregowo. Zamiast sieci połączeń stosowane są dwie pamięci, pierwsza dla wiadomości przesyłanych od węzłów bitowych do kontrolnych i druga dla wiadomości przesyłanych w odwrotnym kierunku. Wiadomości odczytywane są z pamięci szeregowo. W tym przypadku ar-

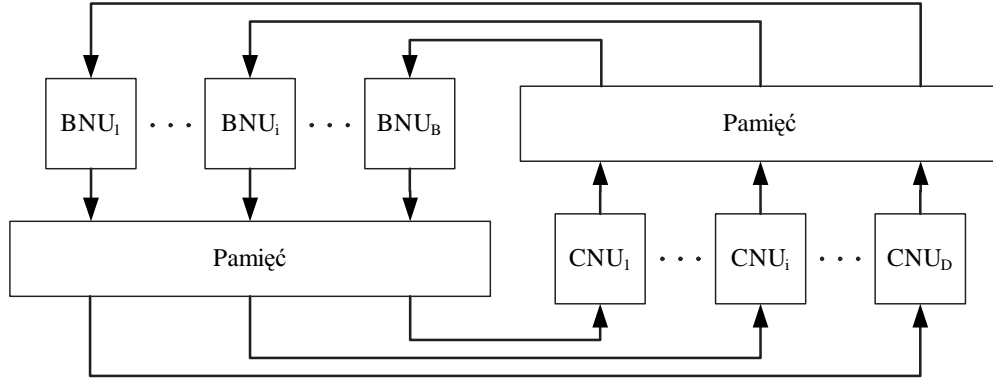
chitektura dekodera ma stosunkowo małą złożoność, jednak czas wykonania jednej iteracji dekodowania, ze względu na współdzielenie jednej jednostki obliczeniowej, jest stosunkowo duży, a przepustowość dekodera zbyt mała w stosunku do typowych wymagań.



Rys. 4.2: Szeregowa architektura dekodera

W celu uzyskania odpowiedniej przepustowości przy dużej długości bloku kodu oraz ograniczonej złożoności układu dekodera, niezbędne jest zastosowanie **architektury szeregowo-równoległej**, przedstawionej na rys. 4.3. Zdecydowana większość implementacji dekoderek kodów LDPC opisanych w literaturze ma właśnie taką architekturę. Podstawowym problemem wiążącym się bezpośrednio z architekturą szeregowo-równoległą jest konflikt dostępu jednostek do pamięci wiadomości. Dla macierzy kontrolnej kodu o całkowicie nieuporządkowanej strukturze (czyli z jedynkami rozmieszczonymi niezależnie w losowy sposób), wszystkie jednostki obliczeniowe powinny mieć równocześnie zagwarantowany dostęp do całego obszaru pamięci wiadomości. Wiąże się to z koniecznością zastosowania pamięci wieloportowych, które są nieosiągalne we współczesnych układach FPGA (przykładowo układ FPGA rodziny VirtexII zawiera pamięci jedynie dwuportowe). Zaprojektowanie dekodera o architekturze szeregowo-równoległej bez korzystania z pamięci wieloportowych jest możliwe, jeśli na strukturę macierzy kontrolnej kodu zostaną narzucone pewne ograniczenia. Dla takich kodów pamięć wiadomości może być bądź to podzielona na mniejsze bloki (typowo na każdą jednostkę obliczeniową przypada wówczas jeden blok pamięci), bądź też słowo pamięci powinno być odpowiednio wydłużone (w jednym słowie zawarta jest wówczas pewna liczba wiadomości dostarczanych równoległe do poszczególnych jednostek obliczeniowych). Efektywne implementacje modułów kodera i dekodera wykorzystują zatem pewną podklasę kodów LDPC — najczęściej kody o macierzy kontrolnej o strukturze blokowej (*Block-Structured LDPC*, [126]), zwane też AA-LDPC (*Architecture-Aware LDPC*, [67]), *Array Based LDPC* [74, 95] bądź QC-LDPC (*Quasi-Cyclic LDPC*). Kody takie nazywane są w niniejszej pracy

kodami zorientowanymi na implementację bądź kodami AA-LDPC.



Rys. 4.3: Szeregowo-równoległa architektura dekodera

4.2 Definicja kodu AA-LDPC

Macierz kontrolna kodu zorientowanego na implementację składa się typowo z pewnej liczby podmacierzy będących macierzami zerowymi, bądź też permutacjami kolumn macierzy jednostkowej lub pewną sumą permutacji kolumn macierzy jednostkowych. Występują pewne różnice w definicjach kodów zorientowanych na implementację przedstawianych w literaturze, stąd też konieczne jest uściślenie pojęcia takiego kodu. Klasa kodów o macierzy kontrolnej zdefiniowanej w niniejszym podrozdziale nazywana jest kodami AA-LDPC.

Permutacją macierzy jednostkowej $\mathbf{I}_{P \times P}$, oznaczaną $\mathbf{I}_{\theta(p)}$, jest macierz kwadratowa składająca się z wszystkich kolumn macierzy jednostkowej, ułożonych w dowolnej kolejności (definiowanej przez $\theta(p)$). Permutacja macierzy jednostkowej zawiera zatem dokładnie jedną jedynkę w każdej kolumnie i dokładnie jedną jedynkę w każdym wierszu.

Macierz kontrolna $\mathbf{H}_{M \times N} = \mathbf{H}_{DP \times LP}$ kodu AA-LDPC to macierz składająca się z $D \times L$ podmacierzy kwadratowych $\mathbf{P}_{d,l}$, $d = 1, \dots, D$, $l = 1, \dots, L$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \cdots & \mathbf{P}_{1,L} \\ \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \cdots & \mathbf{P}_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{D,1} & \mathbf{P}_{D,2} & \cdots & \mathbf{P}_{D,L} \end{bmatrix} \quad (4.1)$$

gdzie każda z podmacierzy $\mathbf{P}_{d,l}$ o rozmiarze $P \times P$ jest macierzą zerową ($\mathbf{P}_{d,l} = \mathbf{0}_{P \times P}$) lub też permutacją macierzy jednostkowej ($\mathbf{P}_{d,l} = \mathbf{I}_{\theta(p)}$). Macierz kontrolna zawiera zatem $M = DP$ wierszy, $N = LP$ kolumn.

Macierz bazowa (*base matrix*) \mathbf{W} kodu AA-LDPC to macierz o rozmiarze $D \times L$, w której element $w_{d,l} = 1$ wtedy i tylko wtedy, gdy $\mathbf{P}_{d,l}$ macierzy kontrolnej \mathbf{H} jest permutacją macierzy jednostkowej; w przeciwnym razie $w_{d,l} = 0$. Graf bazowy to graf Tannera odpowiadający macierzy bazowej.

Zbiór wierzchołków kontrolnych grafu Tannera kodu AA-LDPC można podzielić na podzbiory: $V_c = V_c^1 \cup V_c^2 \cup \dots \cup V_c^D$, a licznosc każdego z nich wynosi $|V_c^d| = P$. Podobnie zbiór wierzchołków bitowych składa się z podzbiorów: $V_b = V_b^1 \cup V_b^2 \cup \dots \cup V_b^L$. Każdy z wierzchołków należących do podzbioru V_c^d jest połączony krawędzią z dokładnie jednym wierzchołkiem z podzbioru V_b^l wtedy i tylko wtedy, gdy $w_{d,l} = 1$. Jeśli $w_{d,l} = 0$, to każdy z wierzchołków należących do podzbioru V_c^d nie jest połączony z żadnym z wierzchołków podzbioru V_b^l . Dzięki temu możliwe jest zaproponowanie struktury dekodera, w której do każdego wierzchołka z podzbioru V_c^d (V_b^l) przyporządkowana jest konkretna jednostka obliczeniowa, realizująca funkcje tego wierzchołka. Ponieważ każdy z wierzchołków podzbioru V_c^d połączony jest krawędzią z co najwyżej jednym wierzchołkiem podzbioru V_b^l , możliwa jest eliminacja konfliktów dostępu do pamięci wiadomości.

4.3 Przegląd implementacji dekodera znanych z literatury

W niniejszym podrozdziale przedstawiono przegląd znanych z literatury sprzętowych realizacji dekodera LDPC. Na tej podstawie sformułowano pewne ogólne wnioski co do obecnego stanu prac naukowych w tej dziedzinie oraz postawiono wymagania dla projektowanego modułu dekodera realizowanego w strukturze FPGA.

Nieliczne znane prace, w których skorzystano z równoległej architektury dekodera dotyczą kodów o stosunkowo małej długości bloku (N nie większe niż 1000). Przykładowo w [46] zaproponowano dekodery o architekturze równoległej dla pewnego specyficznego kodu (regularnego, o rzadkiej macierzy generującej) o długości bloku $N = 576$ i stopie $R = 8/9$. W [4, 36] zaprezentowano dekodery równoległe dla kodu o bloku $N = 1024$, stopie $R = 1/2$, który w układzie $0.16\mu\text{m}$ CMOS, przy powierzchni układu 50mm^2 osiąga przepustowość 1Gb/s . Głównymi modułami dekodera są jednostki węzłów bitowych realizujących sumowanie, jednostki węzłów kontrolnych wyznaczające minimum (algorytm Min-Sum) oraz rejestry przechowujące wiadomości. Jak zauważyli autorzy owych publikacji, głównym problemem im-

plementacji dekodera było poprowadzenie połączeń pomiędzy jednostkami (około 26000 linii). Liczba połączeń rośnie z kwadratem długości bloku N . Nie są znane w pełni równoległe implementacje dekodera dla kodów o długości bloku większej niż 1024 bitów.

W pracy [56] przedstawiono dekodery szeregowy dowolnego kodu LDPC. Dane z pamięci wiadomości pobierane i zapisywane są szeregowo przez pojedyncze jednostki węzła bitowego i kontrolnego. Przepustowość dekodera wynosi 1Mb/s (10 iteracji dekodowania) w układzie FPGA, przy częstotliwości sygnału zegarowego 50MHz. Jedynym sposobem zwiększenia przepustowości dla tej architektury jest zwiększenie częstotliwości sygnału zegarowego, stąd osiągnięcie znacznie większej przepustowości jest niemożliwe.

Zdecydowana większość struktur dekoderek opisanych w literaturze ma architekturę szeregowo-równoległą (ang. *partially-parallel*). Duża część prac dotyczy jednak tylko kodów regularnych, inne prace dotyczą pewnych specyficznych kodów (będących podzbiorem kodów AA-LDPC) lub kodów zdefiniowanych w konkretnym standardzie transmisji. Przykładowo w [23] przedstawiono dekodery przeznaczony dla kodów wykorzystanych w standardzie telewizji cyfrowej DVB-S2 [28] (rozmiar podmacierzy 360×360). Dekoder składa się z konfigurowalnej liczby ścieżek danych (ang. *data path*) i osiąga maksymalną przepustowość 90Mb/s przy 30 iteracjach i częstotliwości zegarowej 300MHz. Wykonany jest w technologii CMOS 90nm, a jego powierzchnia wynosi 4.1mm^2 . Dekoder kodów zdefiniowanych w standardzie DVB-S2 jest przedstawiony również w [45]. Dekoder składa się z 360 jednostek realizujących operacje wierzchołków kontrolnych i bitowych, pamięci i konfigurowalnej sieci połączeń. Przepustowość dekodera wynosi 255Mb/s, przy 30 iteracjach i zajmowanej powierzchni 22.7mm^2 .

W [123] przedstawiono implementację dekodera kodów regularnych o stopniu wierzchołków bitowych $d_b = 3$. Obsługiwane są kody o silniejszych ograniczeniach niż typowo zakładane dla kodów AA-LDPC: macierz kontrolna składa się z trzech podmacierzy o stopniach wszystkich kolumn równych 1. Osiągnięto przepustowość 54Mb/s dla 18 iteracji, przy zastosowaniu struktury FPGA rodziny VirtexII. Praca [95] dotyczy również pewnych szczególnych kodów regularnych. Przedstawiono w niej szeregowo-równoległą implementację dekodera kodów o stopniach wierzchołków $d_b = 3$, $d_c = 6$. Macierz kontrolna kodu złożona jest z $3 \times 6 = 18$ podmacierzy będących cyklicznym przesunięciem macierzy jednostkowej. Podmacierze mają rozmiar nie większy niż 256×256 , stąd maksymalna długość bloku kodu wynosi $N = 1536$. Implementacje dekoderek kodów regularnych w układach FPGA przedstawiono również w [112] (konkretny kod QC-LDPC, implementacja w FPGA firmy

Xilinx), [5] (dekoder kodów zdefiniowanych w [75]) oraz [51] (kod QC-LDPC, implementacja w FPGA firmy Altera).

Implementację dekodera kodów regularnych w układzie typu ASIC zaprezentowano w [68]. Dekoder osiąga przepustowość 640Mb/s przy założeniu 10 iteracji. Implementacje o zbliżonych parametrach w układach typu ASIC przedstawiono również w [88], [94] (mniejsza przepustowość, ale i mniejszy pobór mocy) oraz w [104] (przepustowość 1Gb/s w technologii CMOS 0.13 μ m).

W [13] przedstawiono dekodek nieregularnego kodu strukturalnego o bloku $N = 8088$, stopie $R = 1/2$ i ściśle określonej dystrybucji stopni wierzchołków. Klasa takich kodów nazywana jest przez autorów kodami IPP (*Irregular Partitioned Permutation*, [34]). Dekoder kodów nieregularnych o pewnych określonych wartościach stopni wierzchołków ($d_b = 2, 3, 7$, $d_c = 5, 6, 8, 24$) zaprezentowano również w [117, 118]. Przedstawiono tam wyniki implementacji w układzie FPGA firmy Xilinx. Ideę budowy uniwersalnego dekodera szeregowo-równoległego składającego się z pamięci, matrycy połączeń i konfigurowalnej liczby modułów obliczeniowych opisano w [111]. Zaprezentowano implementację w układach FPGA firmy Altera.

Praca [103] dotyczy dekodera kodów o wysokiej stopie R , stosowanych w systemach zapisu danych na nośnikach magnetycznych. Proponowana struktura dekodera została przedstawiona bardziej szczegółowo w [102]. Dekodera, jak również koder zaimplementowano w układzie FPGA XC2V6000.

W [124] przedstawiono implementację dekodera kodów RS-LDPC (Reed-Solomon LDPC; kod regularny o rozmiarach macierzy kontrolnej $M = 384$, $N = 2048$). Budowa dekodera jest dosyć specyficzna: zaproponowano wykorzystanie pojedynczej jednostki węzła kontrolnego pobierającej wiadomości z pamięci w sposób równoległy oraz 32 jednostek węzła bitowego pobierających i zapisujących wiadomości szeregowo.

Szczegółowe dane dotyczące najlepszych z opisanych w literaturze struktur dekodów, w szczególności implementacji dekodera w układach FPGA, przedstawiono na końcu rozdziału w tabelach 4.2-4.3.

Wiele spośród opublikowanych implementacji dekodera dotyczy pewnego specyficznego kodu lub też pewnego podzbioru kodów AA-LDPC. Jednym z celów niniejszej pracy jest zaprojektowanie uniwersalnego modułu dekodera o jak najszerszym zakresie zastosowań. Jego uniwersalność powinna umożliwić prowadzenie prac związanych z poszukiwaniem kodów AA-LDPC o dobrych właściwościach korekcyjnych, przeprowadzenie eksperymentów mających na celu ustalenie wpływu ograniczonej precyzji wiadomości na jakość dekodowania oraz weryfikację zaproponowanych pomysłów modyfikacji algorytmów dekodowania mających na celu zmniejszenie owego

wpływu. Stąd też projektowany konfigurowalny moduł dekodera powinien:

- mieć postać behawioralnego opisu w języku opisu sprzętu zapewniającego przenaszalność na platformy sprzętowe (FPGA) różnych producentów;
- mieć możliwość obsługi kodów AA-LDPC regularnych, jak i nieregularnych, o dowolnych stopach R oraz szerokim przedziale długości bloku N ;
- zapewniać możliwość wyboru algorytmu dekodowania (w szczególności algorytmu wyznaczania wiadomości w wierzchołkach kontrolnych) oraz dokładności obliczeń – czyli długości stałoprzecinkowego słowa wiadomości (*performance – complexity trade-off*);
- zapewniać możliwość doboru liczby jednostek obliczeniowych stosownie do wymagań co do przepustowości dekodera (*throughput – complexity trade-off*);
- być konkurencyjny w stosunku do rozwiązań znanych z literatury pod względem przepustowości oraz wymaganych zasobów układu programowalnego.

4.4 Algorytm TDMP

Zdecydowano, że struktura dekodera będzie oparta na nieco odmiennym niż klasyczne sposoby przekazywania wiadomości, określanym w publikacjach jako algorytm TDMP (*Turbo Decoding Message Passing*, [65]). Wybór ten został podyktowany pewnymi lepszymi własnościami algorytmu TDMP w porównaniu z klasycznymi metodami przekazywania wiadomości, tzn. [65, 99]:

- szybszą zbieżnością obliczeń,
- zmniejszoną wymaganą pojemnością pamięci,
- obecnością w strukturze dekodera tylko jednego typu jednostek obliczeniowych, co upraszcza proces automatycznej generacji opisu dekodera w języku opisu sprzętu.

Główna zaleta to szybsza zbieżność obliczeń wynikająca z faktu, że algorytm TDMP jest równoważny dekodowaniu z zastosowaniem tzw. harmonogramu szeregowego, który ogólnie rzecz biorąc polega na propagacji wiadomości do sąsiadujących węzłów natychmiast po obliczeniu ich nowych wartości. Metoda ta zostanie wyjaśniona w kolejnym podrozdziale, co umożliwi zaprezentowanie formalnego opisu zaimplementowanego w opracowanym dekodерze algorytmu TDMP.

4.4.1 Dekodowanie z zastosowaniem harmonogramu szeregowego

Odpowiedni harmonogram (*scheduling*) wymiany wiadomości pomiędzy jednostkami BNU i CNU umożliwia poprawę stosunku przepustowości do złożoności dekodera poprzez zmniejszenie średniej liczby iteracji procesu dekodowania. Typowe algorytmy wymiany wiadomości, przedstawione w rozdziale 2.2, opierają się na standardowym harmonogramie (określanym w literaturze jako *flooding schedule*). Przy zastosowaniu tego harmonogramu wszystkie wiadomości z węzłów kontrolnych do bitowych wyznaczone są równocześnie, a w następnej kolejności równocześnie wszystkie wiadomości z węzłów bitowych do kontrolnych (jak np. w algorytmie 2.3). Alternatywne harmonogramy to tzw. harmonogramy szeregowy (*serial scheduling*), w których zakłada się, że obliczenia dla poszczególnych węzłów realizowane są kolejno, a wiadomości propagowane są do sąsiadujących węzłów natychmiast po ich wyznaczeniu. Harmonogram tego typu może być zrealizowany w szeregowej lub szeregowo-równoległej architekturze dekodera (w architekturze równoległej jest to niemożliwe). W projektowanym dekodерze zastosowano harmonogram szeregowy, ze względu na możliwość uzyskania szybszej zbieżności obliczeń (dzięki natychmiastowej propagacji wiadomości do sąsiadujących węzłów uzyskuje się mniejszą średnią liczbę wymaganych iteracji).

Można wyróżnić dwa podstawowe harmonogramy szeregowy [92]: szeregowy-C (*serial-C*) i szeregowy-V (*serial-V*). Dla harmonogramu szeregowego-C zakłada się, że poszczególne węzły kontrolne realizują obliczenia w pewnej założonej kolejności. Do każdego węzła kontrolnego $c_m \in V_c$ przesyłane są wiadomości Q_{nm} , $n \in \mathcal{N}(m)$. Węzeł wyznacza wiadomości wyjściowe R_{mn} , $n \in \mathcal{N}(m)$ i przesyła je do węzłów bitowych b_n , gdzie na tej podstawie uaktualniane są natychmiastowo wiadomości wyjściowe. Analogicznie dla harmonogramu szeregowego-V, węzły bitowe realizują obliczenia w pewnej kolejności, a wiadomości wyznaczone w tych węzłach są natychmiast wykorzystywane do uaktualnienia wiadomości w węzłach kontrolnych.

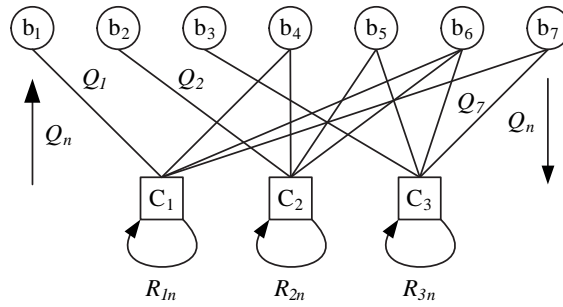
Harmonogram szeregowy-C przedstawiony jest skrótowo jako algorytm 4.1. Wartość LLR prawdopodobieństw *pseudo-posteriori* dla bitu n -tego, oznaczona przez Q_n , wyznaczana jest na podstawie informacji z wszystkich wierzchołków kontrolnych sąsiadujących z wierzchołkiem b_n . W wierszu 5 algorytmu 4.1 wyznaczone są wiadomości wejściowe Q_{nm} węzła kontrolnego c_m , równe różnicy wartości Q_n i wiadomości R_{mn} wyznaczonych w poprzedniej iteracji. Inaczej mówiąc, z pełnej informacji o wartości bitu n -tego skumulowanej w Q_n , usuwana jest porcja informacji wyznaczona w poprzedniej iteracji przez jednostkę wierzchołka kontrolnego c_m , dla którego wykonywane są aktualnie obliczenia. W wierszu 6 algorytmu wy-

znaczane są nowe wartości wiadomości R_{mn} , za pomocą funkcji oznaczonej przez $CN(\{Q_{n'm}\})$ (CN – *Check Node*). Funkcja ta odpowiada jednemu z algorytmów wyznaczania wiadomości w wierzchołku kontrolnym, przedstawionych w rozdziale 2.2 (np. wzór (2.19) dla podstawowego algorytmu LLR-BP). Następnie (wiersz 7 algorytmu 4.1) uaktualniane są prawdopodobieństwa *pseudo-posteriori* dla wszystkich bitów odpowiadających wierzchołkom sąsiadującym z wierzchołkiem c_m . Do obliczeń dla kolejnego (c_{m+1}) wierzchołka kontrolnego wykorzystywane są wartości Q_n uaktualnione przez węzeł c_m .

Algorytm 4.1: Algorytm LLR-BP z harmonogramem szeregowym-C	
Dane wejściowe: Logarytmiczne stosunki prawdopodobieństw:	
$\delta_n = L(x_n y_n) = \ln \left(\frac{P(x_n=0 y_n)}{P(x_n=1 y_n)} \right), n = 1, \dots, N$	
Dane wyjściowe: Wektor kodowy $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \dots, \hat{x}_N]$	
1	<i>Inicjalizacja.</i> $i := 0; Q_n := \delta_n; R_{mn} := 0; \quad n = 1, \dots, N \quad m \in \mathcal{M}(n)$
2	POWTARZAJ
3	$i := i + 1$
4	DLA $m = 1, \dots, M$ (dla każdego wierzchołka kontrolnego)
5	$Q_{nm} := Q_n - R_{mn}, \quad n \in \mathcal{N}(m)$
6	$R_{mn} := CN(\{Q_{n'm} : n' \in \mathcal{N}(m) \setminus n\}), \quad n \in \mathcal{N}(m)$
7	$Q_n := Q_{nm} + R_{mn}, \quad n \in \mathcal{N}(m)$
8	$\hat{x}_n := \frac{1}{2} (\text{sgn}(\gamma_n) + 1) \quad n = 1, \dots, N$
9	AŻ $(\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0})$ lub $(i = i_{\max})$
10	JEŻELI $\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$
11	Wektor $\hat{\mathbf{x}}$ jest wektorem zdekodowanym.
12	W PRZECIWNYM WYPADKU
13	Dekodowanie nie zakończone sukcesem

Należy zauważyć, że przy zastosowaniu algorytmu 4.1, funkcja węzłów bitowych może zostać ograniczona do przechowywania wartości Q_n . Każdemu węzłowi bitowemu odpowiada wówczas pojedyncza komórka pamięci. Natomiast każdy węzeł kontrolny c_m odpowiada za przechowywanie wartości wiadomości R_{mn} , $n \in \mathcal{N}(m)$ oraz wykonywanie obliczeń (zależności w wierszach 5-7 algorytmu 4.1). Sposób przesyłania wiadomości dla przykładowego grafu Tannera przedstawiono na rys. 4.4.

Podstawową korzyścią wynikającą z zastosowania jednego z harmonogramów szeregowych jest szybsza zbieżność obliczeń. Wykazano [92, 93], że przy zastosowaniu harmonogramu szeregowego można osiągnąć około dwukrotnie mniejszą średnią liczbę iteracji wymaganą do zdekodowania pojedynczego słowa. Dzięki temu, zmniejsza-



Rys. 4.4: Schemat propagacji wiadomości dla algorytmu 4.1

jąc maksymalną liczbę iteracji i_{\max} , uzyskać można większą przepustowość przy niezmięonej bitowej stopie błędów lub też (przy niezmięonym i_{\max}) uzyskać poprawę bitowej stopy błędów przy niezmięonej przepustowości. Dodatkową korzyścią wynikającą z zastosowania harmonogramu zgodnego z algorytmem 4.1 jest zmniejszenie wymaganej pojemności pamięci wiadomości: wykorzystywane jest N komórek pamięci dla wiadomości przesyłanych do węzłów bitowych zamiast około $d_b N$ komórek pamięci dla algorytmu standardowego.

Oprócz harmonogramu standardowego oraz harmonogramów szeregowych można wyróżnić również harmonogramy semi-szeregowy (ang. *semi-serial schedule* [92, 121]), dla których zbiór węzłów dzielony jest na pewną liczbę grup. Węzły w ramach grupy realizują obliczenia równolegle, a poszczególne grupy węzłów kolejno uaktualniają wiadomości wyjściowe. W literaturze proponuje się także harmonogramy probabilistyczne, dla których wartości wiadomości w danej iteracji są uaktualniane bądź pozostawiane bez zmian z pewnym prawdopodobieństwem. W pracy [70] zaproponowano uaktualnianie wiadomości z węzła bitowego b_n z prawdopodobieństwem zależnym od długości najmniejszego cyklu zawierającego węzeł b_n w grafie Tannera. Znana jest również [55] modyfikacja harmonogramu szeregowego-V polegająca na uaktualnianiu wiadomości z węzła bitowego b_n z prawdopodobieństwem zależnym od numeru iteracji oraz modułu wartości wejściowej δ_n . Autorzy wspomnianych wyżej prac stawiają tezę, iż zastosowanie tego typu harmonogramów zmniejsza złożoność obliczeniową (liczbę wymaganych uaktualnień wiadomości) przy niezmięonej skuteczności dekodowania. Uzyskanie praktycznych korzyści z zastosowania harmonogramów probabilistycznych w sprzętowej implementacji dekodera o architekturze szeregowo-równoległej wydaje się jednak problematyczne.

4.4.2 Dekodowanie z zastosowaniem algorytmu TDMP

Zdecydowano, że projektowany dekodery oparty będzie na algorytmie TDMP [65], gdyż dla kodów AA-LDPC jest on równoważny algorytmowi z harmonogramem szeregowym-C, a jednocześnie umożliwia skonstruowanie dekodera o architekturze szeregowo-równoległej.

W algorytmie TDMP zakłada się, że macierz kontrolna $\mathbf{H}_{M \times N}$ kodu \mathcal{C} składa się z pewnej liczby podmacierzy $\mathbf{H}^1, \dots, \mathbf{H}^D$ o rozmiarach $P \times N$, tzn. $\mathbf{H} = [\mathbf{H}^1, \dots, \mathbf{H}^D]^T$. Każda z podmacierzy \mathbf{H}^d jest macierzą kontrolną kodu \mathcal{C}^d , zwanego super-kodem. Poprawne słowo kodowe należące do kodu \mathcal{C} należy również do każdego z super-kodów (muszą być spełnione równania kontrolne każdego z super-kodów), a zatem $\mathcal{C} = \mathcal{C}^1 \cap \mathcal{C}^2 \cap \dots \cap \mathcal{C}^D$. W algorytmie TDMP pojedyncza iteracja dekodowania składa się z D subiteracji (ang. *sub-iteration*, [65]), a każda subiteracja odpowiada jednemu z super-kodów. W subiteracji d uaktualniane są wartości LLR prawdopodobieństw *pseudo-posteriori* (Q_n) przy założeniu, że słowo należy do kodu \mathcal{C}^d , przyjmując jako informację *a priori* sumę wartości otrzymanych w poprzednich subiteracjach z wszystkich super-kodów oprócz super-kodu \mathcal{C}^d .

Jeśli w każdej z podmacierzy \mathbf{H}^d wszystkie kolumny mają wagę równą co najwyżej 1 (czyli każdy z bitów słowa kodowego jest zawarty w co najwyżej jednym równaniu kontrolnym kodu \mathcal{C}^d), algorytm TDMP jest równoważny algorytmowi z harmonogramem szeregowym-C. Warunek ten jest spełniony, jeśli kod jest kodem AA-LDPC o macierzy kontrolnej jak w (4.1), gdzie $\mathbf{H}^d = [\mathbf{P}_{d,1} \ \mathbf{P}_{d,2} \ \dots \ \mathbf{P}_{d,L}]$. Waga wszystkich wierszy podmacierzy \mathbf{H}^d jest taka sama i wynosi d_{c^d} . Formalny zapis algorytmu TDMP dla tego przypadku przedstawiono jako algorytm 4.2.

Wiadomościami z wierzchołków kontrolnych R_{mn} są wiadomości oznaczone przez $\lambda_n^{d,p}$, gdzie $d = 1, \dots, D$ to indeks podmacierzy (super-kodu), a $p = 1, \dots, P$ to indeks wiersza w ramach tej podmacierzy (numer węzła kontrolnego w podzbiorze V_c^d). Tak więc $\lambda_n^{d,p}$ to wartość LLR prawdopodobieństw dla bitu n -tego przy założeniu, że spełnione jest p -te równanie kontrolne super-kodu \mathcal{C}^d , przyjmując jako informację *a priori* sumę wartości wiadomości otrzymanych w poprzednich subiteracjach z wszystkich super-kodów oprócz super-kodu \mathcal{C}^d . $\mathcal{N}(d, p)$ oznacza zbiór indeksów wierzchołków bitowych sąsiadujących z wierzchołkiem p -tym super-kodu \mathcal{C}^d . Wiadomości wejściowe do P wierzchołków kontrolnych, dla których prowadzone są obliczenia w danej subiteracji, oznaczane przez ρ_n^p (4.3), równe są różnicy wartości LLR prawdopodobieństw *pseudo-posteriori* Q_n i wiadomości $\lambda_n^{d,p}$ wyznaczonych w poprzedniej iteracji. Wiadomości wyjściowe Λ_n^p wyznaczane są zgodnie z (4.4), za pomocą jednego z algorytmów wyznaczania wiadomości w węźle kontrolnym przed-

Algorytm 4.2: Algorytm TDMP

Dane wejściowe: Logarytmiczne stosunki prawdopodobieństw:

$$\delta_n = L(x_n|y_n) = \ln \left(\frac{P(x_n=0|y_n)}{P(x_n=1|y_n)} \right), \quad n = 1, \dots, N$$

Dane wyjściowe: Wektor kodowy $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \dots, \hat{x}_N]$

Zmienne: Wiadomości: $\lambda_n^{d,p}$, $d = 1, \dots, D$, $p = 1, \dots, P$, $n \in \mathcal{N}(d,p)$

Zmienne: LLR prawdopodobieństw *pseudo-posteriori*: Q_n , $n = 1, \dots, N$

1 *Inicjalizacja.*

$$i := 0 \quad (4.2a)$$

$$\lambda_n^{d,p} := 0 \quad d = 1, \dots, D, \quad p = 1, \dots, P, \quad n \in \mathcal{N}(d,p) \quad (4.2b)$$

$$Q_n := \delta_n \quad n = 1, \dots, N \quad (4.2c)$$

2 **POWTARZAJ**

3 $i := i + 1$

4 **DLA** $d = 1, \dots, D$ (dla każdego super-kodu)

5 **DLA** $p = 1, \dots, P$ (dla każdego węzła kontrolnego super-kodu C^d)

6 *Pobranie wiadomości i wyznaczenie wiadomości zewnętrznych ρ_n^p :*

$$\rho_n^p := Q_n - \lambda_n^{d,p} \quad n \in \mathcal{N}(d,p) \quad (4.3)$$

7 *Wyznaczenie wiadomości wyjściowych:*

$$\Lambda_n^p := \text{CN}(\{\rho_n^p : n \in \mathcal{N}(d,p) \setminus n\}) \quad (4.4)$$

8 *Zapisanie uaktualnionych wiadomości:*

$$\lambda_n^{d,p} := \Lambda_n^p \quad n \in \mathcal{N}(d,p) \quad (4.5a)$$

$$Q_n := \rho_n^p + \Lambda_n^p \quad n \in \mathcal{N}(d,p) \quad (4.5b)$$

9 *Próbné decyzje. Dla $n = 1, \dots, N$:*

$$\hat{x}_n := \frac{1}{2} (\text{sgn}(Q_n) + 1) \quad (4.6)$$

10 **AŻ** ($\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$) lub ($i = i_{\max}$)

11 **JEŻELI** $\mathbf{H}\hat{\mathbf{x}}^T = \mathbf{0}$

12 \lfloor Wektor $\hat{\mathbf{x}}$ jest wektorem zdekodowanym.

13 **W PRZECIWNYM WYPADKU**

14 \lfloor Dekodowanie nie zakończone sukcesem

stawionych w rozdziale 2.2. Wiadomości te zapisywane są jako nowe wartości $\lambda_n^{d,p}$ (4.5a) oraz służą do uaktualnienia prawdopodobieństw *pseudo-posteriori* Q_n (4.5b).

Algorytm TDMP został przedstawiony w publikacjach [65–67], a implementacja dekodera w układzie typu ASIC została zaprezentowana w [68]. Przedstawiony tam dekodery jest zaprojektowany dla kodu AA-LDPC o stopniach wierzchołków kontrolnych $d_c = 6$. Architektura dekodera pozwala na obsługę kodów regularnych o parzystych stopniach wierzchołków kontrolnych (jest to związane z faktem, że jednostki obliczeniowe pobierają i wyznaczają po 2 wiadomości równocześnie).

W odróżnieniu od dekodera opisanego w [68], w zaprojektowanym dekoderyze jednostki obliczeniowe, pracujące w oparciu o algorytm wyznaczania wiadomości ze schematem podwójnej rekurencji, pobierają po jednej wiadomości na cykl zegarowy. Dzięki temu dekodery jest bardziej uniwersalny, gdyż może być wykorzystany dla kodu o dowolnych stopniach wierzchołków kontrolnych. Jest zatem przystosowany do dekodowania szerokiej gamy kodów AA-LDPC regularnych jak i nieregularnych, o dowolnych stopniach wierzchołków, stopach kodu R i długościach bloku N .

4.5 Format wartości wiadomości w dekoderyze

W przedstawionych algorytmach dekodowania domyślnie zakładano, że wartości wiadomości są ze zbioru liczb rzeczywistych, tzn. $Q_n \in \mathbb{R}$, $\lambda_n^{d,p} \in \mathbb{R}$. W implementacji sprzętowej dekodera, wiadomości reprezentowane są przez liczby stałoprzecinkowe o długości słowa B . Dane inicjalizujące, jak również wiadomości mogą przyjąć jedną z $2^B - 1$ wartości należących do zbioru, który zostanie oznaczony przez $\mathcal{O}_{B,Z_{th}}$:

$$\mathcal{O}_{B,Z_{th}} = \{0, \pm\Delta, \pm2\Delta, \dots, \pm Z_{th}\}, \quad \Delta = \frac{Z_{th}}{2^{B-1} - 1} \quad (4.7)$$

gdzie Δ to krok kwantyzacji wiadomości, a Z_{th} to próg obcięcia wartości wiadomości (ang. *threshold*). Dany zbiór \mathcal{O} charakteryzowany jest przez długość słowa B oraz próg obcięcia wartości Z_{th} .

W binarnej reprezentacji, wartości $x \in \mathcal{O}$ odpowiada liczba całkowita x/Δ zapisana w formacie uzupełnienia do 2 (w pamięci dekodera) bądź też w formacie znak-moduł (na potrzeby jednostek obliczeniowych).

Zaokrąglenie wiadomości do wartości ze zbioru $\mathcal{O}_{B,Z_{th}}$ jest przyczyną błędów zaokrąglenia, które są zależne od kroku kwantyzacji Δ oraz błędów obcięcia zależnych od wartości progu Z_{th} . Zmniejszenie tych błędów jest możliwe poprzez zwiększenie długości słowa B . Jak pokazują wyniki symulacji, które zostaną przedstawione w dalszej części rozdziału, dobór wartości progu Z_{th} ma istotny wpływ na jakość

dekodera. Dla danej długości słowa B , odpowiednia wartość Z_{th} jest wyznaczana na drodze symulacji.

Algorytm 4.2 może być wprost zastosowany dla wiadomości ze zbioru \mathcal{O} , przy czym:

- wartości inicjalizujące są zaokrąglone do najbliższych wartości ze zbioru \mathcal{O} , stąd $\delta_n \in \mathcal{O}$,
- wiadomości $Q_n \in \mathcal{O}$, $\lambda_n^{d,p} \in \mathcal{O}$, $\rho_n^p \in \mathcal{O}$,
- funkcja $CN()$ w zależności (4.4) jest określona na zbiorze \mathcal{O} oraz przyjmuje wartości ze zbioru \mathcal{O} ,
- równanie (4.3) przyjmuje postać:

$$\rho_n^p := \text{CLIP}_{Z_{th}}(Q_n - \lambda_n^{d,p}) \quad n \in \mathcal{N}(d, p) \quad (4.8a)$$

$$\text{CLIP}_{Z_{th}}(x) \stackrel{\text{def}}{=} \begin{cases} -Z_{th}, & x \leq -Z_{th} \\ x, & -Z_{th} < x < Z_{th} \\ Z_{th}, & x \geq Z_{th} \end{cases} \quad (4.8b)$$

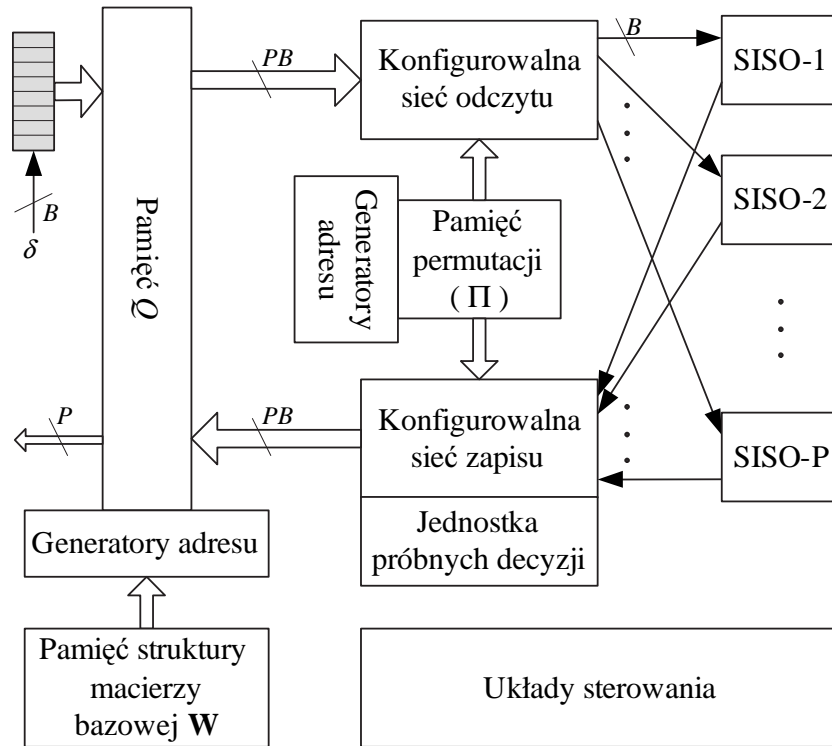
- równanie (4.5b) przyjmuje postać:

$$Q_n := \text{CLIP}_{Z_{th}}(\rho_n^p + \Lambda_n^p) \quad n \in \mathcal{N}(d, p) \quad (4.9)$$

Funkcja $\text{CLIP}_{Z_{th}}(x)$ zdefiniowana w (4.8b) ogranicza wynik sumowania do dopuszczalnego przedziału wartości ze zbioru $\mathcal{O}_{B, Z_{th}}$. Taka operacja jest nazywana obcinaniem wartości (ang. *clipping*) i jest źródłem błędów obcięcia, które – jak wykazały eksperymenty – są dosyć znaczące. Przeprowadzona analiza wpływu błędów obcięcia wartości wiadomości w (4.9) na skuteczność procesu dekodowania będzie przedstawiona w dalszej części pracy, po prezentacji struktury dekodera. Zaproponowano bowiem pewne modyfikacje jednostek obliczeniowych dekodera TDMP, które umożliwiają znaczące zmniejszenie negatywnego wpływu błędów obcięcia.

4.6 Struktura dekodera

Dzięki zastosowaniu algorytmu TDMP oraz jednostek obliczeniowych realizujących swoje funkcje zgodnie ze schematem podwójnej rekurencji, zaprojektowany dekoderek składa się z tylko jednej globalnej pamięci wiadomości oraz jednego typu jednostek obliczeniowych. Jest łatwo konfigurowalny oraz cechuje się dużą uniwersalnością. Głównymi elementami zaprojektowanego dekodera (rys. 4.5) są:



Rys. 4.5: Schemat blokowy dekodera

- Pamięć Q – pamięć wiadomości Q_n , $n = 1, \dots, N$, czyli wartości LLR prawdopodobieństw *pseudo-posteriori* dla poszczególnych bitów. Wartości te przechowywane są w L komórkach pamięci. Każda komórka zawiera P słów B -bitowych wiadomości. Zdecydowano się na połączenie P wiadomości w jedno słowo, co umożliwia pobieranie danych przez (maksymalnie) P jednostek obliczeniowych w jednym cyklu, bez konfliktu dostępu do pamięci. Taka organizacja pamięci jest możliwa do zrealizowania w dekodерze kodu AA-LDPC, zdefiniowanego w podrozdziale 4.2, którego rozmiary podmacierzy wynoszą $P \times P$.
- Jednostki obliczeniowe SISO (*Soft Input Soft Output*), które realizują obliczenia według zależności (4.3)–(4.5) algorytmu 4.2, jak również przechowują w lokalnych modułach pamięci wiadomości λ . W jednostce SISO- p przechowywane są B -bitowe słowa wiadomości $\lambda_n^{d,p}$, $d = 1, \dots, D$, $n \in \mathcal{N}(d,p)$ (słowa te zapisywane i odczytywane są tylko przez jednostkę SISO- p ; mogą być zatem przechowywane lokalnie).

- Konfigurowalna sieć odczytu, która dokonuje podziału (PB) -bitowego słowa z pamięci na B -bitowe wiadomości i dostarcza je do odpowiednich jednostek SISO.
- Konfigurowalna sieć zapisu, która odpowiednio łączy wiadomości uaktualnione w jednostkach SISO w (PB) -bitowe słowa zapisywane do pamięci.
- Jednostka próbnych decyzji sprawdzająca w każdej iteracji, czy twarde decyzje podjęte na podstawie aktualnych wartości Q_n wskazują poprawne słowo kodowe. Twarde decyzje \hat{x}_n są równoważne bitom znaku wiadomości Q_n , tak więc jednostka pobiera najbardziej znaczące bity (MSB) wiadomości i na ich podstawie sprawdza spełnienie kolejnych równań kontrolnych.

Dekoder umożliwia dekodowanie kodu AA-LDPC o macierzy kontrolnej zgodnej z (4.1) i działa zgodnie z algorytmem 4.2. Podmacierz \mathbf{H}^d odpowiadająca superkodowi \mathcal{C}^d jest podmacierzą macierzy (4.1):

$$\mathbf{H}^d = [\mathbf{P}_{d,1} \quad \mathbf{P}_{d,2} \quad \cdots \quad \mathbf{P}_{d,L}] \quad d = 1, \dots, D \quad (4.10)$$

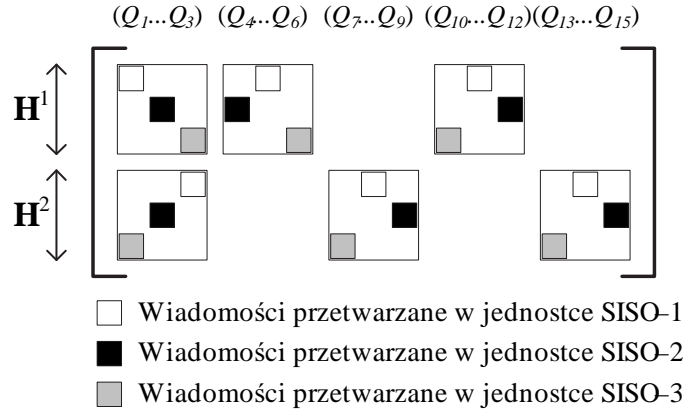
Jedna iteracja dekodowania wykonywana jest w subiteracjach $d = 1, \dots, D$. W subiteracji d , jednostka SISO- p wykonuje obliczenia dla p -tego wiersza podmacierzy \mathbf{H}^d , tzn. uaktualnia wiadomości Q_n , $n \in \mathcal{N}(d, p)$ oraz $\lambda_n^{d,p}$. Poszczególne iteracje pętli zadeklarowanej w wierszu 5 algorytmu 4.2 (**DLA** $p = 1, \dots, P$) wykonywane są zatem równoległe w P jednostkach SISO, natomiast iteracje pętli zadeklarowanej w wierszu 4 algorytmu 4.2 (**DLA** $d = 1, \dots, D$) wykonywane są szeregowo, w D subiteracjach dekodowania.

Przykład 2. Propagacja wiadomości z pamięci Q do jednostek SISO w zaprojektowanym dekodерze

Rozważmy propagację wiadomości dla kodu AA-LDPC o macierzy bazowej $\mathbf{W} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$ i macierzy kontrolnej o rozmiarach 6×15 jak na rys. 4.6, gdzie rozmieszczenie jedynek zobrazowane jest za pomocą małych kwadratów.

Pamięć Q dekodera zawiera w tym przypadku 5 komórek, w każdej komórce przechowywane są po 3 wiadomości $(Q_1, Q_2, Q_3) \dots (Q_{13}, Q_{14}, Q_{15})$. W pierwszej subiteracji wiadomości do jednostek SISO przesyłane są następująco:

	cykl 1	cykl 2	cykl 3
Słowo z pamięci	(Q_1, Q_2, Q_3)	(Q_4, Q_5, Q_6)	(Q_{10}, Q_{11}, Q_{12})
SISO-1	Q_1	Q_5	Q_{11}
SISO-2	Q_2	Q_4	Q_{12}
SISO-3	Q_3	Q_6	Q_{10}



Rys. 4.6: Macierz kontrolna

i po uaktualnieniu zapisywane są w pamięci do tych samych komórek. Podobnie, w drugiej subiteracji przesyłane są następujące wiadomości:

	cykl 1	cykl 2	cykl 3
Słowo z pamięci	(Q_1, Q_2, Q_3)	(Q_7, Q_8, Q_9)	(Q_{13}, Q_{14}, Q_{15})
SISO-1	Q_3	Q_8	Q_{14}
SISO-2	Q_2	Q_9	Q_{15}
SISO-3	Q_1	Q_7	Q_{13}

Odpowiedni podział słowa pamięci na fragmenty przesyłane do poszczególnych jednostek, zgodnie z powyższymi tabelami, jest wykonywany przez konfigurowalną sieć odczytu. Każda z jednostek SISO w subiteracji d jest odpowiedzialna za uaktualnianie wiadomości dla dokładnie jednego wiersza podmacierzy \mathbf{H}^d .

Brak konfliktów dostępu do pamięci możliwy jest dzięki temu, że poszczególne podmacierze \mathbf{H}^d , $d = 1, \dots, D$ nie zawierają dwóch wierszy z jedynkami w tej samej kolumnie. Inaczej mówiąc, zbiory indeksów $\mathcal{N}(d, p)$ dla danego d są rozłączne dla $p = 1, \dots, P$.

Przesłanie odpowiedniego fragmentu słowa pobranego z pamięci do odpowiedniej jednostki SISO jest realizowane w bloku nazwanym konfigurowalną siecią odczytu. Sposób przesyłania poszczególnych fragmentów słowa jest definiowany przez podmacierze permutacji $\mathbf{P}_{d,l}$. Struktura tych podmacierzy przechowywana jest w pamięci permutacji (rys. 4.5), która wraz z generatorem adresu (wybierającym odpowiednią komórkę pamięci w każdym cyklu) steruje konfigurowalną siecią odczytu. W podobny sposób, z wykorzystaniem konfigurowalnej sieci zapisu, dane z wyjść jednostek SISO są łączone w słowa pamięci.

Adresy komórek pamięci Q odczytywanych i zapisywanych w subiteracji d zależą od położenia niezerowych macierzy permutacji w podmacierzy \mathbf{H}^d lub inaczej mówiąc od położenia jedynek w d -tym wierszu macierzy bazowej \mathbf{W} . Generator adresu dla pamięci Q jest zatem sterowany z odpowiednio zapisanej pamięci struktury macierzy bazowej \mathbf{W} .

Wykorzystanie dekodera dla konkretnego kodu AA-LDPC wymaga zatem przede wszystkim: zaimplementowania odpowiedniej liczby jednostek obliczeniowych SISO, zdefiniowania (zapisania) zawartości pamięci struktury macierzy bazowej \mathbf{W} oraz zdefiniowania (zapisania) zawartości pamięci permutacji odpowiednio sformatowanymi słowami, uzależnionymi od poszczególnych podmacierzy permutacji.

4.6.1 Pamięć Q

Pamięć Q zawiera L komórek (PB)-bitowych. Inicjalizacja dekodera polega na wpisaniu do pamięci wartości LLR prawdopodobieństw *a priori* dla poszczególnych bitów $\delta_n = L(x_n|y_n) = \ln\left(\frac{P(x_n=0|y_n)}{P(x_n=1|y_n)}\right)$, $n = 1, \dots, N$. Pojedyncze słowo wpisywane do pamięci zawiera P wartości δ_n . Dane do dekodera mogą być też dostarczane szeregowo, wówczas kolejne P wartości wpisywane są do rejestru szeregowo-równoległego, a następnie z wyjścia rejestru zapisywane do pamięci. Dane wyjściowe dekodera będące twardymi decyzjami dla poszczególnych bitów są równe bitom znaku wiadomości Q_n . Są one odczytywane z dekodera w słowach P -bitowych.

W większości współcześnie dostępnych układów FPGA, w tym w układzie rodziny VirtexII wykorzystywanym w niniejszej pracy, dostępne są pamięci dwuportowe. Taką też pamięć zastosowano w zaprojektowanym dekodерze. W trakcie inicjalizacji, jeden z portów pamięci wykorzystywany jest do zapisywania danych inicjalizujących, a drugi do odczytu wyniku dekodowania poprzedniego bloku. Liczba cykli potrzebnych do inicjalizacji i równoczesnego odczytu poprzedniego bloku wynosi:

$$T_{\text{init}} = L \quad (4.11)$$

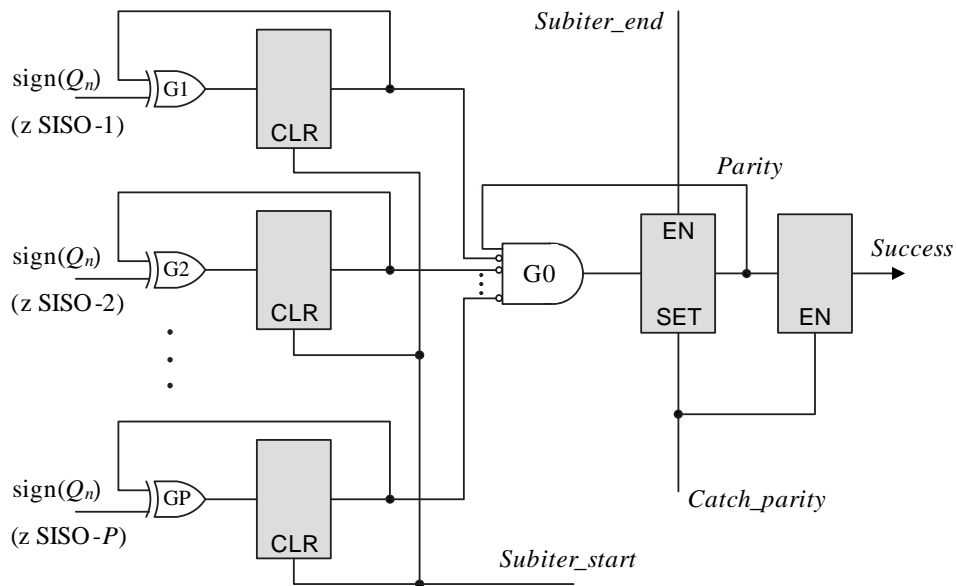
W trakcie procesu dekodowania, jeden z portów wykorzystywany jest do odczytu wartości wiadomości, a drugi do zapisu nowych wartości wiadomości.

4.6.2 Jednostka próbnych decyzji

Zadaniem jednostki próbnych decyzji jest sygnalizacja uzyskania słowa należącego do kodu. W tym celu kontrolowane są na bieżąco wyniki kolejnych równań kontrolnych kodu. Jeśli wszystkie równania kontrolne zostaną spełnione w danej iteracji, jednostka przesyła odpowiedni sygnał do układów sterujących.

Twarde decyzje \hat{x}_n są równoważne bitom znaku wiadomości Q_n , tak więc jednostka pobiera najbardziej znaczące bity (MSB) wiadomości Q_n i na ich podstawie sprawdza spełnienie kolejnych równań kontrolnych. W subiteracji d sprawdzane jest P równań kontrolnych super-kodu \mathcal{C}^d . Równanie kontrolne p super-kodu \mathcal{C}^d jest spełnione, gdy suma modulo-2 bitów znaku wiadomości Q_n dla $n \in \mathcal{N}(d, p)$ jest równa 0. Wiadomości Q_n dla $n \in \mathcal{N}(d, p)$ przesyłane są szeregowo w subiteracji d z jednostki SISO- p . Na wejście jednostki próbnych decyzji dostarczane są zatem bity znaków słów przesyłanych z jednostek SISO.

Schemat jednostki przedstawiono na rysunku 4.7. Jednostka zawiera P układów kontrolnych składających się z bramki Exclusive-Or i przerzutnika. Przerzutniki zerowane są sygnałem *Subiter_start* jeden cykl zegarowy przed pojawieniem się pierwszego słowa w danej subiteracji. Jeden cykl po wczytaniu bitu znaku ostatniego słowa, na wyjściu bramki G0 pojawia się jedynka wtedy i tylko wtedy, gdy wszystkie równania kontrolne super-kodu \mathcal{C}^d są spełnione oraz sygnał *Parity* wskazuje, że równania kontrolne dla super-kodów $1, \dots, d-1$ były również spełnione. Wartość z wyjścia bramki jest wówczas zapisywana do kolejnego przerzutnika i stanowi uaktualnioną wartość sygnału *Parity*. Po zakończeniu ostatniej subiteracji, sygnał *Catch_parity* umożliwia przepisanie tej wartości na wyjście układu. Jeśli wszystkie równania kontrolne są spełnione, sygnał kontrolny *Success* przyjmuje wartość 1.



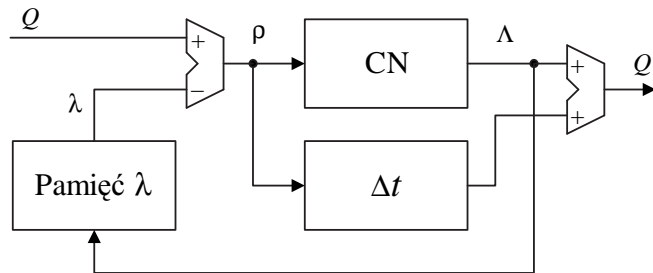
Rys. 4.7: Jednostka próbnych decyzji

4.6.3 Jednostka SISO

Jednostki obliczeniowe SISO-1, ..., SISO- P mają identyczną budowę. Zadaniem jednostki SISO- p jest:

- realizacja obliczeń według zależności (4.3)–(4.5) algorytmu 4.2, dla danego p ,
- przechowywanie B -bitowych wiadomości $\lambda_n^{d,p}$, $d = 1, \dots, D$, $n \in \mathcal{N}(d, p)$.

Jednostka SISO w subiteracji d pobiera szeregowo $d_{c,d}$ wiadomości, wyznacza nowe wartości i zapisuje je szeregowo do pamięci. Na rys. 4.8 przedstawiono schemat operacji realizowanych przez jednostkę SISO. Wiadomości zewnętrzne ρ_n^p równe są różnicy wartości Q_n pobranych z pamięci i wartości $\lambda_n^{d,p}$ wyznaczonych w poprzedniej iteracji (jak w (4.3)). Na ich podstawie w bloku realizującym operacje wierzchołka kontrolnego CN wyznaczone są nowe wartości wiadomości $\lambda_n^{d,p}$, oznaczone przez Λ_n^p (wzór (4.4)). Nowe wartości Q (oznaczone na rysunku 4.8 przez Q') równe są sumie wiadomości λ i ρ , por. (4.5b). Wykonanie operacji CN wymaga pewnej liczby cykli zegarowych, tak więc konieczne jest zastosowanie odpowiedniego układu opóźniającego ciąg wiadomości ρ , oznaczonego na rys. 4.8 przez Δt .



Rys. 4.8: Schemat operacji realizowanych przez jednostkę SISO

Blok CNU (Check Node Unit)

Blok CNU realizuje operacje wierzchołka kontrolnego CN algorytmu LLR-BP. W celu zapewnienia poprawnej pracy dekodera dla kodów nieregularnych, jednostka CNU powinna umożliwiać wykonywanie obliczeń dla wierzchołków kontrolnych o różnych stopniach. Zdecydowano zatem wykorzystać algorytm ze schematem podwójnej rekurencji, w którym wiadomości dostarczane są do bloku CNU pojedynczo. Szeregowo wejście umożliwia zróżnicowanie liczby pobieranych wiadomości w poszczególnych subiteracjach, a co za tym idzie – zróżnicowanie stopni obsługiwanych

wierzchołków. Istotny jest maksymalny stopień wierzchołka, dla którego CNU może wykonać obliczenia. Wartość ta zostanie oznaczona jako $d_{c_{max}}$:

$$d_{c_{max}} = \max(d_{c_1}, d_{c_2}, \dots, d_{c_N}) \quad (4.12)$$

Maksymalny stopień wierzchołka $d_{c_{max}}$ determinuje wymaganą pojemność buforów danych w CNU.

Algorytm 4.3: Schemat podwójnej rekurencji (jednostka SISO- p)	
Dane wejściowe: Wiadomości ρ_n^p , $n \in \mathcal{N}(d, p) = \{1, 2, \dots, d_{c_{max}}\}$	
Dane wyjściowe: Wiadomości Λ_n^p , $n \in \mathcal{N}(d, p) = \{1, 2, \dots, d_{c_{max}}\}$	
1	<i>Rekurencja wprzód</i>
	$L(\alpha_1) := \rho_2^p$ (4.13a)
	$L(\alpha_2) := \rho_1^p$ (4.13b)
2	DLA $j = 3, 4, \dots, d_{c_{max}} - 1$
3	$L(\alpha_j) := L(\alpha_{j-1}) \boxplus \rho_{j-1}^p$ (4.14)
4	$L(\alpha_{d_{c_{max}}}) := L(\alpha_{d_{c_{max}}-1})$ (4.15)
5	<i>Rekurencja w tył</i>
	$L(\beta_{d_{c_{max}}}) := \rho_{d_{c_{max}}-1}^p$ (4.16a)
	$L(\beta_{d_{c_{max}}-1}) := \rho_{d_{c_{max}}}^p$ (4.16b)
6	DLA $j = d_{c_{max}} - 2, d_{c_{max}} - 3, \dots, 2$
7	$L(\beta_j) := L(\beta_{j+1}) \boxplus \rho_{j+1}^p$ (4.17)
8	$L(\beta_1) := L(\beta_2)$ (4.18)
<i>Finalne parowanie</i>	
9	DLA $j = 1, 2, \dots, d_{c_{max}}$
10	$\Lambda_j^p := L(\alpha_j) \boxplus L(\beta_j)$ (4.19)

Wiadomości wyznaczane są zgodnie ze schematem podwójnej rekurencji (2.23),

przy czym wartość wyniku dwuargumentowej operacji \boxplus może być obliczana dokładnie lub też za pomocą jednego z opisanych w rozdziale 2.2 przybliżeń. Ukierunkowany na wykorzystanie w algorytmie TDMP schemat podwójnej rekurencji zapisano formalnie jako algorytm 4.3.

Algorytm 4.3 przedstawia schemat podwójnej rekurencji realizowanej w jednostce SISO- p , dla subiteracji d . Dla uproszczenia notacji założono, że wierzchołki kontrolne super-kodu \mathcal{C}^d mają stopień $d_{c_{max}}$, a zbiór indeksów wierzchołków bitowych sąsiadujących z rozpatrywanym wierzchołkiem kontrolnym (tzn. p -tym wierzchołkiem podzbioru V_c^d) jest równy $\{1, 2, \dots, d_{c_{max}}\}$.

Schemat blokowy jednostki CNU realizującej algorytm 4.3 przedstawiono na rys. 4.9. Znak wiadomości wyznaczany jest w niezależnym bloku, który zostanie przedstawiony w dalszej części podrozdziału. Dwa oddzielne tory zawierające bloki oznaczone CMin-akumulator oraz pamięci typu LIFO (*Last-In First-Out*) umożliwiają wyznaczanie modułów wartości $L(\alpha_j)$ i $L(\beta_j)$. Wartości te służą do obliczania modułów wiadomości wyjściowych w bloku oznaczonym CMin, według zależności (4.19). Nazwa CMin (*Corrected Minimum*) pochodzi od typowego sposobu wyznaczania wartości operacji $|L(\alpha_j) \boxplus L(\beta_j)|$. Jednostki CMin-akumulator służą do rekurencyjnego obliczania kolejnych wartości $|L(\alpha_j)|$, $|L(\beta_j)|$, według (4.14) i (4.17). Bufor LIFO na wejściu gałęzi β odwraca kolejność słów doprowadzanych do akumulatora β , dzięki czemu możliwa jest realizacja rekurencji „w tył”. Bufor LIFO na wyjściu akumulatora α jest konieczny do odpowiedniego dopasowania dwóch strumieni danych (odwrócenia kolejności wartości $|L(\alpha_j)|$). W efekcie dane na wyjściu bloku CNU pojawiają się również w odwróconej kolejności.

Dwa rejestry umieszczone na wejściu (oznaczone szarymi prostokątami) opóźniają wejściowy strumień danych o dwa cykle zegarowe i umożliwiają, wraz z multiplekserami, zamianę kolejności dwóch pierwszych wiadomości doprowadzanych na wejście akumulatorów, w celu inicjalizacji rejestrów zgodnie z (4.13) i (4.16). Moment inicjalizacji akumulatorów jest wskazywany sygnałami sterującymi *Bypass*. Sygnały *Updown* sterują buforami LIFO – wskazują na inkrementację bądź dekrementację wskaźnika danych bufora. Kolejność danych przesyłanych w układzie ilustruje przykład 3.

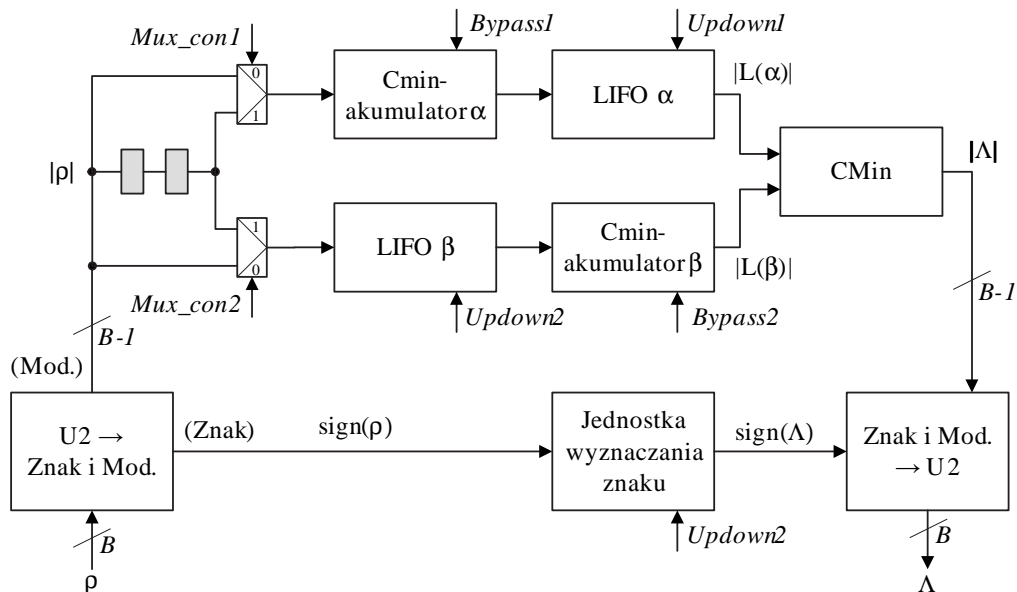
Przykład 3. Propagacja danych w blokach realizujących podwójną rekurencję

Przykład dotyczy obliczeń dla wierzchołka kontrolnego stopnia $d_c = 6$ o zbiorze indeksów węzłów sąsiadujących $\{1, 2, \dots, 6\}$. W poniższej tabeli przedstawiono kolejność danych przesyłanych pomiędzy blokami dwóch gałęzi rekurencji, w kolejnych cyklach zegarowych.

cykl	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
wejście	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	ρ_6										
<i>Mux_con1</i>		0	1	1	1	1	1									
wej CMin α		ρ_2	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5									
wej LIFO α				α_1	α_2	α_3	α_4	α_5	α_6							
wyj $ L(\alpha) $										α_6	α_5	α_4	α_3	α_2	α_1	
<i>Mux_con2</i>		0	0	0	0	0	1									
wej LIFO β		ρ_2	ρ_3	ρ_4	ρ_5	ρ_6	ρ_5									
wej CMin β								ρ_5	ρ_6	ρ_5	ρ_4	ρ_3	ρ_2			
wyj $ L(\beta) $										β_6	β_5	β_4	β_3	β_2	β_1	
wyjście $ \Lambda $											Λ_6	Λ_5	Λ_4	Λ_3	Λ_2	Λ_1

Dla większej przejrzystości zapisu, w powyższej tabeli pominięto oznaczenia wartości bezwzględnych wiadomości oraz funkcji $L(x)$, tzn. zamiast $|L(\alpha_j)|$, w tabeli zamieszczono α_j , a zamiast $|L(\beta_j)|$, zamieszczono β_j .

Dane na wyjściu bloków CMin-akumulator pojawiają się z opóźnieniem dwóch cykli w stosunku do danych wejściowych potrzebnych do ich wyznaczenia. Jest to związane z zastosowaniem w tych blokach dwóch rejestrów potokowych dla zwiększenia maksymalnej częstotliwości pracy. W bloku CMin zastosowano jeden rejestr potokowy, stąd opóźnienie 1 cyklu w tym bloku. Całkowite opóźnienie bloku CNU



Rys. 4.9: Struktura bloku CNU (*Check Node Unit*)

(tzn. liczba cykli zegarowych od momentu pobrania ostatniej wiadomości ρ do momentu pojawienia się na wyjściu pierwszej uaktualnionej wiadomości Λ) wynosi:

$$T_{\text{CNU}} = 5 \quad (4.20)$$

Bloki CMin-akumulator umieszczone w dwóch torach (α i β) są identyczne. Dwuwejściowy blok CMin ma podobną strukturę. Zasadnicza różnica polega na tym, że w bloku CMin-akumulator wyznaczany jest wynik operacji \boxplus dla wartości wejściowej oraz wartości wyznaczonej w poprzednim cyklu (zapisanej w rejestrze – akumulatorze), natomiast w bloku CMin wyznaczany jest wynik operacji \boxplus dla dwóch wartości wejściowych.

Jak wspomniano wynik operacji \boxplus może być wyznaczany za pomocą jednego z opisanych w rozdziale 2.2 przybliżeń, które charakteryzują się różnymi własnościami. W celu dokładnego określenia tych własności (tzn. wymaganych zasobów układu programowalnego, częstotliwości zegarowej oraz skuteczności dekodowania) zdecydowano zaimplementować moduły pracujące zgodnie z różnymi algorytmami wyznaczania wartości operacji \boxplus . Umożliwi to wykonanie eksperymentów pozwalających na porównanie zasobów i częstotliwości zegarowej dekodera oraz bitowej stopy błędów systemu, przy zastosowaniu poszczególnych rozwiązań. W oparciu o te wyniki możliwe jest dobranie jednego z opracowanych rozwiązań do konkretnych wymagań.

Poniżej przedstawione zostaną zaproponowane struktury bloków CMin-akumulator oraz CMin realizujących algorytmy: Min-Sum, Normalized-Min-Sum, Offset-Min-Sum, Corrected-Min-Sum oraz pełny (niezmodyfikowany) LLR-BP.

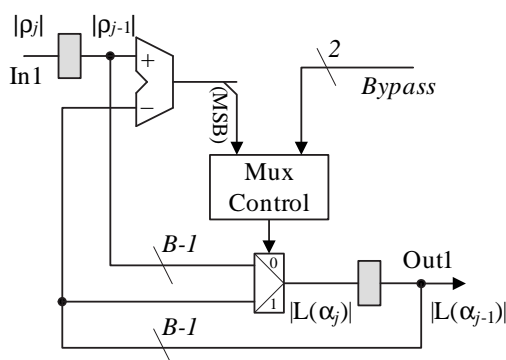
Struktura bloków CMin-akumulator oraz CMin dla algorytmu Min-Sum

W algorytmie Min-Sum zakłada się, że przybliżona wartość modułu wyniku dwuargumentowej operacji \boxplus jest równa minimum modułów argumentów (por. (2.25)), tzn. $|x \boxplus y| \approx \min(|x|, |y|)$. Na rysunku 4.10 przedstawiono blok CMin-akumulator dla algorytmu Min-Sum. Wartość $|L(\alpha_j)|$ (zależność (4.14)) wyznaczana jest według:

$$|L(\alpha_j)| := \min(|L(\alpha_{j-1})|, |\rho_{j-1}|) \quad (4.21)$$

Układ służący do wybierania minimum wartości dwóch słów składa się z subtraktora i multipleksera. Blok Mux-Control dostarcza sygnał sterujący multiplekserem, który jest zależny od sygnału *Bypass* oraz od wartości najstarszego bitu (MSB – *Most Significant Bit*), czyli znaku wyniku odejmowania. Dwubitowy sygnał sterujący *Bypass* określa jeden z trzech możliwych trybów:

- inicjalizacja według (4.13), dla której multiplekser przepuszcza słowo wejściowe,
- rekurencyjne wyznaczanie wartości $|L(\alpha_j)|$, dla której przepuszczane jest jedno ze słów, w zależności od bitu znaku na wyjściu subtraktora,
- w ostatnim cyklu multiplekser przepuszcza słowo z rejestru wyjściowego, zgodnie z (4.15).



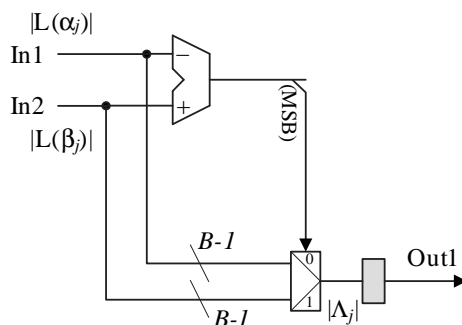
Rys. 4.10: CMin-akumulator dla algorytmu Min-Sum

Struktury bloków CMin-akumulator w gałęziach α oraz β są identyczne.

Schemat jednostki CMin dla algorytmu Min-Sum przedstawiono na rys. 4.11. Obliczenia realizowane są według zależności:

$$|\Lambda_j| := \min(|L(\alpha_j)|, |L(\beta_j)|) \quad (4.22)$$

Wynik zapisywany jest do rejestru potokowego na wyjściu bloku.



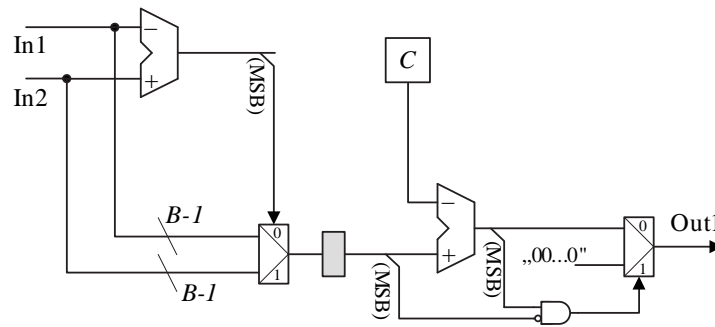
Rys. 4.11: Blok CMin dla algorytmu Min-Sum

Struktura bloków CMin-akumulator oraz CMin dla algorytmu Offset-Min-Sum

W algorytmie Offset-Min-Sum (patrz (2.28)), od wartości wiadomości wyznaczonej algorytmem Min-Sum należy odjąć pewną korygującą stałą dodatnią C . W dekodерze jest to realizowane w bloku CMin, co przedstawiono na rysunku 4.12. Obliczenia przeprowadzane są według zależności:

$$|\Lambda_j| := \max(\min(|L(\alpha_j)|, |L(\beta_j)|) - C, 0) \quad (4.23)$$

Multiplexer umieszczony przy wyjściu układu dostarcza słowo zerowe w sytuacji, gdy w wyniku odejmowania stałej korygującej uzyskana zostaje wartość ujemna.



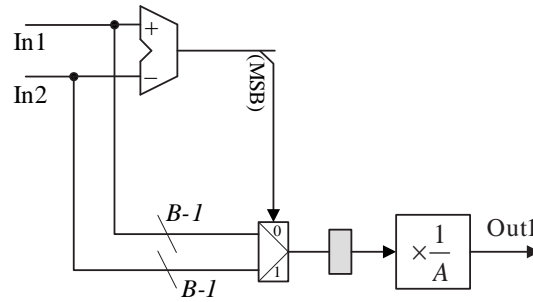
Rys. 4.12: Blok CMin dla algorytmu Offset-Min-Sum

Blok CMin-akumulator jest identyczny jak dla algorytmu Min-Sum (rys. 4.10).

Struktura bloków CMin-akumulator oraz CMin dla algorytmu Normalized-Min-Sum

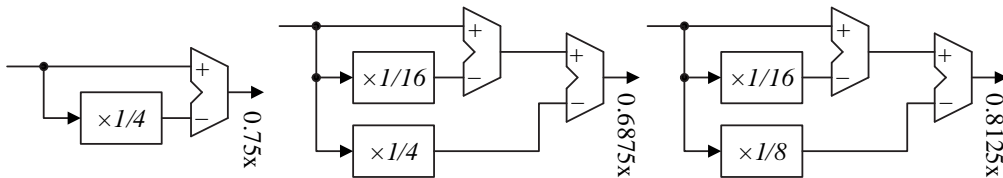
W algorytmie Normalized-Min-Sum, wiadomość wyznaczona algorytmem Min-Sum jest dzielona przez pewną stałą $A > 1$ (por. (2.27)). Normalizacja ta jest realizowana w bloku CMin (rys. 4.13). Blok CMin-akumulator jest identyczny jak dla algorytmu Min-Sum (rys. 4.10). W literaturze opisywane są pewne sposoby optymalizacji wartości A , jednakże (jak wykazały przeprowadzone eksperymenty) zbliżone wyniki dekodowania uzyskuje się dla dość szerokiej gamy wartości parametru A .

Zaproponowano więc następujące wartości parametru: $\frac{1}{A} = 0.6875$, $\frac{1}{A} = 0.75$ oraz $\frac{1}{A} = 0.8125$, dla których układ mnożenia można w stosunkowo prosty sposób wykonać wykorzystując 1 lub 2 sumatory (subtraktory). Bloki mnożenia przez wymienione wartości przedstawiono na rysunku 4.14. Mnożenie przez ujemne potęgi liczby 2 jest realizowane poprzez usunięcie odpowiedniej liczby najmniej znaczących bitów słowa i uzupełnienie zerami na najbardziej znaczących pozycjach – nie wymaga



Rys. 4.13: Blok CMin dla algorytmu Normalized-Min-Sum

zatem żadnych zasobów sprzętowych. Na rysunku 4.15 przedstawiono wyniki symulacji dla kodu regularnego (1008,504) przy zastosowaniu dekodera o parametrach $B = 6$, $Z_{th} = 14$ działającego zgodnie z algorytmem Normalized-Min-Sum z różnymi wartościami stałej normalizującej. Najlepsze rezultaty uzyskano dla $\frac{1}{A} = 0.6875$ oraz $\frac{1}{A} = 0.75$, nieco gorsze $\frac{1}{A} = 0.625$. Podobne wyniki uzyskano dla wielu innych kodów, z czego wyciągnięto wniosek, że najlepszym wyborem w większości przypadków będzie $\frac{1}{A} = 0.75$, gdyż układ mnożący zawiera wówczas tylko jeden subtraktor, zapewniając jednocześnie dobrą skuteczność dekodowania.



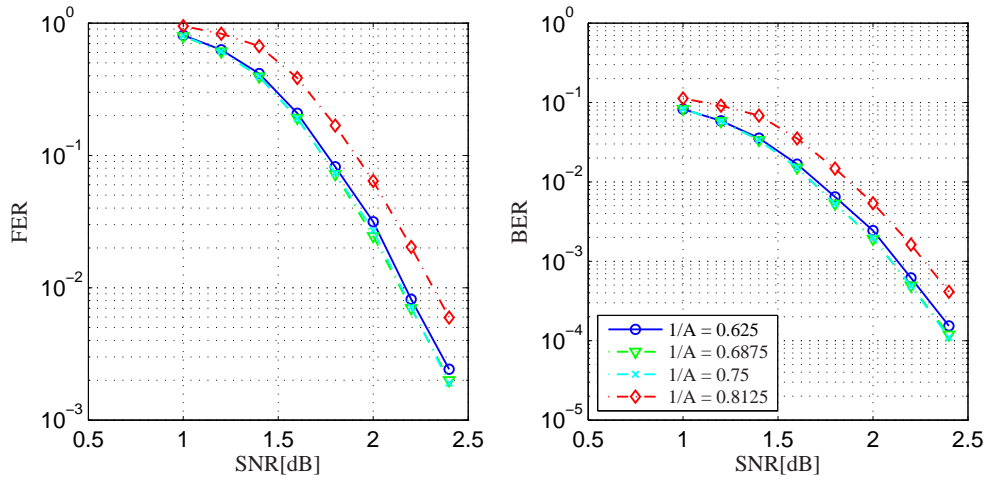
Rys. 4.14: Bloki mnożenia przez 0.75, 0.6875 i 0.8125

Struktura bloków CMin-akumulator oraz CMin dla algorytmu Corrected-Min-Sum

Następująca aproksymacja wyniku operacji \boxplus (por. (2.23), (2.31)) jest podstawą algorytmu Corrected-Min-Sum:

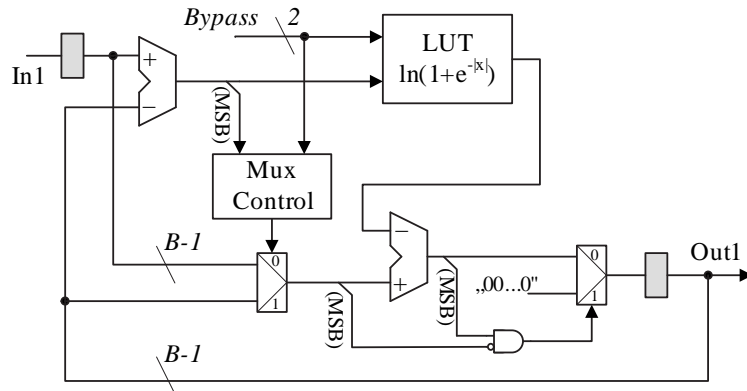
$$|x \boxplus y| \approx \min(|x|, |y|) - \ln(1 + e^{-\|x|-|y\|}) \quad (4.24)$$

Jednostka CMin-akumulator (rys. 4.16) jest w tym przypadku rozbudowana o blok wyznaczający wartość nieliniowej funkcji $\ln(1 + e^{-|x|})$. Wartości tej funkcji są wy-



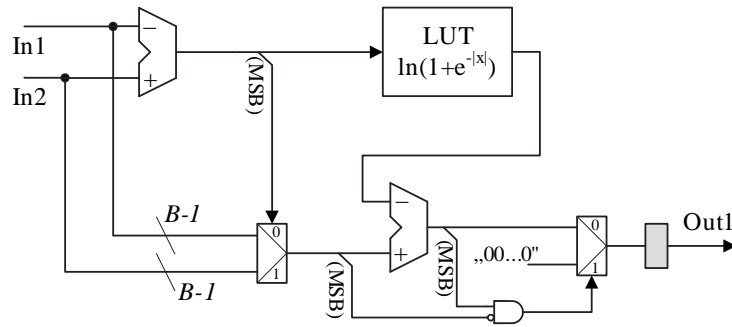
Rys. 4.15: Wyniki symulacji – kod regularny $N = 1008$, $R = 0.5$, algorytm Normalized-Min-Sum, $B = 6$

znaczane w układzie kombinacyjnym (LUT – *Look-Up Table*) opisanym za pomocą tablicy, przy danych parametrach słowa wiadomości B , Z_{th} . Podobnie jak w układzie dla algorytmu Min-Sum, w przypadku gdy wartość funkcji korekcji jest większa niż wartość korygowanego słowa, wyjściowe słowo powinno być zerowe. Zapewnia to multiplexer umieszczony przy wyjściu układu wraz ze sterującą nim bramką.



Rys. 4.16: CMin-akumulator dla algorytmu Corrected-Min-Sum

Schemat jednostki CMin zamieszczono na rysunku 4.17. W podobny sposób jest on rozbudowany o moduł korekcji wartością funkcji $\ln(1 + e^{-|x|})$.



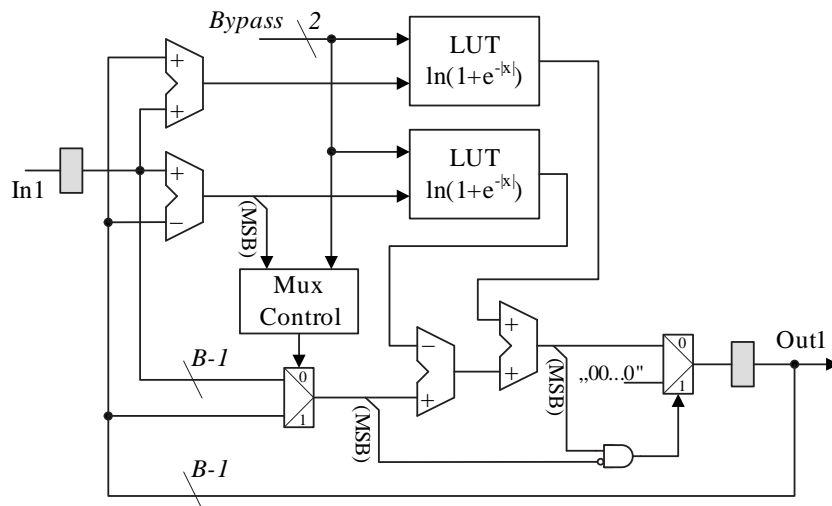
Rys. 4.17: Blok CMin dla algorytmu Corrected-Min-Sum

Struktura bloków CMin-akumulator oraz CMin dla algorytmu LLR-BP

Dokładna wartość wyniku operacji \boxplus (por. (2.23), (2.30)) wynosi:

$$|x \boxplus y| = \min(|x|, |y|) - \ln(1 + e^{-||x|-|y||}) + \ln(1 + e^{-||x|+|y||}) \quad (4.25)$$

Przy wykorzystaniu powyższej zależności, dekodery działają zgodnie z algorytmem LLR-BP. Dokładność obliczeń jest w tym przypadku największa – w wyniku występują tylko błędy zaokrąglenia i obcięta wartości wiadomości. Duża precyzja obliczeń jest okupiona większą złożonością jednostek CMin-akumulator (rys. 4.18) i CMin.



Rys. 4.18: CMin-akumulator dla algorytmu LLR-BP

Jednostka wyznaczania znaków wiadomości

Ostatnim elementem bloku CNU wymagającym komentarza jest jednostka wyznaczania znaków wiadomości. Zgodnie z (2.19) znak wiadomości Λ_n równy jest iloczynowi znaków wszystkich wiadomości zewnętrznych oprócz ρ_n , tzn.:

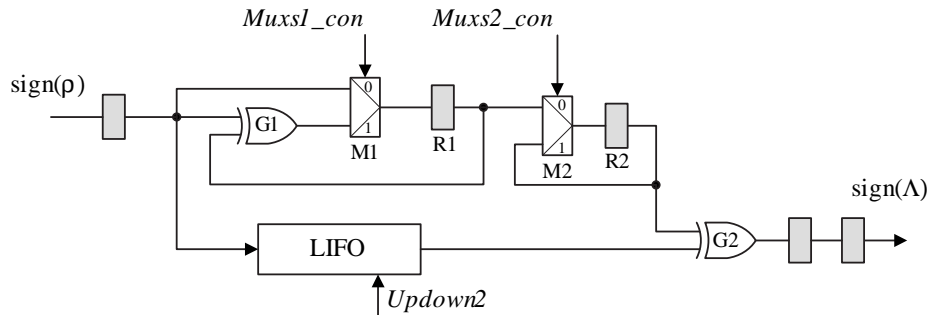
$$\text{sgn}(\Lambda_n^p) = \prod_{n' \in \mathcal{N}(d,p) \setminus n} \text{sgn}(\rho_{n'}^p) \quad (4.26)$$

Iloczynowi znaków odpowiada funkcja Exclusive-Or bitów znaku wiadomości. Wzór (4.26) można wyrazić również następująco:

$$\text{sgn}(\Lambda_n^p) = \text{sgn}(\rho_n^p) \prod_{n' \in \mathcal{N}(d,p)} \text{sgn}(\rho_{n'}^p) \quad (4.27)$$

co pozwala zaproponować stosunkowo prostą strukturę modułu sprzętowego.

Jednostka wyznaczania znaków w subiteracji d pobiera $|\mathcal{N}(d,p)|$ znaków i wyznacza tyleż samo znaków wiadomości wyjściowych. W pierwszej kolejności wyznaczany jest iloczyn wszystkich znaków doprowadzanych szeregowo na wejście $\prod_{n' \in \mathcal{N}(d,p)} \text{sgn}(\rho_{n'}^p)$, a następnie wartość tego iloczynu jest mnożona przez kolejne znaki, które są zapamiętane w buforze.



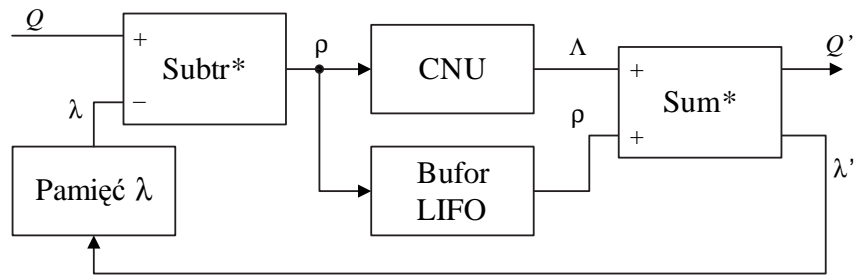
Rys. 4.19: Jednostka wyznaczania znaku wiadomości

Schemat zrealizowanej jednostki przedstawiono na rysunku 4.19. Bramka G1 wraz rejestrem R1 służą do wyznaczenia iloczynu znaków wszystkich wiadomości (czynnikiem $\prod \text{sgn}(\rho_{n'}^p)$ w zależności (4.27)). Wyznaczenie tego iloczynu wymaga $|\mathcal{N}(d,p)|$ cykli zegarowych, po czym jest on przepisywany do rejestru R2 (odpowiedni moment jest wskazywany sygnałem $Muxs2_con$). Bramka G2 wykonuje mnożenie przez znaki kolejnych wiadomości (według (4.27)), które są zapamiętywane w buforze LIFO. Dodatkowo LIFO odwraca kolejność danych wejściowych, w celu odpowiedniego dopasowania strumienia danych wyjściowych do strumienia danych

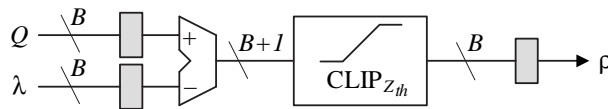
z toru wyznaczania modułu wiadomości. Konieczne są również dwa rejestry umieszczone na wyjściu odpowiednio opóźniające strumień danych wyjściowych.

Struktura jednostki SISO

Struktura jednostki SISO zgodna ze schematem realizowanych operacji (rys. 4.8) przedstawiona jest na rys. 4.20. Blok opóźniający wiadomości ρ jest buforem LIFO, który oprócz opóźnienia realizuje też odwrócenie kolejności wiadomości, w celu dopasowania ich kolejności do strumienia danych na wyjściu bloku CNU. Bloki oznaczone Subtr* oraz Sum* to w podstawowej wersji układy (odpowiednio) subtraktora oraz sumatora, wyposażone w rejestry potokowe (rys. 4.21 i 4.22). Zaproponowano jednak pewne modyfikacje w budowie tych bloków umożliwiające zmniejszenie wpływu błędów obcinania wiadomości na skuteczność dekodowania (stąd „*” w oznaczeniu bloków). Ze względu na sporą wagę tego problemu, analizie błędów obcinania wiadomości oraz prezentacji propozycji modyfikacji bloków obliczeniowych zostanie poświęcony oddzielny podrozdział.



Rys. 4.20: Struktura jednostki SISO

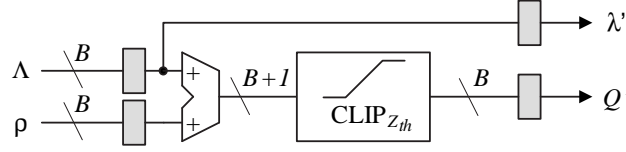


Rys. 4.21: Blok Subtr*

Ze względu na zastosowanie dwóch rejestrów potokowych, opóźnienia bloków Subtr* oraz Sum* wynoszą odpowiednio:

$$T_{\text{Subtr}^*} = 2 \quad (4.28a)$$

$$T_{\text{Sum}^*} = 2 \quad (4.28b)$$



Rys. 4.22: Blok Sum*

Sumaryczne opóźnienie jednostki SISO (tzn. liczba cykli zegarowych od momentu pobrania ostatniej wiadomości Q_n w danej subiteracji do momentu pojawienia się na wyjściu pierwszej uaktualnionej wiadomości) wynosi:

$$T_{\text{SISO}} = T_{\text{Subtr}^*} + T_{\text{CNU}} + T_{\text{Sum}^*} = 9 \quad (4.29)$$

4.6.4 Analiza błędów obcięcia wiadomości

W niniejszym podrozdziale przedstawiona jest analiza negatywnego wpływu obcięcia wartości wiadomości (zależność (4.9)) na skuteczność dekodowania w dekodrze TDMP. Zaprezentowany będzie sposób zredukowania tego wpływu poprzez pewne modyfikacje układu dekodera, jak również wyniki symulacji potwierdzające celowość tych modyfikacji.

Wartość wiadomości Q_n uaktualnionej w jednostce SISO- p , zgodnie z (4.9) wynosi:

$$Q_n := \text{CLIP}_{Z_{th}}(\rho_n^p + \Lambda_n^p) = \rho_n^p + \Lambda_n^p + \epsilon_{(Q_n)} \quad (4.30)$$

gdzie $\epsilon_{(Q_n)}$ to błąd obcięcia wartości wiadomości Q_n wynoszący:

$$\epsilon_{(Q_n)} = \begin{cases} Z_{th} - (\rho_n^p + \Lambda_n^p), & (\rho_n^p + \Lambda_n^p) > Z_{th} \\ 0, & -Z_{th} \leq (\rho_n^p + \Lambda_n^p) \leq Z_{th} \\ -Z_{th} - (\rho_n^p + \Lambda_n^p), & (\rho_n^p + \Lambda_n^p) < -Z_{th} \end{cases} \quad (4.31)$$

Przykład 4. Zaburzenie wartości ρ_n^p przez błąd obcięcia $\epsilon_{(Q_n)}$ z poprzedniej iteracji

Rozpatrywana jest jednostka SISO- p , w subiteracji $d = 1$ dekodera o dowolnej długości słowa B oraz $Z_{th} = 10$.

Niech $n_1 \in \mathcal{N}(1, p)$.

Iteracja 1, subiteracja $d = 1$. Niech wartość wiadomości $Q_{n_1} = \delta_{n_1} = 10$ (czyli prawdopodobieństwa *a priori* wskazują na wartość bitu $x_{n_1} = 1$ z maksymalną możliwą wiarygodnością).

Rozpatrywana jest iteracja 1, tak więc:

$$\lambda_{n_1}^{1,p} = 0$$

$$\rho_{n_1}^p = Q_{n_1} = 10$$

Niech $\Lambda_{n_1}^p := \text{CN}(\{\rho_n^p : n \in \mathcal{N}(1,p) \setminus n_1\}) = 10$ (czyli prawdopodobieństwa wyznaczone na podstawie równania kontrolnego wskazują również wartość bitu $x_{n_1} = 1$ z maksymalną możliwą wiarygodnością).

Przy bezpośrednim zastosowaniu algorytmu 4.2 nowe wartości wiadomości wynoszą:

$$Q_{n_1} := \text{CLIP}_{Z_{th}}(\rho_{n_1}^p + \Lambda_{n_1}^p) = \rho_{n_1}^p + \Lambda_{n_1}^p + \epsilon_{(Q_{n_1})} = 10$$

$$\lambda_{n_1}^{1,p} := \Lambda_{n_1}^p = 10$$

Błąd obciążenia wynosi $\epsilon_{(Q_{n_1})} = -10$.

Iteracja 2, subiteracja $d = 1$. Niech wartość wiadomości $Q_{n_1} = 10$ (czyli prawdopodobieństwa *pseudo-posteriori*, po zakończeniu iteracji 1, wskazują nadal wartość bitu $x_{n_1} = 1$ z maksymalną możliwą wiarygodnością).

Wartość wiadomości zewnętrznej $\rho_{n_1}^p$ wynosi:

$$\rho_{n_1}^p := Q_{n_1} - \lambda_{n_1}^{1,p} = 10 - 10 = 0$$

czyli wskazuje ona na niepewność co do wartości bitu x_{n_1} (gdy tymczasem zarówno wartości *a priori*, jak również – w domniemaniu – pozostałe równania kontrolne, w których partycypuje bit x_{n_1} wskazywały na prawie całkowitą pewność co do wartości bitu x_{n_1}). Jest to związane z faktem, że wartość $\rho_{n_1}^p$ jest obciążona błędem obciążenia $\epsilon_{(Q_{n_1})}$ z poprzedniej iteracji.

Obciążenie $\rho_{n_1}^p$ błędem obciążenia powoduje, że nowe wartości wiadomości $\lambda_n^{1,p}$ dla $n \in \mathcal{N}(1,p)$ są również obciążone owym błędem (dokładniej: ponieważ $\rho_{n_1}^p = 0$, to moduły tych wiadomości wynoszą również 0). Okazuje się, że ma to silnie negatywny wpływ na skuteczność dekodowania.

W celu przeprowadzenia analizy błędów obciążenia, oznaczenia wiadomości zostaną uzupełnione o numery iteracji i subiteracji: $Q_n^{(i,d)}$ to wartość wiadomości Q_n po zakończeniu subiteracji d w iteracji i , podobnie $\lambda_n^{d,p,(i)}$, $\Lambda_n^{p,(i,d)}$ oraz $\rho_n^{p,(i,d)}$. Indeksy umieszczone w nawiasach oznaczają zatem „czas”, a indeksy bez nawiasów oznaczają „miejsce”, tzn. przykładowo $\lambda_n^{d,p,(i)}$ to wartość umieszczona w komórce pamięci λ jednostki SISO- p o adresie zależnym od n oraz d , zapisana w iteracji i .

Wartość wiadomości $Q_n^{(i,d)}$ wynosi:

$$Q_n^{(i,d)} = \text{CLIP}_{Z_{th}}(\rho_n^{p,(i,d)} + \Lambda_n^{p,(i,d)}) = \rho_n^{p,(i,d)} + \Lambda_n^{p,(i,d)} + \epsilon_{(Q_n^{(i,d)})} \quad (4.32)$$

gdzie $\epsilon_{(Q_n^{(i,d)})}$ to błąd obciążenia wartości wiadomości $Q_n^{(i,d)}$.

Ponieważ zgodnie z (4.3) i (4.5a):

$$\Lambda_n^{p,(i,d)} = \lambda_n^{d,p,(i)} \quad (4.33a)$$

$$\rho_n^{p,(i,d)} = \begin{cases} Q_n^{(i-1,D)} - \lambda_n^{d,p,(i-1)} & d = 1 \\ Q_n^{(i,d-1)} - \lambda_n^{d,p,(i-1)} & d > 1 \end{cases} \quad (4.33b)$$

to dla $d > 1$ można $Q_n^{(i,d)}$ przedstawić jako:

$$Q_n^{(i,d)} = Q_n^{(i,d-1)} - \lambda_n^{d,p,(i-1)} + \lambda_n^{d,p,(i)} + \epsilon_{(Q_n^{(i,d)})} \quad (4.34)$$

przy czym w (4.33b) pominięto błędy obciążenia wartości $\rho_n^{p,(i,d)}$, gdyż nie są one istotne z punktu widzenia niniejszej analizy. Poza tym są niezerowe tylko wówczas, gdy moduły wartości $\lambda_n^{d,p,(i-1)}$ i $Q_n^{(i-1,D)}$ są duże, a znaki przeciwne. Jest to sytuacja bardzo rzadka, ponieważ $\lambda_n^{d,p,(i-1)}$ jest jednym ze składników wartości $Q_n^{(i-1,D)}$.

Niech $\mathcal{DP}(n)$ oznacza zbiór takich par (d, p) , dla których wierzchołek bitowy b_n jest połączony krawędzią z p -tym wierzchołkiem kontrolnym super-kodu \mathcal{C}^d , czyli:

$$\mathcal{DP}(n) = \{d, p : n \in \mathcal{N}(d, p)\} \quad (4.35)$$

Na podstawie (4.34) wyprowadzono wzór na wartość wiadomości Q_n po zakończeniu iteracji i (czyli $Q_n^{(i,D)}$) w zależności od wartości tej wiadomości po zakończeniu poprzedniej iteracji ($Q_n^{(i-1,D)}$), z uwzględnieniem wszystkich błędów obciążenia wartości Q_n powstałych w iteracji i . Wzór ten jest następujący:

$$Q_n^{(i,D)} = Q_n^{(i-1,D)} + \sum_{(d,p) \in \mathcal{DP}(n)} \left(-\lambda_n^{d,p,(i-1)} + \lambda_n^{d,p,(i)} \right) + \sum_{(d,p) \in \mathcal{DP}(n)} \epsilon_{(Q_n^{(i,d)})} \quad (4.36)$$

Wartość $Q_n^{(i,D)}$ jest zatem sumą „prawdziwej” (czyli nie obciążonej błędem obciążenia w iteracji i) wartości $[Q_n^{(i,D)}]_{TRUE}$:

$$[Q_n^{(i,D)}]_{TRUE} = Q_n^{(i-1,D)} + \sum_{(d,p) \in \mathcal{DP}(n)} \left(-\lambda_n^{d,p,(i-1)} + \lambda_n^{d,p,(i)} \right) \quad (4.37)$$

oraz sumarycznego błędu obciążenia w iteracji i -tej, który będzie oznaczany $\epsilon_n^{(i)}$:

$$\epsilon_n^{(i)} = \sum_{(d,p) \in \mathcal{DP}(n)} \epsilon_{(Q_n^{(i,d)})} \quad (4.38)$$

Liczność zbioru par (d, p) , dla których wykonywane jest sumowanie jest oczywiście równa stopniowi wierzchołka bitowego b_n , czyli $|\mathcal{DP}(n)| = d_{b_n}$.

Niech pierwszy element zbioru $\mathcal{DP}(n)$ to (d_1, p_1) , tzn. dla $d < d_1$ nie istnieje element $(d, p) \in \mathcal{DP}(n)$. W subiteracjach $1, \dots, d_1 - 1$ wartość Q_n nie jest modyfikowana, tak więc wartość wiadomości zewnętrznej w subiteracji d_1 wynosi:

$$\rho_n^{p_1,(i+1,d_1)} = Q_n^{(i,D)} - \lambda_n^{d_1,p_1,(i)} = [Q_n^{(i,D)}]_{TRUE} - \lambda_n^{d_1,p_1,(i)} + \epsilon_n^{(i)} \quad (4.39)$$

W podobny sposób można przedstawić zależność określającą wiadomości $\rho_n^{p,(i+1,d)}$ dla kolejnych wartości $(d, p) \in \mathcal{DP}(n)$.

Okazuje się, że można zmniejszyć obciążenie wiadomości $\rho_n^{p,(i+1,d)}$ błędem $\epsilon_n^{(i)}$ w (4.39). W wyniku przeprowadzonych rozważań oraz szeregu eksperymentów, opracowano dwie metody, które będą nazywane podstawową modyfikacją algorytmu dekodowania oraz warunkową normalizacją wiadomości. Pierwsza dotyczy bloku Sum*, a druga bloku Subtr*. Łączne zastosowanie tych modyfikacji umożliwia znaczne zmniejszenie obciążenia wiadomości $\rho_n^{p,(i+1,d)}$ błędem obciążenia, a co za tym idzie, poprawę jakości dekodowania.

Podstawowa modyfikacja algorytmu dekodowania

Jeśli wartość wiadomości λ w (4.5a), zamiast prostego podstawienia $\lambda_n^{d,p} := \Lambda_n^p$, jest wyznaczana w sposób następujący:

$$[\lambda_n^{d,p}]_{ALT} := \text{CLIP}_{Z_{th}}(\rho_n^p + \Lambda_n^p) - \rho_n^p \quad (4.40)$$

to składnikiem $[\lambda_n^{d,p}]_{ALT}$ jest błąd obciążenia $\epsilon_{(Q_n)}$, tzn.:

$$\begin{aligned} [\lambda_n^{d,p,(i)}]_{ALT} &= \rho_n^{p,(i,d)} + \Lambda_n^{p,(i,d)} + \epsilon_{(Q_n^{(i,d)})} - \rho_n^{p,(i,d)} \\ &= \Lambda_n^{p,(i,d)} + \epsilon_{(Q_n^{(i,d)})} = \lambda_n^{d,p,(i)} + \epsilon_{(Q_n^{(i,d)})} \end{aligned} \quad (4.41)$$

Wówczas, wiadomość zewnętrzna ρ w subiteracji d_1 ma wartość:

$$\begin{aligned} \rho_n^{p_1,(i+1,d_1)} &= Q_n^{(i,D)} - [\lambda_n^{d_1,p_1,(i)}]_{ALT} \\ &= [Q_n^{(i,D)}]_{TRUE} - \lambda_n^{d_1,p_1,(i)} + [\epsilon_n^{(i)} - \epsilon_{(Q_n^{(i,d_1)})}] \end{aligned} \quad (4.42)$$

Porównując (4.39) z (4.42) można zauważyć, że w (4.42) z sumarycznego błędu obciążenia obciążającego wartość $\rho_n^{p_1,(i+1,d_1)}$ usunięty został składnik $\epsilon_{(Q_n^{(i,d_1)})}$.

Przykład 5. Zaburzenie wartości ρ_n^p – algorytm zmodyfikowany

Dane z przykładu 4, algorytm z zaproponowaną modyfikacją.

Iteracja 1, subiteracja $d = 1$.

Niech $Q_{n_1} = \delta_{n_1} = 10$

$\lambda_{n_1}^{1,p} = 0$

$\rho_{n_1}^p = Q_{n_1} = 10$

Niech $\Lambda_{n_1}^p := \text{CN}(\{\rho_n^p : n \in \mathcal{N}(1,p) \setminus n_1\}) = 10$

Jeśli jako wiadomość λ podstawiona zostanie zmodyfikowana wartość, zdefiniowana w (4.41), to nowe wartości wiadomości wynoszą:

$Q_{n_1} := \text{CLIP}_{Z_{th}}(\rho_{n_1}^p + \Lambda_{n_1}^p) = \rho_{n_1}^p + \Lambda_{n_1}^p + \epsilon_{(Q_{n_1})} = 10$

$\lambda_{n_1}^{1,p} := \text{CLIP}_{Z_{th}}(\rho_{n_1}^p + \Lambda_{n_1}^p) - \rho_{n_1}^p = 10 - 10 = 0$

Błąd obciążenia wynosi $\epsilon_{(Q_{n_1})} = -10$.

Iteracja 2, subiteracja $d = 1$.

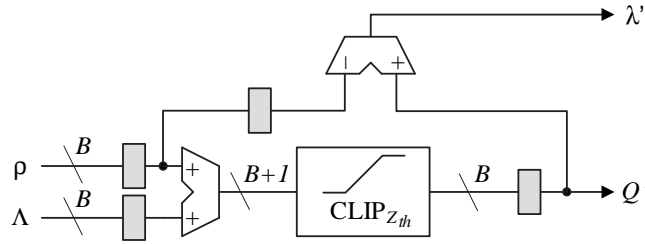
Niech $Q_{n_1} = 10$

Wartość wiadomości zewnętrznej $\rho_{n_1}^p$ wynosi:

$\rho_{n_1}^p := Q_{n_1} - \lambda_{n_1}^{1,p} = 10 - 0 = 10$

czyli wskazuje ona wartość bitu $x_{n_1} = 1$ z maksymalną możliwą wiarygodnością – nie jest obciążona błędem obciążenia z subiteracji $d = 1$ poprzedniej iteracji.

Schemat bloku Sum*, w którym zastosowano opisaną modyfikację, przedstawiony jest na rysunku 4.23.



Rys. 4.23: Blok Sum* – wersja z podstawową modyfikacją

W żadnym ze znanych autorowi źródeł literaturowych opisujących algorytm TDMP nie wspomina się o opisanym problemie. Jednak, jak wykazały liczne eksperymenty, zastosowanie opisaney podstawowej modyfikacji jest kluczowe dla poprawnego działania dekodera przy ograniczonej liczbie bitów B . W przeciwnym razie jakość dekodera (bitowa stopa błędów) jest daleko niezadowalająca. Wszystkie wyniki eksperymentów pochodzące ze środowiska wykorzystującego dekodier implementowany w programowalnym układzie logicznym zrealizowane są z wykorzystaniem dekodera z przedstawioną modyfikacją.

Warunkowa normalizacja wiadomości λ

Bezpośrednie usunięcie pozostałych składników $\epsilon_n^{(i)}$ z wartości ρ_n wymagałoby zastosowania globalnej pamięci $\epsilon_n^{(i)}$ oraz sieci konfigurowalnych połączeń do jednostek obliczeniowych – a zatem ogromnego zwiększenia złożoności układu dekodera.

Zaproponowana metoda opiera się na spostrzeżeniu, że składniki błędu $\epsilon_n^{(i)}$ są zmiennymi losowymi, których rozkład jest zależny od aktualnej wartości wiadomości Q_n .

Rozpatrywany jest wierzchołek stopnia 3, tzn. $\mathcal{DP}(n) = \{(d_1, p_1), (d_2, p_2), (d_3, p_3)\}$, stąd $\epsilon_n^{(i)} = \epsilon_{(Q_n^{(i,d_1)})} + \epsilon_{(Q_n^{(i,d_2)})} + \epsilon_{(Q_n^{(i,d_3)})}$. Błąd $\epsilon_{(Q_n^{(i,d_1)})}$ usuwany jest z wartości ρ metodą opisaną powyżej, pozostają zatem składniki $\epsilon_{(Q_n^{(i,d_2)})} + \epsilon_{(Q_n^{(i,d_3)})}$. Można zauważyć, że:

- Do jednostki SISO- p_1 w subiteracji d_1 jest dostarczana wartość $Q_n^{(i,d_3)}$ (jako wiadomość wejściowa w (4.3)).
- Błąd $\epsilon_{(Q_n^{(i,d_3)})}$ ma wartość nieujemną wtedy i tylko wtedy, gdy $Q_n^{(i,d_3)} = -Z_{th}$ oraz ma wartość niedodatnią wtedy i tylko wtedy, gdy $Q_n^{(i,d_3)} = Z_{th}$. W pozostałych wypadkach $\epsilon_{(Q_n^{(i,d_3)})} = 0$.

- Błąd $\epsilon_{(Q_n^{(i,d_2)})}$ może być traktowany jako zmienna losowa o pewnym rozkładzie, który jest zależny od $Q_n^{(i,d_3)}$: dla wartości $Q_n^{(i,d_3)} > 0$ prawdopodobieństwo, że $\epsilon_{(Q_n^{(i,d_2)})}$ ma wartość ujemną jest większe niż prawdopodobieństwo, że ma wartość dodatnią (odwrotnie dla $Q_n^{(i,d_3)} < 0$). Im większy moduł wartości $Q_n^{(i,d_3)}$, tym większe prawdopodobieństwo, że $\epsilon_{(Q_n^{(i,d_2)})}$ jest niezerowe.

W podobny sposób można rozszerzyć powyższe rozumowanie dla przypadku wierzchołka o stopniu większym niż 3.

Na podstawie powyższych spostrzeżeń można zauważyć, iż dla $Q_n^{(i,d_3)} = Z_{th}$ (lub w ogólności $Q_n^{(i,d_3)} \gg 0$) istnieje duże prawdopodobieństwo, że $\epsilon_{(Q_n^{(i,d_2)})} + \epsilon_{(Q_n^{(i,d_3)})}$ ma wartość ujemną. Podobnie dla $Q_n^{(i,d_3)} = -Z_{th}$ (lub w ogólności $Q_n^{(i,d_3)} \ll 0$) istnieje duże prawdopodobieństwo, że $\epsilon_{(Q_n^{(i,d_2)})} + \epsilon_{(Q_n^{(i,d_3)})}$ ma wartość dodatnią.

Biorąc to pod uwagę, zaproponowano warunkową normalizację wiadomości λ , która polega na modyfikacji wartości ρ_n^p w (4.3) wtedy, gdy moduł wejściowej wartości wiadomości Q_n (czyli $Q_n^{(i,d_3)}$ w powyższych rozważaniach) jest większy od pewnego progu Q_{th} . Dla warunkowej normalizacji wiadomości λ , w (4.3) należy zastosować następujące podstawienie:

$$\rho_n^p := \begin{cases} Q_n - \lambda_n^{d,p}, & |Q_n| \leq Q_{th} \\ Q_n - \frac{\lambda_n^{d,p}}{A_\lambda}, & |Q_n| > Q_{th} \end{cases} \quad (4.43)$$

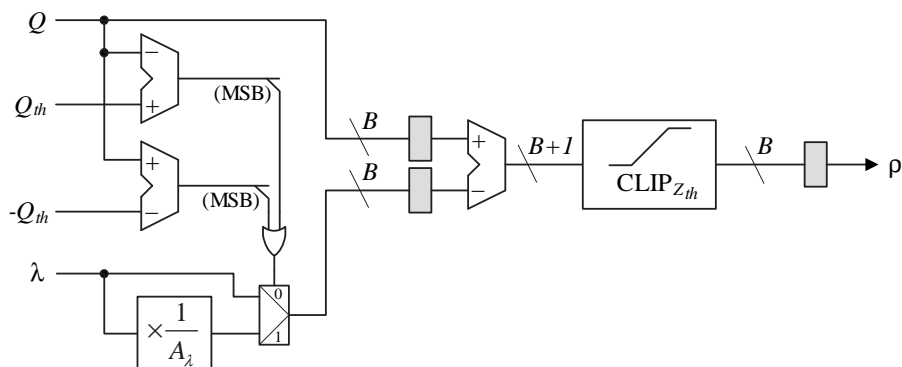
gdzie Q_{th} to próg normalizacji, $1/A_\lambda$ to stała normalizacji.

Jeśli wartość wiadomości $|Q_n|$ jest duża, tzn. większa od progu Q_{th} , to normalizacja wartości $\lambda_n^{d,p}$ w (4.43) modyfikuje ρ_n^p o pewną wartość, której znak jest przeciwny do znaku (przewidywanej) wartości błędu $\epsilon_{(Q_n^{(i,d_2)})} + \epsilon_{(Q_n^{(i,d_3)})}$.

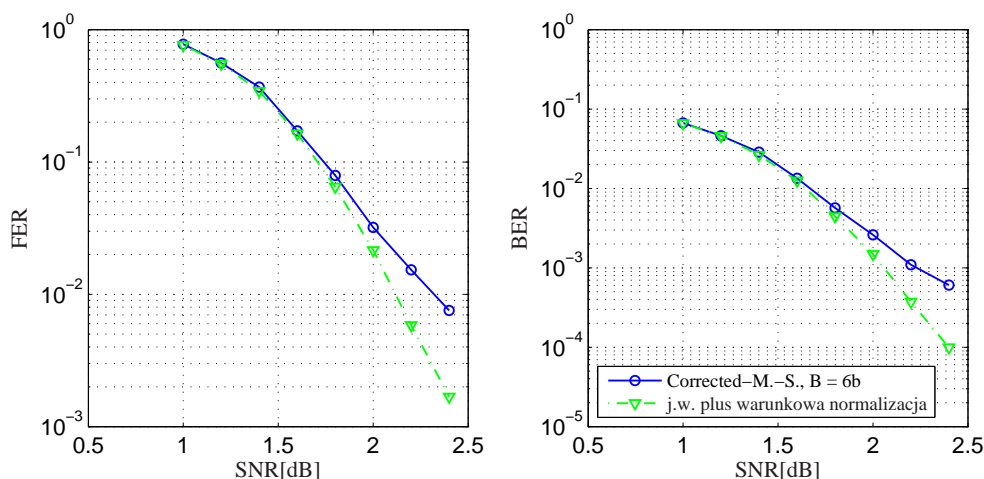
Schemat jednostki Subtr*, w której zaimplementowano warunkową normalizację wartości λ przedstawiono na rysunku 4.24. Dwa subtraktory służą do sprawdzenia spełnienia warunku w (4.43), na podstawie czego multiplekser propaguje wartość wejściową λ lub też wartość znormalizowaną.

Wyniki przykładowych symulacji przy wykorzystaniu dekodera bez oraz z warunkową normalizacją wartości λ przedstawiono na rysunku 4.25. Parametry wynoszące $Q_{th} = Z_{th} - 3\Delta$, $\frac{\lambda_n^{d,p}}{A_\lambda} = 0.5$ zostały dobrane eksperymentalnie. Podobne wyniki uzyskano dla innych kodów i algorytmów wyznaczania wiadomości. Jak widać, zastosowanie opisanej modyfikacji umożliwia znaczne zmniejszenie bitowej stopy błędów systemu, szczególnie dla większych wartości SNR. Wzrost powierzchni dekodera jest przy tym stosunkowo niewielki: rozbudowa o elementy przedstawione na rys. 4.23 i 4.24 skutkuje zwiększeniem powierzchni bloku SISO o około 7%. Biorąc pod uwagę

cały dekodery, odpowiada to zwiększeniu jego powierzchni o około 4%. Tak więc niewielkim kosztem można uzyskać zilustrowaną na rys. 4.25 znaczącą poprawę bitowej stopy błędów systemu.



Rys. 4.24: Blok Subtr* z warunkową normalizacją wartości λ



Rys. 4.25: Wyniki symulacji systemu transmisji dla dekodera kodu regularnego (1008,514) bez oraz z warunkową normalizacją wiadomości

4.6.5 Konfigurowalne sieci zapisu i odczytu

Konfigurowalne sieci zapisu oraz odczytu to sieci multiplekserów o PB wejściach i wyjściach, których specyfikację opracowano w postaci behawioralnego opisu w języku VHDL. Każda z sieci zawiera jedną warstwę rejestrów potokowych, stąd opóź-

nienie każdej z nich wynosi:

$$T_{\text{MUX}} = 1 \quad (4.44)$$

4.6.6 Optymalizacja przetwarzania potokowego

Użycie w jednostkach obliczeniowych dużej liczby rejestrów potokowych umożliwiło uzyskanie wysokiej częstotliwości zegarowej. Przykładowo, jak pokazały wyniki syntezy dla układu rodziny VirtexII, maksymalna częstotliwość zegarowa układu bez rejestrów potokowych w blokach Subtr* oraz Sum* wynosi około 30MHz, podczas gdy układu z rejestrami około 100MHz. Z drugiej strony, ze względu na opóźnienie toru przetwarzania spowodowane wprowadzeniem rejestrów, dla przypadków gdy w dwóch kolejnych subiteracjach wykorzystywana jest ta sama porcja danych z pamięci, konieczne jest wprowadzenie cykli jałowych w celu odczekania na pojawienie się uaktualnionych danych. Korzyść z zastosowania przetwarzania potokowego byłaby zatem niewielka, gdyby taka sytuacja (konieczności wykonywania cykli jałowych) powtarzała się w wielu subiteracjach. Stąd też konieczne stało się przeprowadzenie głębokiej analizy, w wyniku której opracowano algorytm optymalizacji macierzy, pozwalający na minimalizację niekorzystnych przestoju obliczeń realizowanych w procesie dekodowania.

Wyznaczanie liczby cykli jałowych

Całkowite opóźnienie toru składającego się z jednostki SISO oraz konfigurowalnych sieci zapisu i odczytu, które będzie nazywane opóźnieniem przetwarzania potokowego T_P , wynosi:

$$T_P = 2T_{\text{MUX}} + T_{\text{SISO}} = 11 \quad (4.45)$$

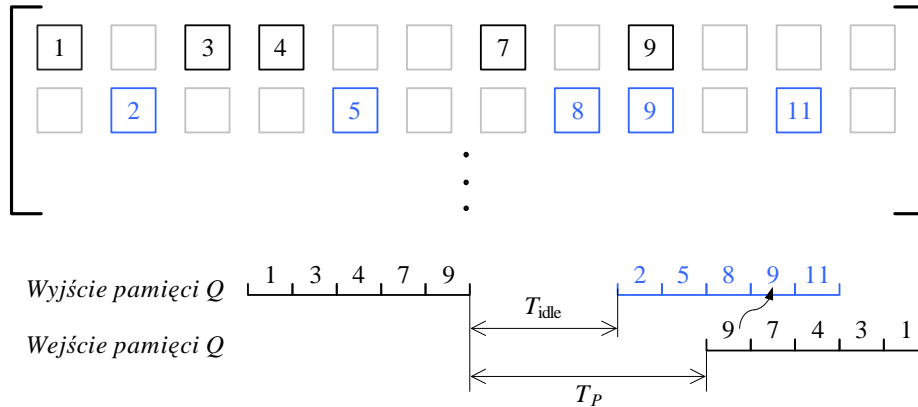
gdzie T_{MUX} to opóźnienie konfigurowalnej sieci zapisu / odczytu (4.44), natomiast T_{SISO} to sumaryczne opóźnienie jednostki SISO wyznaczone w (4.29).

Liczba cykli jałowych T_{idle}^m niezbędnych do rozpoczęcia subiteracji m , gdzie $m = 0, \dots, D - 1$, zależy od tego, czy wiersz m macierzy bazowej zawiera jedynki w tych samych kolumnach co wiersze $(m - 1) \bmod D$, $(m - 2) \bmod D$, $(m - 3) \bmod D$. Najlepiej można to zilustrować za pomocą przykładu.

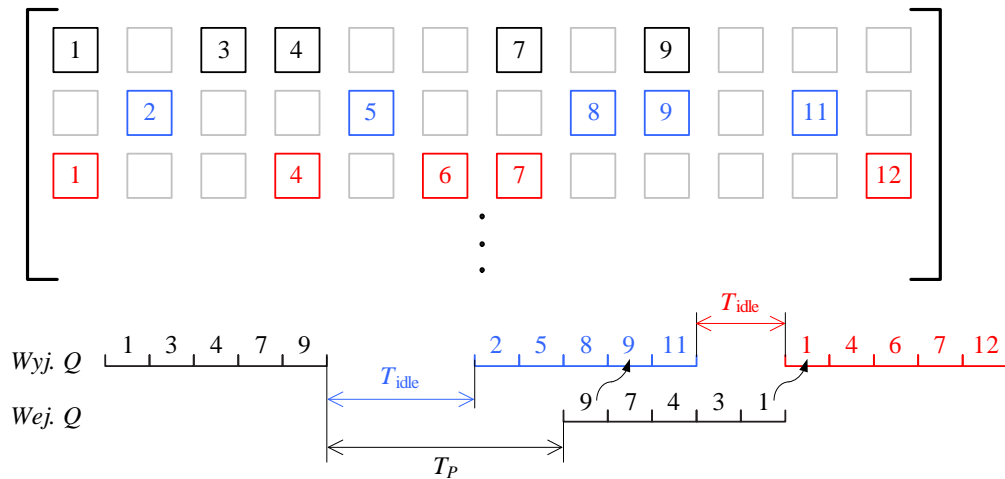
Przykład 6. Określenie czasów bezczynności dekodera

Na rysunku 4.26 przedstawiono fragment macierzy kontrolnej kodu, na którym ponumerowane kwadraty oznaczają podmacierze permutacji odpowiadające jedynkom macierzy bazowej. Poniżej pokazano kolejność przesyłania danych z / do pamięci Q (numery komórek pamięci, z których pobierane / zapisywane są dane, odpowiadające kolejnym kolumnom macierzy bazowej). Zanim

zostanie rozpoczęta subiteracja dla wiersza oznaczonego na niebiesko, niezbędne jest wykonanie takiej liczby cykli bezczynności T_{idle} , aby komórka o numerze 9 była odczytywana dopiero po zapisaniu w niej wartości uaktualnionej w poprzedniej subiteracji, oznaczonej na czarno. Konieczność wykonywania cykli jałowych jest zatem spowodowana tym, że w kolejnych dwóch subiteracjach jedna (lub więcej) spośród przetwarzanych danych (wiadomości) jest wspólna, co jest wynikiem „nakładania” się jedynek w kolejnych wierszach macierzy bazowej (tutaj: w kolumnie 9).



Rys. 4.26: Przykładowy fragment macierzy kontrolnej i kolejność danych na portach pamięci



Rys. 4.27: Przykładowy fragment macierzy kontrolnej i kolejność danych na portach pamięci

Jeśli dwa kolejne wiersze macierzy bazowej nie zawierają jedynek we wspólnej kolumnie, mimo wszystko pewien czas bezczynności może być konieczny. Ilustruje to rysunek 4.27 przedstawiający

sytuację, w której dla rozpoczęcia rozpatrywanej subiteracji (wiersz czerwony) nie jest konieczne oczekiwanie na uaktualnienie danych wyznaczonych w poprzedniej (wiersz niebieski), niezbędne jest jednak wstrzymanie pracy do czasu zapisania danych z subiteracji jeszcze wcześniejszej (wiersz czarny). W tym przypadku owe dwa wiersze (czerwony i czarny) zawierają jedynkę we wspólnej kolumnie 1.

Okazuje się, że dla danej macierzy bazowej, istnieje możliwość wyznaczenia liczby cykli bezczynności T_{idle}^m w subiteracji m za pomocą odpowiedniego wzoru, który został w tym celu zaproponowany. We wzorze wykorzystywana jest pewna pomocnicza zmienna $X_{(m_2, m_1)}$, która definiowana jest następująco:

- jeśli wiersze m_2 i m_1 macierzy bazowej nie zawierają jedynki we wspólnej kolumnie, to $X_{(m_2, m_1)} = \infty$
- jeśli wiersze m_2 i m_1 zawierają jedynkę we wspólnej kolumnie l , to $X_{(m_2, m_1)}$ równe jest różnicy liczby jedynek w wierszu m_2 w kolumnach o indeksach mniejszych od l oraz liczby jedynek w wierszu m_1 w kolumnach o indeksach większych od l , pomniejszonej o jeden.

Odwołując się do powyższego przykładu, dla przypadku z rys. 4.26, wartość $X_{(m_2, m_1)}$ (gdzie m_1 odnosi się do pierwszego, a m_2 do drugiego wiersza) wynosi $X_{(m_2, m_1)} = 4 - 1 - 1 = 2$. Łatwo zauważyć, że liczba $X_{(m_2, m_1)}$ oznacza różnicę pomiędzy wartością opóźnienia przetwarzania potokowego T_P a liczbą cykli jałowych.

Liczba cykli jałowych T_{idle}^m , jeśli rozpatrzeć tylko oczekiwanie na zakończenie subiteracji $m-1$ powinna być zatem równa różnicy pomiędzy T_P i $X_{(m, m-1)}$, a ściślej rzecz biorąc:

$$T_1^m = \max \left[0, (T_P - X_{(m, m-1)}) \right], \quad 1 \leq m \leq D - 1 \quad (4.46)$$

Z kolei jeśli konieczne jest oczekiwanie na zakończenie subiteracji $m-2$ (czyli przypadek z rys. 4.27), to liczba cykli jałowych wynosi:

$$T_2^m = \max \left[0, (T_P - X_{(m, m-2)} - d_{c_{m-1}} - T_1^m) \right], \quad 2 \leq m \leq D - 1 \quad (4.47)$$

gdzie $d_{c_{m-1}}$ to waga wiersza $m-1$ równa liczbie cykli pobierania / zapisywania danych do pamięci. W bardzo rzadkim przypadku, konieczność wykonania cykli jałowych przed subiteracją m wynika z oczekiwania na zakończenie subiteracji $m-3$. Dla tego przypadku liczba cykli jałowych:

$$T_3^m = \max \left[0, (T_P - X_{(m, m-3)} - d_{c_{m-1}} - T_1^m - d_{c_{m-2}} - T_2^m) \right], \quad 3 \leq m \leq D - 1 \quad (4.48)$$

Uogólnienie wzorów (4.46)-(4.48) na przypadek $0 \leq m \leq D-1$ wymaga zastąpienia występujących tam operacji odejmowania $(m-1)$, $(m-2)$, $(m-3)$ odpowiednio przez wyrażenia $(m-1) \bmod D$, $(m-2) \bmod D$, $(m-3) \bmod D$.

Wyznaczenie liczby cykli T_{idle}^m wymaga rozpatrzenia najgorszego spośród wymienionych wyżej przypadków, co można wyrazić następująco:

$$T_{\text{idle}}^m = \max [T_1^m, T_2^m, T_3^m] \quad (4.49)$$

Algorytm optymalizacji macierzy bazowej

W celu minimalizacji czasu bezczynności, a co za tym idzie znacznego zwiększenia przepustowości dekodera, opracowano heurystyczny algorytm optymalizacji macierzy bazowej kodu AA-LDPC, który polega na takim posortowaniu kolumn i wierszy macierzy bazowej, aby sumaryczna liczba cykli bezczynności:

$$T_{\text{idle}} = \sum_{m=0}^{D-1} T_{\text{idle}}^m \quad (4.50)$$

była możliwie jak najmniejsza. Algorytm przedstawiono na stronie 78 jako algorytm 4.4.

Pierwszy krok polega na sortowaniu kolumn macierzy \mathbf{W} , według rosnących wag. Jest to podyktowane spostrzeżeniem, że większe czasy bezczynności występują w sytuacji, gdy jedynki we wspólnych kolumnach umieszczone są przy mniejszych indeksach kolumn (rys. 4.26 i 4.27). Przesunięcie kolumn o większej wadze na prawą stronę macierzy zmniejsza prawdopodobieństwo tej niekorzystnej sytuacji.

Następnie poszukiwana jest macierz \mathbf{W}' , składająca się z wszystkich wierszy \mathbf{W} , ułożonych w możliwie najlepszej kolejności ze względu na parametr T_{idle} wyznaczany zgodnie z (4.50). Pierwszy wiersz wybierany jest w sposób losowy, a każdy kolejny (m -ty) selekcionowany jest spośród tych, które umożliwiają osiągnięcie najmniejszej wartości T_{idle}^m wyznaczonej za pomocą wzorów (4.46)-(4.49).

Ze względu na fakt, że w dokonywanych wyborach istnieje pewien element losowości, opisane czynności są powtarzane pewną liczbę razy (1000), a spośród otrzymanych macierzy wybierana jest ta o najmniejszej wartości parametru T_{idle} . Jak wykazały eksperymenty, zwiększenie liczby powtórzeń powyżej 1000 w zdecydowanej większości przypadków nie pozwala znacząco poprawić uzyskanego rezultatu.

Algorytm 4.4: Sortowanie kolumn i wierszy macierzy bazowej prowadzące do minimalizacji czasu bezczynności dekodera	
Dane wejściowe: Macierz bazowa $\mathbf{W}_{D \times L}$, opóźnienie przetwarzania potokowego T_P	
Dane wyjściowe: Przekształcona macierz bazowa \mathbf{W}'	
1	Posortować kolumny \mathbf{W} według ich wag: kolumny o najmniejszej wadze po lewej stronie macierzy.
2	$T_{\text{opt}} := \infty$
3	DLA $k=1, \dots, 1000$
4	$\mathbf{W}' := \mathbf{0}$
5	Wybrać losowo wiersz macierzy \mathbf{W} i zapisać go jako pierwszy wiersz \mathbf{W}' .
6	DLA $m = 2, \dots, D$
7	Spośród niewybranych dotychczas (w aktualnej iteracji k) wierszy macierzy \mathbf{W} wybrać taki, który wstawiony jako wiersz m -ty do macierzy \mathbf{W}' daje najmniejszą liczbę cykli T_{idle}^m wyznaczoną zgodnie ze wzorem (4.49). Jeśli więcej niż jeden wiersz spełnia ten warunek, to spośród nich dokonać wyboru losowego.
8	Zapisać wybrany wiersz jako m -ty wiersz macierzy \mathbf{W}' i usunąć z \mathbf{W} .
9	Wyznaczyć sumaryczną liczbę cykli bezczynności T_{idle} dla macierzy \mathbf{W}' , zgodnie ze wzorem (4.50)
10	JEŻELI $T_{\text{idle}} < T_{\text{opt}}$
11	$\mathbf{W}_{\text{opt}} := \mathbf{W}'$
12	$T_{\text{opt}} := T_{\text{idle}}$
13	$\mathbf{W}' := \mathbf{W}_{\text{opt}}$

W tabeli 4.1 przedstawiono wartości sumarycznej liczby cykli jałowych dekodera T_{idle} dla macierzy bazowych kilku kodów oraz przekształconych macierzy \mathbf{W}' otrzymanych za pomocą algorytmu 4.4, przy wartości opóźnienia przetwarzania potokowego $T_P = 11$. Dla kodów o stopie $R = 0.5$ we wszystkich przypadkach uzyskano wartość T_{idle} bliską lub równą 0. Nieco mniej korzystna jest sytuacja dla kodów o wyższych stopach ($R = 0.75$), gdyż dla tych przypadków w macierzy \mathbf{W} występuje mniej wierszy, co (przy niezmiennych wagach kolumn) powoduje większe zagęszczenie jedynek. Trudniej jest zatem uzyskać sytuację, w której w kolejnych wierszach nie powtarzają się jedynki na tych samych pozycjach.

Podsumowując, opracowany algorytm sortowania wierszy / kolumn macierzy kontrolnej umożliwia znaczące zmniejszenie liczby cykli bezczynności, a co za tym

Tab. 4.1: Przykładowe liczby cykli bezczynności dekodera

Parametry macierzy bazowej	T_{idle} dla \mathbf{W}	T_{idle} dla \mathbf{W}'
$\mathbf{W}_{55 \times 110}$ ($R = 0.5$), regularna $d_b = 3$	217	0
$\mathbf{W}_{64 \times 128}$ ($R = 0.5$), nieregularna $d_{b_{\max}} = 15$	813	0
$\mathbf{W}_{32 \times 64}$ ($R = 0.5$), nieregularna $d_{b_{\max}} = 8$	311	8
$\mathbf{W}_{16 \times 64}$ ($R = 0.75$), regularna $d_b = 3$	253	80
$\mathbf{W}_{10 \times 44}$ ($R = 0.77$), regularna $d_b = 3$	204	121

idzie wykorzystanie w pełni zalet przetwarzania potokowego w dekodерze. Algorytm jest szczególnie skuteczny dla kodów o stosunkowo niskich stopach. Dzięki temu osiągnięte wielkości przepustowości dekodera są wysokie w stosunku do wymaganych zasobów, co zostanie pokazane w podrozdziale prezentującym przykładowe wyniki implementacji.

4.7 Środowisko symulacji systemów kodowania LDPC

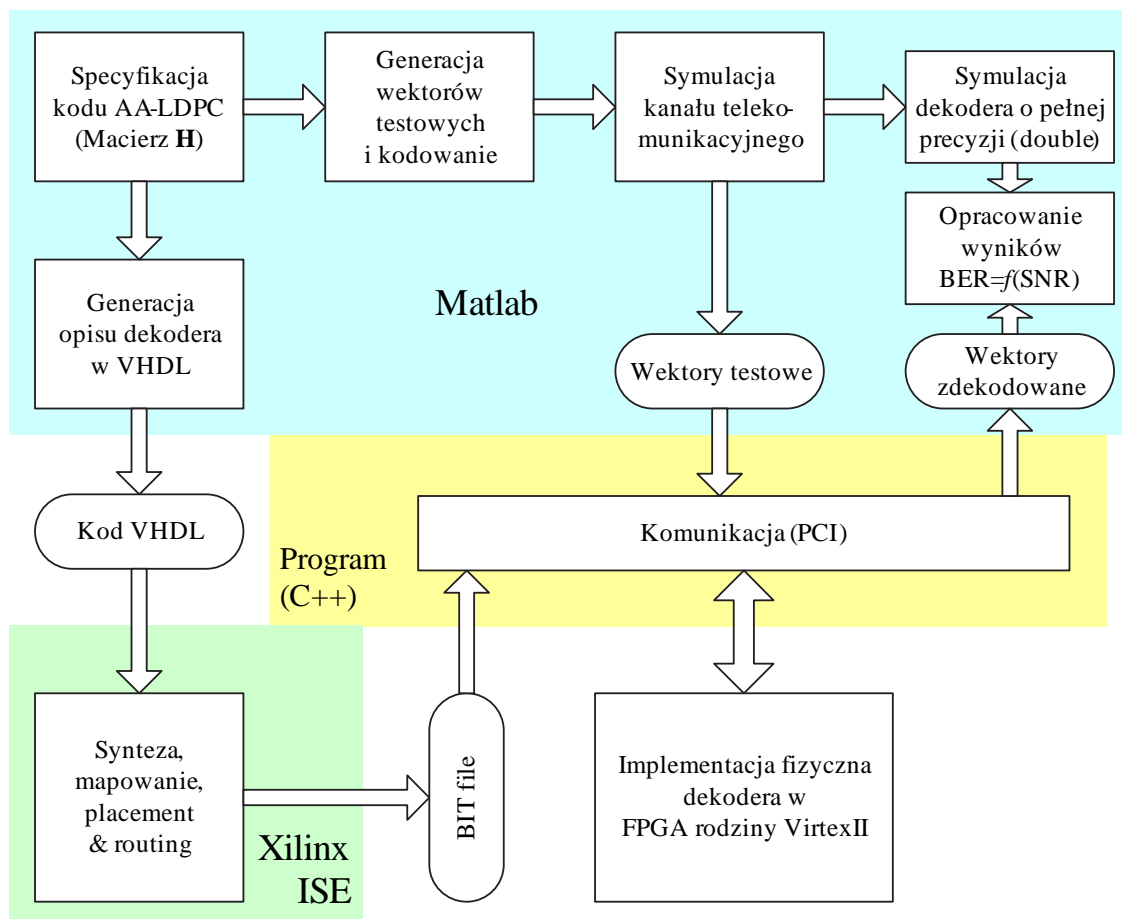
Dla weryfikacji poprawności działania implementowanych w strukturach programowalnych dekodерów oraz jakości opracowanych kodów AA-LDPC koniecznym było opracowanie kompleksowego środowiska projektowania i symulacji systemów transmisji wykorzystujących kodowanie LDPC.

Zintegrowane środowisko projektowania i symulacji systemów kodowania LDPC zostało zatem opracowane dla realizacji następujących zadań:

- weryfikacji dobrych własności dekodera zaimplementowanego w układzie programowalnym typu FPGA i porównania poszczególnych algorytmów wyznaczania wiadomości w dekodерze pod względem ich jakości (bitowej stopy błędów systemu)
- weryfikacji celowości zaproponowanych modyfikacji algorytmów wyznaczania wiadomości,
- wyznaczania własności korekcyjnych kodów AA-LDPC z wykorzystaniem platformy sprzętowej typu FPGA znacznie przyspieszającej obliczenia.

W celu integracji środowiska symulacji dekodera z narzędziami zapewniającymi dużą swobodę projektowania algorytmów tworzenia macierzy kontrolnej, środowisko oparte zastało na oprogramowaniu MATLAB. Do implementacji dekodera wykorzystano narzędzia programistyczne firmy Xilinx oraz zestaw uruchomieniowy zawiera-

jący układ FPGA rodziny VirtexII. Główne elementy środowiska z uwzględnieniem sposobów komunikacji przedstawiono na rys. 4.28 [98].



Rys. 4.28: Środowisko projektowania i symulacji systemów kodowania LDPC

Dla konkretnej macierzy kontrolnej (np. uzyskanej za pomocą technik, które będą opisane w kolejnym rozdziale) konieczne jest dostosowanie fragmentów opisu dekodera w języku opisu sprzętu (VHDL). W szczególności konieczne jest zdefiniowanie zawartości pamięci struktury macierzy bazowej oraz pamięci permutacji, określenie liczby jednostek SISO oraz pewne modyfikacje modułu sterowania. W celu zapewnienia wygody użytkownika oraz możliwości szybkiej weryfikacji własności systemu kodowania, opracowano skrypty umożliwiające automatyczne dostosowanie opisu dekodera w języku VHDL. Dostosowanie to wykonywane jest na podstawie macierzy kontrolnej kodu oraz pewnego zestawu parametrów definiujących format słów B , Z_{th} oraz algorytm wyznaczania wiadomości (dowolny spośród opisanych w niniejszym rozdziale). Pełen opis dekodera, składający się z wielu plików zawierających

sumarycznie typowo ok. 2000 linii kodu źródłowego, jest poddawany procesowi syntezy, rozmieszczania elementów i prowadzenia połączeń dla układu FPGA (*mapping, placement & routing*), za pomocą narzędzi wchodzących w skład pakietu Xilinx ISE.

Uzyskany moduł dekodera sprzętowego implementowany jest na płycie ewaluacyjnej wyposażonej w układ rodziny VirtexII, a komunikacja z modułem odbywa się z wykorzystaniem złącza PCI. Do dekodera przesyłane są dane, składające się z wartości inicjalizujących, tzn. wielkości LLR prawdopodobieństw *a priori* dla kolejnych bitów bloku. Dane przygotowywane są w środowisku MATLAB, w którym wykonywana jest losowa generacja bloków danych, które są następnie kodowane oraz symulacja pracy modulatora, kanału oraz demodulatora. W eksperymentach prezentowanych w niniejszej pracy wykorzystano dwuwartościowe techniki modulacji oraz model kanału AWGN. Tym niemniej nic nie stoi na przeszkodzie, aby opracowane środowisko uzupełnić o dowolne inne modele kanałów i systemów modulacji.

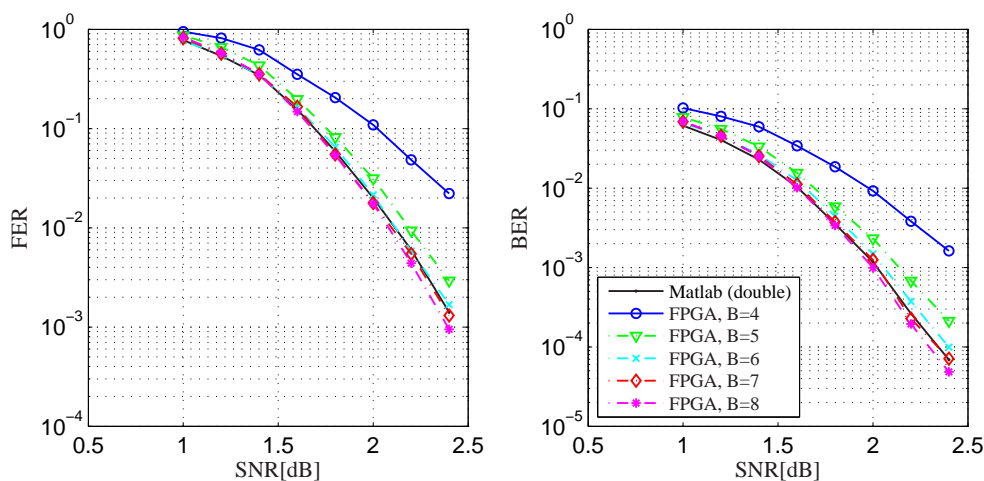
Implementacja dekodera w strukturze FPGA znacznie przyspiesza obliczenia. Zaobserwowano, że w porównaniu z dekodowaniem przy wykorzystaniu oprogramowania MATLAB, dekodery w strukturze FPGA umożliwiają ponad 500-krotne przyspieszenie obliczeń. Dodatkowo możliwa jest weryfikacja skuteczności dekodowania sprzętowego, z wykorzystaniem wiadomości o ograniczonej precyzji.

4.8 Analiza porównawcza algorytmów dekodowania

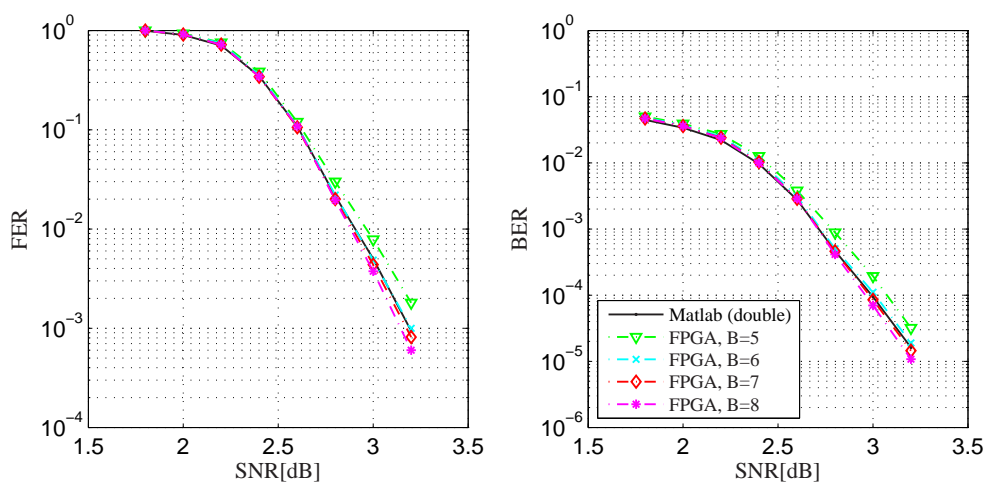
Świadomy wybór parametrów dekodera jest bardzo istotnym elementem projektowania dobrego systemu sprzętowego dekodowania LDPC. Pierwszym etapem tego wyboru powinny być eksperymenty umożliwiające porównanie poszczególnych sposobów konfiguracji dekodera ze względu na jego własności korekcyjne. Wyniki takich eksperymentów zaprezentowane są w niniejszym podrozdziale. Wszystkie eksperymenty przeprowadzono z wykorzystaniem dekodera wyposażonego w warunkową normalizację wiadomości λ (modyfikacja opisana w podrozdziale 4.6.4).

Na rysunkach 4.29 oraz 4.30 przedstawiono wykresy bitowej i blokowej stopy błędów otrzymane przy zastosowaniu dekodery o różnych długościach słowa wiadomości B . Oczywistym jest, że większa długość słowa zapewnia lepsze własności dekodera. Co interesujące, dla dekodera 8-bitowego uzyskuje się zwykle wyniki nieznacznie lepsze niż dla algorytmu LLR-BP operującego na liczbach zmiennoprzecinkowych (MATLAB). Wynika to stąd, że algorytm dekodowania jest tak czy inaczej suboptymalny, a pewne (niewielkie) zmniejszenie dokładności obliczeń wraz z zaproponowanymi modyfikacjami algorytmu może skutkować poprawą skuteczności.

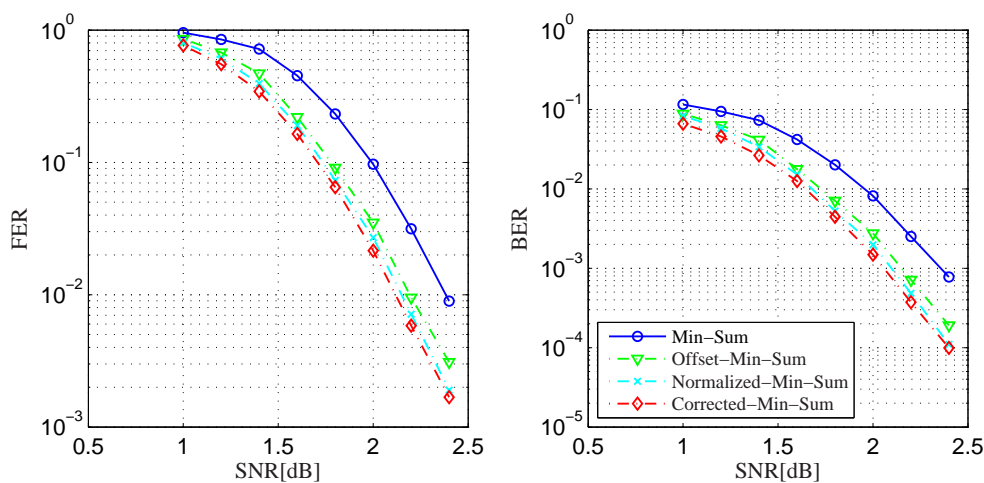
Należy zauważyć, że ograniczenie precyzji do $B = 4$ bitów skutkuje znacznym pogorszeniem jakości dekodera (rys. 4.29). Do praktycznego zastosowania najwłaściwszym wyborem jest zatem zwykle długość słowa $B = 5$ lub $B = 6$. Taki wniosek można wyciągnąć analizując rys. 4.29 i 4.30, odnoszące się do kodów o stopach odpowiednio $R = 0.5$ i $R = 0.75$. Podobne wyniki uzyskano dla wielu innych kodów.



Rys. 4.29: Porównanie wyników dla różnych długości słowa B , algorytm Corrected-Min-Sum, kod regularny (1008,514)

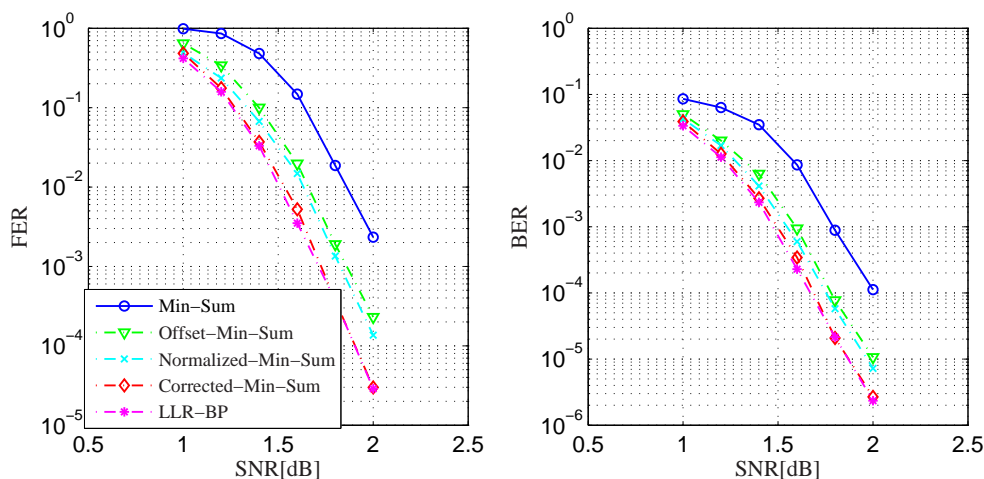


Rys. 4.30: Porównanie wyników dla różnych długości słowa B , algorytm Corrected-Min-Sum, kod regularny (2048,1536)



Rys. 4.31: Porównanie algorytmów wyznaczania wiadomości, dla $B = 6$, kod regularny (1008,514)

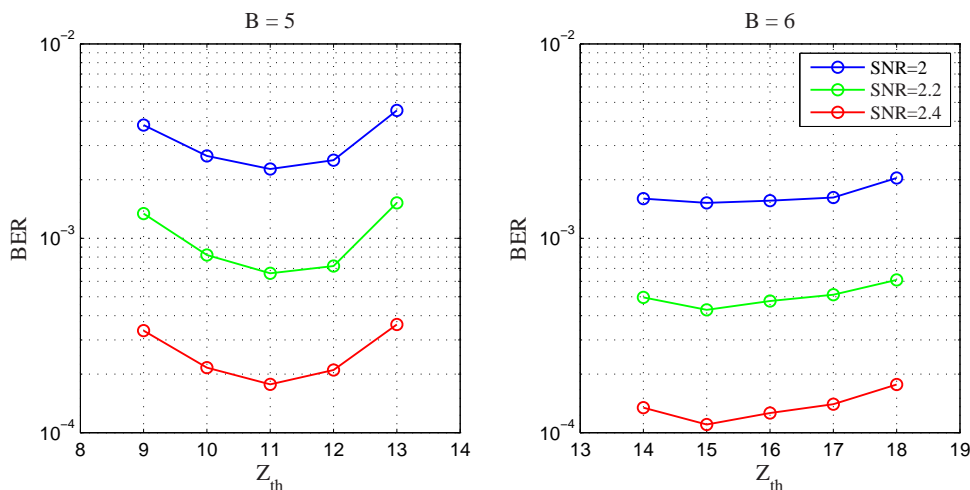
Kolejnym elementem analizy jest porównanie poszczególnych algorytmów wyznaczania wiadomości. Wykresy ilustrujące to zagadnienie dla przykładowych kodów: regularnego oraz nieregularnego zaprezentowano na rys. 4.31 i 4.32. Algorytm Corrected-Min-Sum daje bardzo zbliżone wyniki do podstawowego algorytmu LLR-BP (rys. 4.32). Nieco gorsze rezultaty (szczególnie dla kodów nieregularnych, rys. 4.32) dają algorytmy Offset-Min-Sum (wyznaczone eksperymentalnie



Rys. 4.32: Porównanie algorytmów wyznaczania wiadomości, dla $B = 6$, kod nieregularny (2048,1024)

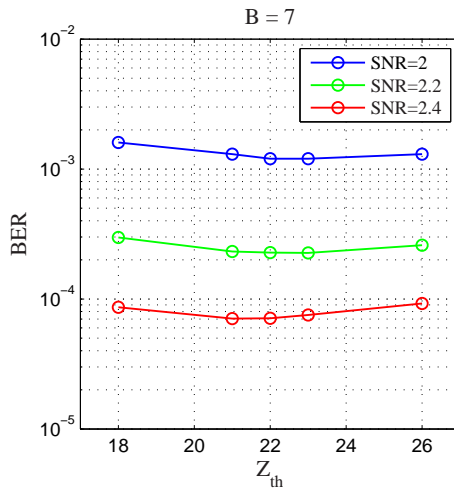
$C = 0.5$) oraz Normalized-Min-Sum (czynnik $1/A = 0.75$). Jak zostanie pokazane w kolejnym podrozdziale, umożliwiając one jednak uzyskanie większej częstotliwości zegarowej oraz mniejszych zasobów sprzętowych dekodera. Jak należało się spodziewać, zdecydowanie najmniej korzystny ze względu na bitową stopę błędów systemu jest algorytm Min-Sum. Podsumowując, uszeregowanie od najlepszego do najgorszego algorytmu ze względu na możliwości korekcyjne jest następujące: 1) LLR-BP, 2) Corrected-Min-Sum, 3) Normalized Min-Sum, 4) Offset-Min-Sum, 5) Min-Sum.

Dosyć istotny jest odpowiedni dobór wartości progu obciążenia wiadomości Z_{th} [97]. Na rysunku 4.33 przedstawiono zależności bitowej stopy błędów od wartości tego progu dla wiadomości 5-bitowych oraz 6-bitowych, a na rys. 4.34 – 7-bitowych. Można zauważyć, że dla $B = 5$ odpowiedni dobór wartości Z_{th} jest bardzo ważny, a nieco mniej krytyczny dla dłuższych długości słowa B . Jak wykazały eksperymenty przeprowadzane dla wielu kodów, najlepsze rezultaty uzyskuje się zwykle dla wartości w przybliżeniu równych: $Z_{th} = 11$ przy $B = 5$, $Z_{th} = 15$ przy $B = 6$, $Z_{th} = 22$ przy $B = 7$ oraz $Z_{th} = 32$ przy $B = 8$.



Rys. 4.33: Porównanie wyników dla różnych wartości Z_{th} , $B = 5$ oraz $B = 6$, kod regularny (1008,514)

Podsumowując, z przeprowadzonych eksperymentów wynika, że najkorzystniejszym wyborem do praktycznych zastosowań będzie algorytm Corrected-Min-Sum lub Normalized-Min-Sum o czynniku $1/A = 0.75$, przy parametrach słowa wiadomości $B = 6$, $Z_{th} = 14$ lub $B = 5$, $Z_{th} = 11$. Dla konkretnego kodu, wybór algorytmu powinien być jednak poprzedzony eksperymentami, co można natychmiast wykonać wykorzystując opracowane środowisko symulacji.



Rys. 4.34: Porównanie wyników dla różnych wartości Z_{th} , $B = 7$, kod regularny (1008,514)

Na potrzeby eksperymentów dotyczących testowania własności kodów AA-LDPC, które zostaną przedstawione w dalszej części pracy, wykorzystano algorytm Corrected-Min-Sum o parametrach $B = 7$, $Z_{th} = 22$ (dobór $Z_{th} = 22$ – por. rys. 4.34). Wybór ten został podyktowany spostrzeżeniem, że własności takiego dekodera są najbardziej zbliżone do własności modelu dekodera w środowisku MATLAB (rys. 4.29-4.30), co umożliwi obiektywne porównanie kodów AA-LDPC dekodowanych na platformie FPGA z kodami LDPC dekodowanymi z wykorzystaniem programu MATLAB.

4.9 Wyniki implementacji

Omówione w poprzednim podrozdziale własności korekcyjne dekodowników o różnych algorytmach i parametrach powinny być rozpatrywane w kontekście wyników ich implementacji: przepustowości oraz wymaganych zasobów. Analiza wyników syntezy, jak również porównanie parametrów zaprojektowanego dekodera z implementacjami zaczerpniętymi z literatury zostanie przedstawiona w niniejszym podrozdziale. Ze względu na fakt, że używano oprogramowanie i sprzęt firmy Xilinx, spośród doniesień literaturowych wybrano głównie implementacje w układach właśnie tej firmy.

Parametry zaprojektowanego dekodera pochodzą z raportów syntezy wykonywanej za pomocą programu XST z pakietu Xilinx ISE wersji 8.2, dla układu XC2V3000-4 rodziny VirtexII (układ ten jest wykorzystywany w eksperymentach). Najistotniejsze parametry to liczba wykorzystywanych podstawowych komórek (tzw. Slice'ów)

układu FPGA i liczba bloków pamięci (BRAM) charakteryzujące złożoność uzyskanego modułu oraz maksymalna częstotliwość zegarowa.

Na podstawie maksymalnej częstotliwości zegarowej można wyznaczyć przepustowość dekodera, definiowaną jako liczba zdekodowanych bitów informacyjnych na sekundę. Przepustowość zależy od liczby cykli koniecznych do wykonania jednej iteracji dekodowania T_{iter} , która jest równa sumie liczby cykli obliczeniowych oraz cykli bezczynności. W subiteracji m wykonywanych jest tyle cykli obliczeniowych, ile wynosi waga wiersza m -tego macierzy bazowej (d_{c_m}). Liczba cykli obliczeniowych we wszystkich subiteracjach jest zatem równa $\sum_m(d_{c_m})$, stąd:

$$T_{\text{iter}} = \sum_m(d_{c_m}) + T_{\text{idle}} \quad (4.51)$$

gdzie T_{idle} to czas bezczynności wyznaczony zgodnie z (4.50).

Przepustowość TH (ang. *throughput*) dekodera zależy od wartości T_{iter} , liczby cykli inicjalizacji dekodera T_{init} (patrz (4.11)), częstotliwości zegarowej f_{clk} oraz liczby iteracji dekodowania i_{max} w sposób następujący:

$$TH = \frac{f_{\text{clk}}M}{T_{\text{iter}}i_{\text{max}} + T_{\text{init}}} \quad [b/s] \quad (4.52)$$

Autorzy publikacji przy wyznaczaniu przepustowości przyjmują różne założenia co do liczby iteracji. Można przyjąć wartość maksymalną i_{max} (jak we wzorze (4.52)), która zwykle jest rzędu kilkudziesięciu (wówczas rozpatrywany jest „najgorszy” przypadek) lub też średnią wartość liczby iteracji, wyznaczoną statystycznie i wynoszącą zwykle mniej niż 10. Można również wyznaczać wartość przepustowości na jedną iterację dekodowania. W celu ujednoczenia prezentowanych wyników, w niniejszej pracy przedstawiana jest wartość przepustowości przy założeniu 10 iteracji, a wszystkie wyniki pochodzące ze źródeł zostały odpowiednio przeskalowane.

Przykład 7. Wyznaczenie przepustowości dekodera kodu regularnego (2640,1320)

Dla kodu regularnego o macierzy bazowej $\mathbf{W}_{55 \times 110}$, liczba cykli jałowych wynosi $T_{\text{idle}} = 0$ (tab. 4.1). Wykonanie jednej iteracji wymaga $T_{\text{iter}} = \sum_m(d_{c_m}) + T_{\text{idle}} = 55 \cdot 6 + 0 = 330$ cykli zegarowych. Inicjalizacja dekodera oraz wykonanie 10 iteracji zajmuje $110 + 10 \cdot 330 = 3410$ cykli.

Dla kodu o wielkości podmacierzy $P = 24$, blok ma długość $N = 2640$, a liczba bitów informacyjnych $M = 1320$. Jeden blok danych, zawierający M bitów informacyjnych, dekodowany jest w czasie $t_{\text{dek}} = 3410/f_{\text{clk}}$. Przepustowość dekodera dla $f_{\text{clk}} = 95$ MHz wynosi zatem $TH = M/t_{\text{dek}} = 1320 \cdot 95/3410 = 36,8$ Mb/s. Wartość tą można otrzymać podstawiając odpowiednie dane bezpośrednio do wzoru (4.52).

Zaczerpnięte z literatury parametry kilku dekoderek implementowanych w układach rodziny Virtex przedstawiono w tabeli 4.2, a w tabeli 4.3 – parametry innych rozwiązań: dekodera w układzie FPGA firmy Altera oraz układzie typu ASIC. Tabela 4.4 prezentuje parametry zaprojektowanego dekodera dla kilku różnych kodów.

Tab. 4.2: Wyniki zaczerpnięte z literatury: dekodery w układach rodziny Virtex

Źródło	[95]
Układ	FPGA, VirtexII XC2V6000-5
Obsługiwane kody	<i>Array based</i> , regularne $d_b = 3$, $d_c = 6$, maks. $N = 1536$
Kod	Regularny $d_b = 3$, $N = 1536$, $R = 1/2$
Algorytm dekodowania	Min-Sum, 8-bit
Częstotliwość zegara	100MHz
Przepustowość (10 iter.)	7,6Mb/s
Zasoby	1296 Slices, 102 BRAMs
Źródło	[124]
Układ	FPGA, VirtexII Pro XC2VP50
Obsługiwane kody	RS-LDPC, strukturalne, regularne $d_b = 6$, $d_c = 32$
Kod	Regularny $d_b = 6$, $N = 2048$, $R = 0.8125$
Algorytm dekodowania	LLR-BP, 6-bit
Częstotliwość zegara	100MHz
Przepustowość (10 iter.)	24Mb/s
Zasoby	7320 Slices, 129 BRAMs
Źródło	[117]
Układ	FPGA, VirtexII XC2V8000-6
Obsługiwane kody	AA-LDPC, regularne i nieregularne
Kod	Regularny, $N = 9000$, $R = 1/2$
Algorytm dekodowania	Min-Sum z warunkową korekcją wartością stałą, 6-bit
Częstotliwość zegara	100MHz
Przepustowość (10 iter.)	90Mb/s
Zasoby	34127 Slices, 102 BRAMs
Źródło	[112]
Układ	FPGA, VirtexII XC2V6000-6
Obsługiwane kody	QC-LDPC, regularne $d_b = 4$, $d_c = 32$
Kod	Regularny, $N = 8176$, $R = 0.875$
Algorytm dekodowania	LLR-BP, 6-bit
Częstotliwość zegara	192MHz
Przepustowość (10 iter.)	258Mb/s
Zasoby	23053 Slices, 128 BRAMs

Porównując uzyskane wyniki należy podkreślić, że zaprojektowany dekoderek jest niewątpliwie konkurencyjny do znanych z literatury implementacji. Przykładowo je-

Tab. 4.3: Wyniki zaczerpnięte z literatury: dekodery w innych układach

Źródło	[51]
Układ	FPGA, Altera Excalibur EPXA10
Obsługiwane kody	QC-LDPC, regularne
Kod	Regularny $d_b = 3$, $N = 12288$, $R = 1/2$
Algorytm dekodowania	Nie podano
Częstotliwość zegara	32MHz
Przepustowość	19.1Mb/s
Zasoby	3064 LCs, BRAM 307200b
Źródło	[68]
Układ	0.18 μ m CMOS
Obsługiwane kody	AA-LDPC, regularne $d_b = 3$, $d_c = 6$
Kod	Regularny, $N = 2048$, $R = 1/2$
Algorytm dekodowania	Min-Sum z korekcją funkcją odcinkowo-liniową, 4-bit
Częstotliwość zegara	125MHz
Przepustowość	640Mb/s
Zasoby	14.3mm ²

Tab. 4.4: Wyniki implementacji zaprojektowanego dekodera, różne kody

Kod	Algorytm	f_{clk}	TH	Slices	BRAMs
(1024,512), reg., $P = 16$	Cor.-M.-S., $B = 6$	95MHz	24,5Mb/s	2616	8
(2640,1320), reg., $P = 24$	Cor.-M.-S., $B = 6$	95MHz	36,8Mb/s	4631	13
(2048,1024), niereg., $P = 32$	Cor.-M.-S., $B = 6$	95MHz	43,2Mb/s	6654	17
(2048,1536), reg., $P = 32$	Cor.-M.-S., $B = 6$	95MHz	70Mb/s	6511	17

go wymagane zasoby są nieco mniejsze niż zasoby dekodera [124], a jednocześnie przepustowość jest prawie 3-krotnie większa. W porównaniu do dekoderek przedstawionych w [112, 117] przepustowość zaprojektowanego dekodera jest mniejsza, jednak wykorzystywane zasoby struktury programowalnej są znacząco mniejsze. Należy zauważyć, że autorzy owych prac wykorzystują większe struktury FPGA (XC2V6000, XC2V8000) niż struktura wykorzystywana na potrzeby niniejszej pracy (XC2V3000).

W tabeli 4.5 przedstawiono przykładowe wymagane zasoby strukturalne pojedynczego modułu SISO realizującego różne algorytmy wyznaczania wiadomości, przy zastosowaniu pamięci wiadomości λ w postaci rozproszonej (wykorzystującej podstawowe komórki układu FPGA, czyli komórki typu Slice). Istnieje również możliwość zaimplementowania centralnej pamięci wiadomości λ – wówczas uzyskuje się kilkuprocentowe zmniejszenie powierzchni modułów SISO, kosztem zastosowania kilku modułów pamięci BRAM współdzielonych przez wszystkie jednostki SISO,

z czego można skorzystać w przypadku realizacji dekodera w układzie zawierającym niewykorzystane bloki BRAM. Wyniki zamieszczone w tabeli 4.5 pokazują, że zastosowanie algorytmu Corrected-Min-Sum prowadzi do zwiększenia o 18% powierzchni bloku SISO (co pociąga za sobą ok. 10% wzrost powierzchni całego dekodera). Jest to koszt, który w pewnych wypadkach może być uzasadniony, biorąc pod uwagę nieco lepsze możliwe własności korekcyjne.

Tab. 4.5: Zasoby (liczba komórek typu Slice) modułu SISO dla różnych algorytmów wyznaczania wiadomości, $B = 6$, $d_{cmax} = 6$

Min-Sum	Norm.-Min-Sum $1/A = 0.75$	Off.-Min-Sum $C = 0.5$	Cor.-Min-Sum	LLR-BP
84	89	89	105	128

W tabelach 4.6-4.7 zamieszczono wyniki syntezy i parametry dekoderek wykorzystujących różne algorytmy oraz długości słowa wiadomości.

Tab. 4.6: Wyniki implementacji dekodera kodu regularnego (2640,1320), $P = 24$, różne algorytmy wyznaczania wiadomości

Algorytm	f_{clk}	TH	Slices	BRAMs
Min-Sum, $B = 6$, $Z_{th} = 14$	142MHz	55Mb/s	4127	13
Offset-Min-Sum, $C = 0.5$, $B = 6$, $Z_{th} = 14$	142MHz	55Mb/s	4247	13
Normalized-Min-Sum, $1/A = 0.75$, $B = 6$, $Z_{th} = 14$	142MHz	55Mb/s	4247	13
Corrected-Min-Sum, $B = 6$, $Z_{th} = 14$	95MHz	36,8Mb/s	4631	13
Pełny LLR-BP, $B = 6$, $Z_{th} = 14$	75MHz	29Mb/s	5183	13

Tab. 4.7: Wyniki implementacji dekodera kodu regularnego (2640,1320), $P = 24$, różne długości słowa wiadomości B

Algorytm	f_{clk}	TH	Slices	BRAMs
Corrected-Min-Sum, $B = 5$, $Z_{th} = 11$	128MHz	49,6Mb/s	4081	13
Corrected-Min-Sum, $B = 6$, $Z_{th} = 14$	95MHz	36,8Mb/s	4631	13
Corrected-Min-Sum, $B = 7$, $Z_{th} = 22$	94MHz	36,4Mb/s	5529	13
Corrected-Min-Sum, $B = 8$, $Z_{th} = 32$	86MHz	33,3Mb/s	6286	13

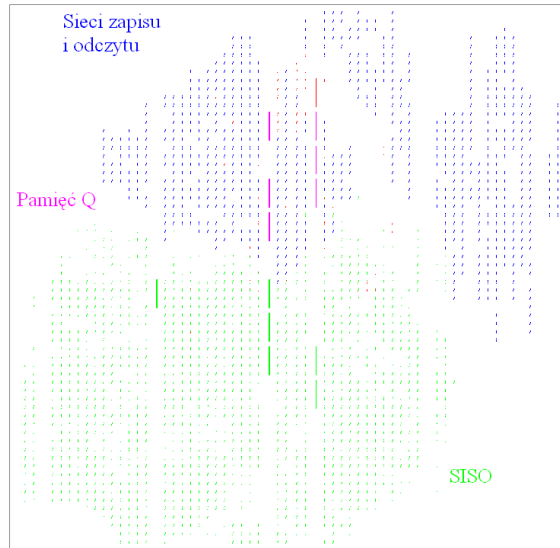
Analiza przedstawionych rozwiązań pozwala sformułować następujące wnioski:

- Różnica pomiędzy wymaganymi zasobami dekodera z algorytmem Min-Sum oraz Offset-Min-Sum (lub Normalized-Min-Sum) jest na tyle niewielka (tab. 4.5, 4.6), że sens praktycznego wykorzystania algorytmu Min-Sum jest dosyć wątpliwy (w kontekście omówionych w poprzednim podrozdziale własności korekcyjnych, rys. 4.31 i 4.32).

- Bardziej znacząca jest różnica pomiędzy zasobami oraz (w szczególności) przepustowością dekodерów z algorytmami Normalized-Min-Sum oraz Corrected-Min-Sum (tab. 4.6). W kontekście rozważań przedstawionych w poprzednim podrozdziale, algorytmy Normalized-Min-Sum bądź Corrected-Min-Sum wydają się być najlepszym wyborem w większości przypadków, przy czym ten drugi zapewnia nieco lepsze własności korekcyjne przy nieco większych zasobach struktury FPGA i mniejszej przepustowości. Mniejsza przepustowość wynika z dłuższej ścieżki danych w bloku CMin-akumulator, w której ze względu na istnienie sprzężenia zwrotnego umieszczanie rejestrów potokowych jest bezcelowe.
- Dostyc znacząca jest różnica pomiędzy wymaganymi zasobami dekodерów o różnych długościach słowa (tab. 4.7). Z tego względu typowym wyborem powinna być wartość $B = 5$ lub też $B = 6$, w których to przypadkach dekodер gwarantuje zbliżone własności korekcyjne do dekodera operującego na wiadomościach zmiennoprzecinkowych (rys. 4.29).

Na rys. 4.35 pokazano zasoby dekodera kodu regularnego (2640,1320) o $P = 24$ jednostkach SISO, zaimplementowanego w układzie XC2V3000. Oznaczono zasoby wykorzystywane przez główne bloki: SISO (zielony), konfigurowalne sieci zapisu i odczytu (niebieski), pamięć Q (fioletowy) oraz układy sterowania (czerwony). W tabeli 4.8 zaprezentowano wyniki syntezy pokazujące jaki udział w zasobach strukturalnych dekodera mają poszczególne bloki. Jak widać, jednostki obliczeniowe SISO zajmują nieco ponad połowę powierzchni dekodera. Pozostała część to przede wszystkim konfigurowalne sieci zapisu i odczytu. Układy sterowania i generatory adresów stanowią tylko 4% powierzchni. Jednakże zaprojektowanie tych układów stanowiło duży problem w procesie specyfikacji dekodera. Staranny opis, wymagający sporego wysiłku, znacząco wpływa na parametry dynamiczne uzyskanego rozwiązania.

Podsumowując, biorąc pod uwagę wyniki zaprezentowane w poprzednim oraz niniejszym podrozdziale, najbardziej korzystnym zestawieniem własności korekcyjnych oraz strukturalnych cechują się dekodery oparte na algorytmach Corrected-Min-Sum oraz Normalized-Min-Sum, przy długości słowa wiadomości $B = 5 \dots 6$. Tym niemniej, użytkownik opracowanego środowiska projektowego ma do wyboru pełną gamę algorytmów. Wybór może zostać dokonany w oparciu o zaprezentowane wyniki oraz ukierunkowane na konkretne zastosowanie eksperymenty, które mogą zostać szybko wykonane za pomocą opracowanego środowiska projektowo-symulacyjnego.



Rys. 4.35: Dekoder kodu (2640,1320) o $P = 24$ jednostkach SISO, w układzie XC2V3000

Tab. 4.8: Elementy dekodera kodu regularnego (2640,1320), $P = 24$, algorytm Corrected-Min-Sum, $B = 6$, $Z_{th} = 14$

Element	Slices	Slices [%]	BRAMs
24 moduły SISO	2520	54.5	0
Konfigurowalne sieci zapisu i odczytu	1894	40.9	0
Pamięć permutacji	0	0	8
Pamięć wiadomości Q	0	0	4
Pamięć struktury \mathbf{W}	0	0	1
Jednostka próbných decyzji	30	0.6	0
Układy sterowania i generatory adresów	187	4	0
Suma	4631	100	13

4.10 Podsumowanie

W rozdziale przedstawiono szczegółowo strukturę i własności opracowanego uniwersalnego dekodera sprzętowego kodów AA-LDPC. Dużą wartością zaprojektowanego dekodera jest możliwość szybkiego (automatycznego) dostosowania go do dowolnego kodu AA-LDPC. Użytkownik może również dokonać wyboru algorytmu wyznaczania wiadomości oraz długości słowa. Pomocne w tym są przedstawione wykresy bitowej stopy błędów oraz wyniki implementacji (zasoby i przepustowość dekodera) dla kilku przykładowych kodów. Wyniki odnoszące się do innych kodów można uzyskać przy wykorzystaniu opracowanego zintegrowanego środowiska symu-

lacji systemów kodowania LDPC.

Opracowanie uniwersalnego dekodera było zadaniem nietrywialnym. Duży wysiłek wiązał się z zaprojektowaniem układów sterowania (generacji sygnałów kontrolnych), których prezentacja w niniejszej pracy została pominięta. Opis w języku VHDL modułu dekodera składa się typowo z około 2000 linii, z których fragmenty definiujące kod AA-LDPC są generowane automatycznie za pomocą skryptów środowiska MATLAB. Zaprojektowany dekodery jest konkurencyjny ze względu na przepustowość oraz wymagane zasoby strukturalne do rozwiązań prezentowanych w literaturze.

Cennym i oryginalnym elementem jest zaprezentowana teoretyczna analiza błędów obciążenia wiadomości stałoprzecinkowych, stanowiąca podstawę zaproponowanych modyfikacji układowych prowadzących do zmniejszenia wpływu owych błędów. Uzyskano znaczącą poprawę bitowej stopy błędów, szczególnie dla dużych wartości SNR.

Dokonano również analizy wpływu struktury macierzy kontrolnej na „płynność” przetwarzania potokowego w dekodery oraz zaproponowano sposób optymalizacji macierzy prowadzący do lepszego wykorzystania zalet przetwarzania potokowego. Uzyskano stosunkowo wysoką częstotliwość pracy, a zaproponowany algorytm optymalizacji macierzy umożliwił minimalizację czasów bezczynności. W efekcie uzyskana przepustowość dekodera, przy stosunkowo niewielkich zasobach sprzętowych, jest znakomita.

5. Algorytmy tworzenia kodów AA-LDPC

Bardzo istotny, pozwalający na osiągnięcie dobrych własności korekcyjnych kodu, jest odpowiedni sposób tworzenia macierzy kontrolnej kodu AA-LDPC. Zdarza się, że w pewnych przypadkach (dla bardzo dużych długości bloku N) dobre rezultaty można osiągnąć przy zastosowaniu losowo wygenerowanej macierzy, nie jest to jednak regułą. Dla zapewnienia znakomitych właściwości kodów AA-LDPC o dowolnych parametrach, konieczne jest opracowanie odpowiednich algorytmów tworzenia macierzy kontrolnych.

W niniejszym rozdziale przedstawiono zatem metody tworzenia macierzy kontrolnych „dobrych” kodów AA-LDPC, zapewniających konkurencyjny do znanych kodów LDPC poziom bitowej stopy błędów systemu transmisji. Dokładano starań, aby opracowane algorytmy były uniwersalne, tzn. umożliwiały tworzenie kodów regularnych, jak i nieregularnych, o dowolnej długości bloku N , stopie kodu R oraz rozmiarze podmacierzy P . Utworzony kod może zostać natychmiast przetestowany za pomocą środowiska symulacji, z wykorzystaniem opracowanego dekodera implementowanego w układzie FPGA. Jeśli bitowa stopa błędów systemu nie spełnia wymagań użytkownika, to konieczna staje się zmiana długości bloku lub stopy kodu i powtórzenie procesu generacji macierzy.

Proces tworzenia macierzy kontrolnej kodu AA-LDPC jest wieloetapowy, a podstawowymi trzema etapami są:

- 1. Wyznaczenie dystrybucji stopni wierzchołków grafu Tannera.**
- 2. Utworzenie macierzy bazowej (grafu bazowego) kodu.**
- 3. Ekspansja macierzy bazowej, czyli wyznaczenie poszczególnych podmacierzy permutacji.**

W pierwszych dwóch etapach wykorzystano (z pewnymi modyfikacjami) znane metody tworzenia kodów LDPC nie zorientowanych na implementację. Analiza

literatury oraz wyniki przeprowadzonych eksperymentów pozwoliły na zaadaptowanie odpowiednio wybranych algorytmów. Oryginalne elementy znalazły się przede wszystkim w etapie 3 (ekspansji macierzy bazowej), który jest kluczowy w procesie tworzenia „dobrego” kodu AA-LDPC. Opracowane algorytmy pozwalają na uzyskanie kodów o zbliżonych, a nawet lepszych własnościach korekcyjnych od znanych ze źródeł kodów LDPC, a jednocześnie dekodery mogą być efektywnie implementowane w strukturze programowalnej.

Na początku procesu tworzenia kodu należy określić, czy pożądany kod powinien być regularny, czy nieregularny. W tym drugim przypadku konieczne jest określenie dystrybucji stopni wierzchołków grafu Tannera. Dla pewnych szczególnych wartości stopy kodu R , „dobre” dystrybucje można uzyskać z wyników publikowanych w postaci tabelarycznej, np. w [81, 113]. Aby opracowane narzędzie programowe było kompletne i realizowało swoje funkcje dla kodów o dowolnej stopie, zaimplementowano w nim algorytm wyznaczania dystrybucji wykorzystujący techniki optymalizacyjne (podobnie jak w [81]). Zaproponowano jednak poszukiwanie rozwiązania wśród dyskretnego zbioru możliwych dystrybucji, gdyż macierz bazowa kodu AA-LDPC jest stosunkowo niewielka, więc zbiór możliwych dystrybucji nie jest bardzo liczny.

Kolejne etapy to utworzenie grafu bazowego (macierzy bazowej) o wyznaczonej dystrybucji stopni wierzchołków, a następnie grafu kodu AA-LDPC, poprzez tzw. ekspansję grafu bazowego. Wykorzystywane algorytmy zostały opracowane w taki sposób, aby uzyskać graf kodu o jak najlepszych własnościach. Bitowa stopa błędów systemu dla dużych wartości SNR przy zastosowaniu dekodowania ML zależy od odległości minimalnej d_{\min} kodu oraz dystrybucji odległości Hamminga słów kodowych [44], które to parametry zależą od struktury grafu Tannera. Przy zastosowaniu iteracyjnych algorytmów dekodowania, dodatkowo na jakość systemu ma wpływ obecność pewnych szkodliwych podgrafów, tzn. krótkich cykli oraz tzw. zbiorów \mathcal{TS} (*Trapping Sets*). Tak więc optymalizacja struktury grafu ze względu na wymienione parametry jest podstawą zaproponowanych algorytmów.

Odległość minimalna kodu AA-LDPC jest nie mniejsza od odległości minimalnej kodu skojarzonego z macierzą bazową. Źródłem cykli oraz zbiorów \mathcal{TS} w grafie Tannera kodu są odpowiadające im cykle oraz zbiory \mathcal{TS} w grafie bazowym. Stąd też graf kodu AA-LDPC ma własności nie gorsze niż graf bazowy kodu. Pożądane jest zatem zapewnienie „dobrych” własności grafu bazowego, co można osiągnąć jedną ze znanych metod tworzenia macierzy kontrolnych kodów LDPC nie zorientowanych na implementację. Do tego celu wybrano algorytm, który jest uniwersalny a jednocześnie zapewnia dobre własności grafu kodu [37]. Uzupełniono go o metodę

dostosowania macierzy do możliwości efektywnego kodowania (algorytm 2.1) oraz optymalizację przetwarzania potokowego w dekodерze (algorytm 4.4).

Graf bazowy, pomimo zastosowania odpowiedniej metody jego tworzenia, ze względu na dosyć niewielkie rozmiary zawiera zwykle wiele krótkich cykli, a odległość minimalna kodu z nim skojarzonego jest niewielka. Stąd też odpowiedni sposób ekspansji grafu jest najistotniejszym etapem, decydującym o własnościach kodu AA-LDPC. W opracowanym algorytmie, graf bazowy jest poddawany analizie, która polega na wyszukaniu cykli oraz zbiorów \mathcal{TS} , następnie tworzona jest priorytetowa lista tych struktur, aż wreszcie za pomocą odpowiedniego algorytmu uwzględniającego priorytety, określone są takie podmacierze permutacji, dla których na etapie ekspansji grafu „usuwana” jest jak największa liczba szkodliwych struktur.

5.1 Wyznaczenie dystrybucji stopni wierzchołków grafu Tannera

Jak zostało wykazane [59, 81], kody nieregularne cechują się lepszymi własnościami korekcyjnymi, szczególnie dla niskiej wartości SNR (tzw. *waterfall region*). Przykładowo autorzy pracy [16] uzyskali kod o długości bloku $N = 10^7$ osiągający $\text{BER} = 10^{-6}$ przy SNR zaledwie o 0.0045dB większym od teoretycznej granicy Shannona. Uzyskanie kodu regularnego o takich własnościach jest niemożliwe [83]. Z drugiej strony, niekorzystne zakrzywienie charakterystyki $\text{BER} = f(\text{SNR})$ przy dużych wartościach SNR (tzw. *error floor*) dla kodów nieregularnych występuje zwykle przy większej wartości BER, co zostało po raz pierwszy pokazane w [63]. Stąd też, jeśli system ma zapewniać bardzo niską bitową stopę błędów dla wysokich wartości SNR, czasem lepszym rozwiązaniem może okazać się zastosowanie kodu regularnego. Najlepsze własności ze względu na wspomniany efekt zakrzywienia charakterystyki $\text{BER} = f(\text{SNR})$ mają kody o małych stopniach wierzchołków bitowych (najczęściej rozpatrywane są zatem kody o $d_b = 3$).

Na etapie wyznaczania dystrybucji stopni wierzchołków, rozpatrywane są klasy kodów $\mathcal{C}^{\text{db}, \text{dc}}(N, K)$, spośród których wybierana jest klasa o najlepszych oczekiwanych własnościach. W celu określenia własności klasy kodów wykorzystuje się tzw. ewolucję gęstości prawdopodobieństwa (*Density Evolution*, [83]), która polega na obliczaniu rozkładów prawdopodobieństwa wartości wiadomości dla kolejnych iteracji procesu dekodowania słowa kodowego składającego się z samych zer. Na podstawie rozkładu można wówczas obliczyć prawdopodobieństwo błędnego określenia wartości bitu w danej iteracji, które charakteryzuje daną klasę kodów.

5.1.1 Ewolucja gęstości prawdopodobieństwa

Dla rzeczywistego dekodera, wiadomości przyjmują wartości ze skończonego zbioru ($\mathcal{O}_{B,Z_{th}}$ zdefiniowane w (4.7)), tak więc rozkłady prawdopodobieństw są dyskretne:

$$p_Q[k] = P(Q_{nm} = k\Delta), \quad k \in \left\{ -\frac{Z_{th}}{\Delta}, \dots, \frac{Z_{th}}{\Delta} \right\} \quad (5.1a)$$

$$p_R[k] = P(R_{mn} = k\Delta), \quad k \in \left\{ -\frac{Z_{th}}{\Delta}, \dots, \frac{Z_{th}}{\Delta} \right\} \quad (5.1b)$$

gdzie Q_{nm} to wiadomość od węzła bitowego do kontrolnego, a R_{mn} – wiadomość od węzła kontrolnego do bitowego. Wektor \mathbf{p}_Q składający się z wartości $p_Q[k]$ określa rozkład prawdopodobieństwa, że wiadomość od węzła bitowego do kontrolnego przyjmuje wartość $k\Delta$, natomiast wektor \mathbf{p}_R – rozkład prawdopodobieństwa, że wiadomość od węzła kontrolnego do bitowego przyjmuje wartość $k\Delta$.

Procedurę ewolucji gęstości prawdopodobieństw dla dyskretnych wiadomości, opisaną dokładniej np. w pracach [16, 85], można zapisać za pomocą zależności, które zostaną przytoczone poniżej. Niech \mathbf{p}_{in} będzie rozkładem prawdopodobieństwa wartości wiadomości wejściowych dla algorytmu LLR-BP, przy założeniu słowa kodowego składającego się z samych zer. Ponieważ wiadomości wyznaczane w węzłach bitowych równe są sumie wiadomości wejściowych, to rozkład prawdopodobieństwa $\mathbf{p}_Q^{(i)}$ (w i -tej iteracji) można wyznaczyć jako dyskretny spłot rozkładów wiadomości wejściowych (przy założeniu, że wiadomości wejściowe są statystycznie niezależne). Dla kodu regularnego o wierzchołkach bitowych stopnia d_b :

$$\mathbf{p}_Q^{(i)} = \mathbf{p}_{in} * \left(\bigotimes_{d_b-1} \mathbf{p}_R^{(i)} \right) \quad (5.2)$$

Natomiast wiadomości w wierzchołkach kontrolnych wyznaczane są poprzez rekurencyjne obliczanie wyniku dwuargumentowej operacji \boxplus (jak w (2.23)), tak więc rozkłady prawdopodobieństw wartości tych wiadomości (dla kodów regularnych o stopniach węzłów d_c) można wyznaczyć następująco:

$$\mathbf{p}_R = \Upsilon(\mathbf{p}_Q, \Upsilon(\mathbf{p}_Q \dots, \Upsilon(\mathbf{p}_Q, \mathbf{p}_Q) \dots)) = \Upsilon^{d_c-1}(\mathbf{p}_Q) \quad (5.3a)$$

$$\text{gdzie } \Upsilon(\mathbf{p}_1, \mathbf{p}_2)[k] = \sum_{(i,j): k\Delta=(i\Delta)\boxplus(j\Delta)} p_1[i]p_2[j] \quad (5.3b)$$

W przypadku klasy kodów nieregularnych, należy uśrednić (z odpowiednimi wagami) rozkłady prawdopodobieństw otrzymane dla wierzchołków o poszczególnych stopniach. Niech $\boldsymbol{\kappa} = [\kappa_2, \kappa_3, \dots, \kappa_{d_{bmax}}]$ będzie względną dystrybucją krawędzi incydentnych do wierzchołków bitowych, gdzie κ_t to stosunek liczby krawędzi incydentnych do węzłów bitowych stopnia t do liczby wszystkich krawędzi. Podobnie

niech $\boldsymbol{\chi} = [\chi_2, \chi_3, \dots, \chi_{d_{c_{max}}}]$ określa względną dystrybucję krawędzi incydujących do wierzchołków kontrolnych. Ewolucję gęstości prawdopodobieństw dla kodu nieregularnego o względnych dystrybucjach $(\boldsymbol{\kappa}, \boldsymbol{\chi})$ można wyrazić następująco:

$$\mathbf{p}_Q^{(0)} = \mathbf{p}_{in} \quad (5.4a)$$

$$\mathbf{p}_R^{(i)} = \sum_{s=2}^{d_{c_{max}}} \chi_s \Upsilon^{s-1}(\mathbf{p}_Q^{(i-1)}) \quad (5.4b)$$

$$\mathbf{p}_Q^{(i)} = \mathbf{p}_{in} * \sum_{t=2}^{d_{b_{max}}} \kappa_t \left(\bigotimes_{t=2}^{t-1} \mathbf{p}_R^{(i)} \right), \quad i = 1, \dots, i_{max} \quad (5.4c)$$

Wiadomość z wierzchołka bitowego wskazuje na błędną wartość bitu wtedy, gdy jest mniejsza lub równa zero, gdyż dla słowa kodowego składającego się z samych zer, wiadomości w postaci LLR wskazujące poprawne wartości bitów są dodatnie. Prawdopodobieństwo błędnej decyzji w iteracji i -tej można więc wyznaczyć zgodnie z zależnością (5.5).

$$p_{err}^{(i)} = \sum_{k \leq 0} p_Q^{(i)}[k] \quad (5.5)$$

5.1.2 Optymalizacja dystrybucji stopni wierzchołków

Optymalizacja dystrybucji stopni wierzchołków polega na wyznaczeniu klasy kodów o względnych dystrybucjach $(\boldsymbol{\kappa}, \boldsymbol{\chi})$, dla której tzw. wartość progowa (ang. *threshold*, [17, 81]) pewnego parametru charakteryzującego kanał ma najkorzystniejszą wartość. Dla kanału AWGN wartość progowa σ_{th}^2 definiowana jest jako maksymalna wartość wariancji σ^2 procesu losowego skojarzonego z szumem, dla której przy liczbie iteracji i dążącej do nieskończoności, prawdopodobieństwo błędu (5.5) dąży do zera.

W celu wyznaczenia wartości progowej σ_{th}^2 należy wielokrotnie przeprowadzić procedurę ewolucji gęstości prawdopodobieństwa, przy rozkładzie prawdopodobieństwa wartości wiadomości wejściowych \mathbf{p}_{in} wyznaczonym dla kanałów o różnych wartościach σ^2 , a następnie określić największą wartość σ^2 , dla której prawdopodobieństwo błędu p_{err} dąży do zera. W praktyce przyjmuje się, że warunek ten jest spełniony, jeśli dla bardzo dużej liczby iteracji, prawdopodobieństwo p_{err} jest bardzo małe, np. $p_{err}^{(i)} < 10^{-6}$ dla $i = 500$.

Dla kodów AA-LDPC, względna dystrybucja stopni wierzchołków grafu Tannera kodu równa jest względnej dystrybucji stopni grafu bazowego kodu. W typowym przypadku macierz bazowa ma stosunkowo niewielkie rozmiary (rzędu kilkudziesięciu wierszy i kolumn), tak więc liczba możliwych dystrybucji nie jest bardzo wielka.

Zaproponowano zatem metodę optymalizacji poprzez przeszukanie wszystkich możliwych dystrybucji w celu wyboru tej, dla której klasa kodów ma największą wartość progową σ_{th}^2 . Algorytm optymalizacji przedstawiony jest w ramce algorytm 5.1.

Dla „dobrych” dystrybucji, stopnie wierzchołków kontrolnych przyjmują prawie zawsze jedynie dwie kolejne wartości całkowite (np. $d_c = 6$, $d_c = 7$) [8, 81]. Stąd też w algorytmie 5.1 rozpatrywane są różne dystrybucje stopni wierzchołków bitowych, natomiast dystrybucja wierzchołków kontrolnych jest niezerowa tylko dla dwóch elementów χ . Wyznaczana jest ona na podstawie wartości κ w taki sposób, aby liczby krawędzi incydujących do wszystkich węzłów bitowych i kontrolnych były sobie równe.

Inne spostrzeżenie płynące z analizy wyników opublikowanych w [81] jest takie, że w dystrybucjach występują najczęściej tylko stopnie 2, 3, 4 oraz pewna maksymalna wartość $d_{b_{max}}$, która wynosi od 5 do kilkudziesięciu. Dodatkowo liczba wierzchołków stopnia 2 jest zawsze nie większa niż połowa wszystkich wierzchołków, a wartość średnia stopnia wierzchołka bitowego jest z przedziału od 3 do 4.

W pierwszym etapie algorytmu poszukiwane są wszystkie kombinacje wartości całkowitych $\{l_2, l_3, l_4, l_{d_{b_{max}}}\}$ (gdzie l_t to liczba wierzchołków bitowych stopnia t) spełniające warunki (5.6). Liczba wszystkich wierzchołków równa jest L , stąd też wartości l_t ograniczone są do przedziału od zera do L , za wyjątkiem l_2 , które jest ograniczane (zgodnie ze wspomnianym wyżej spostrzeżeniem) do $L/2$. Warunek (5.6c) to ograniczenie średniej wartości stopnia wierzchołka bitowego. Przykładowo dla macierzy bazowej o $L = 32$ kolumnach, przy $d_{b_{max}} = 8$ liczba rozpatrywanych kombinacji spełniających warunki (5.6) wynosi 2142, a dla $L = 64$, $d_{b_{max}} = 10$, liczba kombinacji to 11579. Czas obliczeń według algorytmu 5.1 w środowisku MATLAB na komputerze klasy PC jest wówczas rzędu kilku, kilkunastu godzin.

Dla każdej kombinacji $\{l_2, l_3, l_4, l_{d_{b_{max}}}\}$ wyznaczana jest wartość progowa σ_{th}^2 , przy założeniu kanału AWGN oraz dekodera LLR-BP. Jako wynik optymalizacji wybierana jest ta kombinacja, dla której wartość progowa jest największa. Maksymalny stopień wierzchołka bitowego $d_{b_{max}}$ jest parametrem algorytmu, przy czym typowo dla większych wartości $d_{b_{max}}$ można uzyskać większe σ_{th}^2 . Dla kodów o małych i średnich długościach bloku wartość $d_{b_{max}}$ nie może być jednak zbyt wielka ze względu na rosnące prawdopodobieństwo powstawania krótkich cykli w grafie Tannera.

Algorytm 5.1: Algorytm wyznaczania dystrybucji stopni wierzchołków grafu	
Dane wejściowe: Liczba wierzchołków bitowych L , liczba wierzchołków kontrolnych D , maksymalny stopień wierzchołka bitowego $d_{b_{max}}$	
Dane wyjściowe: Dystrybucja stopni wierzchołków bitowych \mathbf{d}_b	
1	Utworzyć wszystkie możliwe kombinacje liczb całkowitych nieujemnych $\{l_2, l_3, l_4, l_{d_{b_{max}}}\}$ spełniających następujące warunki:
	$l_2 + l_3 + l_4 + l_{d_{b_{max}}} = L$ (5.6a)
	$0 \leq l_2 \leq L/2$ (5.6b)
	$3 \leq \sum_j j l_j / L \leq 4$ (5.6c)
2	$\sigma_{best}^2 := 0$
3	DLA każdej spośród kombinacji wartości $\{l_2, l_3, l_4, l_{d_{b_{max}}}\}$
4	Wyznaczyć względną dystrybucję $\boldsymbol{\kappa} = [\kappa_2, \kappa_3, \kappa_4, \kappa_{d_{b_{max}}}]$ krawędzi incydentnych do wierzchołków bitowych:
	$\kappa_t = \frac{t l_t}{\sum_j j l_j}, \quad t = 2, 3, 4, d_{b_{max}}$ (5.7)
5	Wyznaczyć względną dystrybucję $\boldsymbol{\chi} = [\chi_{s_1}, \chi_{s_2}]$ krawędzi incydentnych do wierzchołków kontrolnych, gdzie $s_1 = \left\lfloor \sum_j j l_j / D \right\rfloor$, $s_2 = s_1 + 1$, w taki sposób, aby liczby krawędzi incydentnych do węzłów bitowych i kontrolnych były sobie równe
6	Za pomocą ewolucji gęstości prawdopodobieństwa wyznaczyć wartość progową σ_{th}^2 dla klasy kodów o dystrybucjach względnych $(\boldsymbol{\kappa}, \boldsymbol{\chi})$
7	JEŻELI $\sigma_{th}^2 > \sigma_{best}^2$
8	Zapamiętać aktualne wartości $\{l_2, l_3, l_4, l_{d_{b_{max}}}\}$ jako najlepsze.
9	$\sigma_{best}^2 := \sigma_{th}^2$
10	Dla zapamiętanych najlepszych wartości $\{l_2, l_3, l_4, l_{d_{b_{max}}}\}$ utworzyć wektor $\mathbf{d}_b = [d_{b_1}, d_{b_2}, \dots, d_{b_L}]$, w którym l_t elementów równych jest t , $t = 2, 3, 4, d_{b_{max}}$

W tabeli 5.1 przedstawiono dystrybucje uzyskane za pomocą algorytmu 5.1 dla $L = 64$, $D = 32$ przy maksymalnych stopniach wierzchołków bitowych 6 oraz 10 (kolumny z nagłówkiem odpowiednio 6 i 10). Dla porównania w tabeli umieszczono również wyniki opublikowane w [81] (kolumny opisane „6, w/g [81]” oraz „10,

w/g [81]” oraz dystrybucje uzyskane poprzez zmodyfikowanie („zaokrąglenie”) wartości z [81] w taki sposób, aby dokładnie odzwierciedlały dystrybucje możliwe do uzyskania przy liczbie wierzchołków $L = 64$ (kolumny „6, w/g [81] zaokrąglone” oraz „10, w/g [81] zaokrąglone”). W tabeli przedstawiono również wartości progowe σ_{th} , uzyskane przy zastosowaniu ewolucji gęstości prawdopodobieństwa dla algorytmu o wiadomościach stałoprzecinkowych przy $B = 8$, $Z_{th} = 30$. Wartości progowe dla dystrybucji [81] przepisane są bezpośrednio z pracy [81].

Tab. 5.1: Dystrybucje wierzchołków bitowych uzyskane za pomocą algorytmu 5.1 dla $L = 64$, $D = 32$

$d_{b_{max}}$	6	10	6, w/g [81]	6, w/g [81] zaokrąglone	10, w/g [81]	10, w/g [81] zaokrąglone
κ_2	0.3054	0.2393	0.3324	0.3265	0.2511	0.2489
κ_3	0.2808	0.3333	0.2463	0.2449	0.3094	0.3219
κ_4	0	0	0.1101	0.1224	0.001	0
κ_6	0.4138	0	0.3111	0.3061	0	0
κ_{10}		0.4274			0.4385	0.4292
σ_{th}	0.9249	0.9452	0.9304	0.9213	0.9558	0.9438

Można zauważyć, że wartości progowe dla wyznaczonych dystrybucji są bardzo zbliżone do wartości uzyskanych w [81] – są nieco większe niż dla „zaokrąglonych” wartości dystrybucji, natomiast nieco mniejsze od wyników pochodzących wprost z [81]. Wynika to z faktu, że autorzy pracy [81] prowadzili obliczenia na ciągłych funkcjach rozkładów gęstości prawdopodobieństwa, natomiast w niniejszej pracy zastosowano rozkłady dyskretne, odpowiadające implementowanemu sprzętowo algorytmowi dekodowania dla liczb stałoprzecinkowych, dla którego wyznaczona wartość progowa jest nieco mniejsza. Jak wykazały eksperymenty, kody uzyskane z wykorzystaniem wyznaczonych dystrybucji oraz dystrybucji pochodzących z [81] mają bardzo zbliżone własności. Dowodzi to skuteczności opracowanego algorytmu.

Spełniony został zatem podstawowy cel: opracowano narzędzie programowe umożliwiające wyznaczanie dystrybucji stopni wierzchołków grafu, które może być zastosowane dla kodów AA-LDPC o dowolnych stopach. Tym samym można uznać, że prace dotyczące pierwszego etapu tworzenia „dobrego” kodu AA-LDPC zakończyły się sukcesem.

5.2 Definicje pojęć związanych z grafem Tannera

Zanim zaprezentowane zostaną kolejne etapy generacji macierzy kontrolnych (grafów) kodów AA-LDPC, konieczne jest zdefiniowanie i wyjaśnienie znaczenia pewnych pojęć, związanych ze strukturą grafu Tannera.

Łańcuchem w grafie (ang. *walk* [24]) nazywana będzie sekwencja przyległych krawędzi oraz skojarzona z nią sekwencja sąsiadujących wierzchołków, przy czym każde dwie kolejne krawędzie w sekwencji muszą być różne. Łańcuch nazywamy **łańcuchem prostym**, jeśli dowolna krawędź występuje w łańcuchu co najwyżej jeden raz, i **łańcuchem elementarnym**, jeśli dowolny węzeł występuje w łańcuchu co najwyżej jeden raz.

Cykl to łańcuch, w którym pierwszy węzeł w sekwencji jest równy węzłowi ostatniemu. Jeśli pierwszy i ostatni węzeł stanowią w nim jedyną parę węzłów identycznych, to jest to **cykl elementarny**, natomiast jeśli łańcuch tworzący cykl jest łańcuchem prostym, to mówi się o **cyklu prostym**. Liczba krawędzi cyklu stanowi **długość cyklu**, która w grafie dwudzielnym ma zawsze wartość parzystą. Cykl o długości k nazywany będzie skrótowo k -cyklem (np. 4-cykl, 6-cykl itd.) i oznaczany jako $e_1 \sim e_2 \sim \dots \sim e_k \sim$, gdzie e_1, e_2, \dots, e_k to kolejne krawędzie cyklu.

Należy zauważyć, że przedstawiona definicja cyklu (za [53]) jest nieco odmienna od definicji używanych w większości prac odnoszących się do grafów Tannera (np. [49]), w których pojęciem „*cycle*” określany jest zazwyczaj cykl elementarny. Uzasadnieniem zastosowanej definicji jest fakt, że cykl elementarny w grafie kodu AA-LDPC odpowiada dowolnemu cyklowi (niekoniecznie elementarnemu) w grafie bazowym kodu. Tak więc na etapie analizy grafu bazowego kodu należy rozpatrywać wszystkie cykle (a nie tylko cykle elementarne).

Istotnym parametrem grafu Tannera kodu jest długość najkrótszego cyklu występującego w grafie czyli **obwód grafu** ([114], ang. *girth* [24]), oznaczany przez g . Pojęcie **obwodu grafu w węźle bitowym** b_n , g_{b_n} oznacza długość najkrótszego cyklu zawierającego węzeł bitowy b_n , a **wartość średnia obwodu** w węzłach bitowych \bar{g} dana jest wzorem:

$$\bar{g} = \frac{1}{N} \sum_{n=1}^N g_{b_n} \quad (5.8)$$

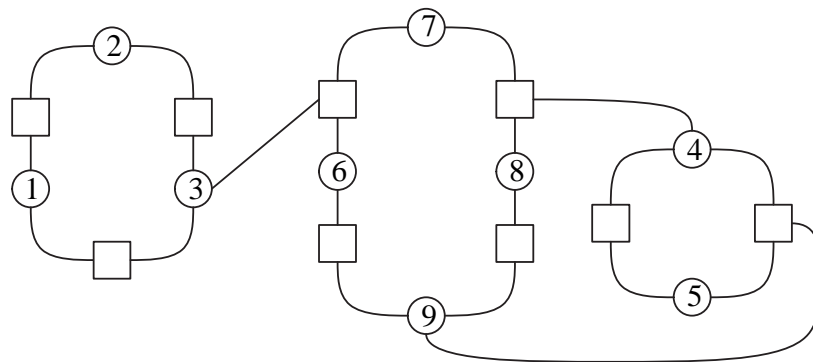
Dla danej dystrybucji stopni wierzchołków grafu, możliwy do uzyskania obwód grafu g rośnie logarytmicznie z długością bloku kodu N ([31]), a typowe wartości obwodu grafu kodów o rozmiarze bloku rzędu kilku tysięcy bitów wynoszą 6...12. Należy też zauważyć, że możliwa do uzyskania wartość obwodu grafu zależy od stopy kodu (im większa stopa, tym mniejszy osiągalny obwód). Wiele algorytmów tworze-

nia kodów opartych jest na „usuwaniu” krótkich cykli z grafu Tannera, bądź też maksymalizacji wartości średniej obwodu w węzle bitowym. Jak zostało wspomniane w rozdziale 2.2, krótkie cykle mają negatywny wpływ na dokładność dekodowania BP. Drugim, niemniej istotnym uzasadnieniem poszukiwania grafów o dużej wartości obwodu jest związek pomiędzy g a minimalną odległością kodu d_{\min} , który ma postać dolnego ograniczenia wartości minimalnej odległości kodu w zależności od obwodu grafu [38]. Tak więc maksymalizacja wartości g pośrednio przyczynia się do zwiększenia minimalnej odległości kodu d_{\min} .

Bardziej ogólny sposób tworzenia grafów „dobrych” kodów polega na minimalizacji obecności w grafie pewnych szkodliwych struktur: podgrafów indukowanych przez zbiory \mathcal{SS} (*Stopping Set*) i \mathcal{TS} (*Trapping Set*). Struktury te składają się typowo z pewnej liczby połączonych cykli, a ich obecność ma ścisły związek z minimalną odległością kodu, jak również może powodować zatrzymywanie procesu dekodowania w tzw. pułapkach (*traps*) [80]. W tym miejscu koniecznym staje się przedstawienie precyzyjnych definicji owych struktur.

Podgrafem indukowanym przez podzbiór wierzchołków bitowych nazywany będzie graf składający się z wszystkich wierzchołków tego podzbioru, sąsiadujących wierzchołków kontrolnych oraz wszystkich krawędzi łączących owe wierzchołki [24].

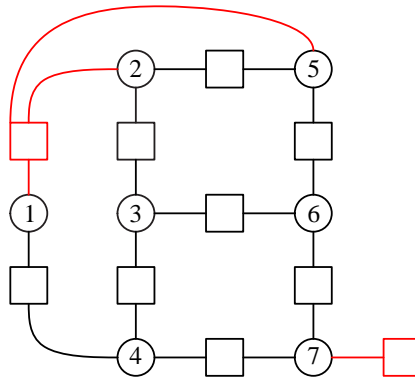
Zbiór \mathcal{SS}_a (*Stopping Set*) [22] jest to podzbiór a wierzchołków bitowych indukujących podgraf, w którym wszystkie wierzchołki kontrolne mają stopień nie mniejszy niż 2. Jest to zilustrowane przykładem na rys. 5.1, na którym wszystkie wierzchołki kontrolne (oznaczone przez „□”) są połączone z co najmniej 2 wierzchołkami bitowymi („○”), stąd zbiór wierzchołków bitowych $\{1, 2, \dots, 9\}$ stanowi \mathcal{SS}_9 .



Rys. 5.1: Podgraf indukowany przez przykładowy \mathcal{SS}_9

Zbiór $\mathcal{TS}_{a,b}$ (*Trapping Set*) [80] jest to podzbiór a wierzchołków bitowych in-

dukujących podgraf zawierający dokładnie b wierzchołków kontrolnych nieparzystego stopnia. Zbiór $\mathcal{TS}_{a,b}$ jest **elementarnym zbiorem** $\mathcal{TS}_{a,b}$ wtedy, gdy wszystkie wierzchołki kontrolne w podgrafie indukowanym mają stopień równy 1 lub 2. Przykład nieelementarnego zbioru $\mathcal{TS}_{7,2}$ przedstawiono na rys. 5.2. Kolorem czerwonym oznaczono wierzchołki kontrolne nieparzystego stopnia. Jeden z owych wierzchołków ma stopień 3, stąd przedstawiony $\mathcal{TS}_{7,2} = \{1, 2, \dots, 7\}$ nie jest zbiorem elementarnym.



Rys. 5.2: Podgraf indukowany przez $\mathcal{TS}_{7,2}$

Zbiór \mathcal{LD}_a (Linearly Dependent Set) [107] jest to podzbiór a wierzchołków bitowych, dla których skojarzone kolumny macierzy kontrolnej są liniowo zależne, ale dowolny podzbiór tych kolumn jest liniowo niezależny. Kolumny są liniowo zależne wtedy i tylko wtedy, gdy wszystkie wierzchołki kontrolne w podgrafie indukowanym mają parzysty stopień, stąd:

- $\mathcal{LD}_a = \mathcal{TS}_{a,0}$ oraz
- każdy zbiór \mathcal{LD}_a jest zbiorem \mathcal{SS}_a (ale odwrotne twierdzenie nie jest prawdziwe).

Można wykazać [107], że graf kodu o odległości minimalnej d_{\min} zawiera przynajmniej jeden zbiór $\mathcal{LD}_{d_{\min}}$, lecz nie zawiera zbioru \mathcal{LD}_a dla $a < d_{\min}$. Stąd wniosek, że w celu uzyskania kodu o dużej odległości minimalnej d_{\min} , należy utworzyć graf Tannera w taki sposób, aby nie zawierał zbiorów \mathcal{LD}_a o małej mocy równej a .

Uzasadnione jest zatem poszukiwanie grafów pozbawionych zbiorów $\mathcal{TS}_{a,0} = \mathcal{LD}_a$ o małej wartości a . Wykazano [22, 54, 80], że również obecność zbiorów $\mathcal{TS}_{a,b}$ o małych wartościach $a, b > 0$ oraz zbiorów \mathcal{SS}_a ma negatywny wpływ na jakość

systemu kodowania. Jeśli bowiem a bitów skojarzonych z wierzchołkami bitowymi należącymi do $\mathcal{TS}_{a,b}$ jest przekłamanych (tzn. w danej iteracji procesu dekodowania twarde decyzje wskazują niepoprawne wartości bitów), to wiadomości przesyłane z b wierzchołków kontrolnych wskazują na to przekłamanie, a pozostałe wskazują poprawne słowo kodowe. Jeśli liczba wierzchołków b w podgrafie jest znacznie mniejsza od liczby pozostałych wierzchołków (wskazujących poprawne słowo), to proces dekodowania może zatrzymać się w tym stanie, a twarde decyzje wskazują wówczas na tzw. słowo bliskie słowu kodowemu (*near-codeword*, [62]). Nie jest to słowo należące do kodu, jednak tylko niewielka liczba równań kontrolnych jest niespełniona, stąd proces dekodowania może zatrzymać się w „pułapce” (*trap*). Z kolei im mniejsza liczba wierzchołków bitowych a , tym większe prawdopodobieństwo wstępnego przekłamania odpowiadających im bitów. Stąd też większa jest „szkodliwość” zbiorów $\mathcal{TS}_{a,b}$ o stosunkowo małych wartościach a i b . Wspomniany efekt jest szerzej opisany np. w [14, 62, 80].

W celu wyróżnienia wśród zbiorów \mathcal{TS} tych, które mają bardziej negatywny wpływ na jakość systemu kodowania, definiowane jest pojęcie *Absorbing Set* [26].

Dla danego wierzchołka bitowego należącego do zbioru $\mathcal{TS}_{a,b}$, niech l_{odd} oznacza liczbę sąsiadujących z nim wierzchołków kontrolnych stopnia parzystego w podgrafie indukowanym, natomiast l_{even} – liczbę sąsiadujących wierzchołków kontrolnych stopnia nieparzystego w podgrafie indukowanym. Konfiguracją wierzchołka bitowego b_n w zbiorze $\mathcal{TS}_{a,b}$ będzie nazywana para $(l_{odd} : l_{even})$.

Zbiór $\mathcal{AS}_{a,b}$ (*Absorbing Set*) [26] to zbiór $\mathcal{TS}_{a,b}$, dla którego w podgrafie indukowanym każdy wierzchołek bitowy połączony jest większą liczbą krawędzi z wierzchołkami kontrolnymi stopnia parzystego niż z wierzchołkami stopnia nieparzystego. Inaczej mówiąc, konfiguracja $(l_{odd} : l_{even})$ każdego z wierzchołków bitowych należących do $\mathcal{AS}_{a,b}$ spełnia warunek $l_{even} > l_{odd}$.

W dwudzielnym grafie nie zawierającym wierzchołków bitowych stopnia 1, graf indukowany przez dowolny zbiór \mathcal{SS}_a zawiera co najmniej jeden cykl. (Dowód: graf indukowany nie jest drzewem, gdyż drzewo zawiera liście stopnia 1, [108]). W ogólnym przypadku graf indukowany przez \mathcal{SS}_a składa się z pewnej liczby połączonych cykli (rys. 5.1). Podobnie, grafy indukowane przez zbiory $\mathcal{AS}_{a,b}$ składają się z pewnej liczby połączonych cykli. Dostatecznie istotne dla dalszych rozważań jest następujące spostrzeżenie [108]: prawdopodobieństwo, że dany cykl jest fragmentem podgrafu indukowanego przez zbiór \mathcal{AS} maleje wraz ze wzrostem liczby krawędzi od węzłów bitowych cyklu do węzłów kontrolnych nie należących do cyklu (węzłów zewnętrznych). Liczba tych krawędzi wiąże się bezpośrednio z parametrem zwanym EMD.

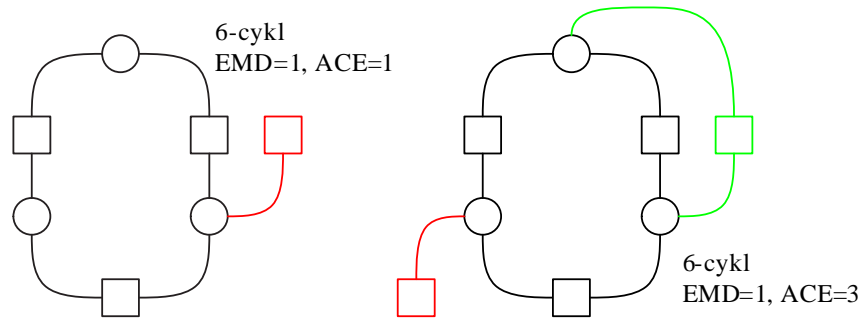
Zewnętrzny węzeł kontrolny dla podzbioru wierzchołków bitowych to węzeł,

który sąsiaduje tylko z jednym wierzchołkiem tego podzbioru.

Parametr EMD (*Extrinsic Message Degree*) podzbioru wierzchołków bitowych określa liczbę zewnętrznych węzłów kontrolnych tego podzbioru. Cykle o dużej wartości parametru EMD mają mniej negatywny wpływ na jakość systemu kodowania niż cykle o małej wartości EMD [108], gdyż z mniejszym prawdopodobieństwem stanowią fragment podgrafu indukowanego przez $\mathcal{AS}_{a,b}$ o małych wartościach a, b .

Definiowany jest również **parametr ACE** (*Approximate Cycle EMD*), który jest przybliżoną wartością EMD, równą sumie stopni wierzchołków bitowych cyklu pomniejszonych o dwa, tzn. $ACE = \sum_n (d_{b_n} - 2)$, gdzie sumowanie jest wykonywane po wszystkich wierzchołkach bitowych cyklu. Jakkolwiek w pewnych pracach, np. [107, 115], ze względu na prostotę obliczeń wykorzystuje się parametr ACE zamiast EMD, w algorytmach przedstawianych w niniejszej pracy stosowana będzie dokładna wartość EMD. Przedstawiony zostanie prosty sposób wyznaczania EMD dla dowolnego podzbioru wierzchołków bitowych.

Wartości parametrów EMD i ACE dwóch przykładowych 6-cykli pokazane są na rys. 5.3, na którym zewnętrzne węzły kontrolne oznaczono kolorem czerwonym. Cykl pokazany po prawej stronie rysunku charakteryzuje się odmiennymi wartościami: $EMD = 1$, $ACE = 3$. Ze względu na obecność węzła kontrolnego (oznaczonego na zielono), który jest połączony z dwoma węzłami bitowymi cyklu, parametr ACE w tym przypadku nie odzwierciedla faktycznej liczby wierzchołków zewnętrznych. Ze względu na tylko jeden wierzchołek zewnętrzny, obydwa cykle na rys. 5.3 mogą z dużym prawdopodobieństwem być elementem podgrafu indukowanego przez $\mathcal{AS}_{a,b}$.



Rys. 5.3: Przykładowe 6-cykle oraz wartości skojarzonych parametrów EMD i ACE

5.3 Metody generacji kodów LDPC

Przed prezentacją algorytmu tworzenia macierzy bazowej kodu, zostanie przedstawiony w niniejszym podrozdziale przegląd metod generacji kodów LDPC znanych z literatury. Stanowi on wprowadzenie do dalszej części rozdziału. Do generacji macierzy bazowej kodu AA-LDPC zaadaptowano bowiem jeden ze znanych algorytmów tworzenia grafów kodów LDPC.

Wśród znanych sposobów tworzenia macierzy kontrolnych kodów LDPC (nie zorientowanych na implementację) można wyróżnić metody deterministyczne wykorzystujące pewne konstrukcje geometryczne lub konfiguracje kombinatoryczne, metody generacji grafów Tannera polegające na sukcesywnym wstawianiu krawędzi dla uzyskania grafu o jak najmniejszej liczbie krótkich (szkodliwych) cykli, jak również metody losowe, dla których macierz kontrolna generowana jest w sposób losowy z uwzględnieniem dystrybucji stopni wierzchołków grafu.

Pierwsze z wymienionych metod umożliwiają konstruowanie macierzy kodów regularnych, o ściśle określonych długościach bloków N , przy zapewnionej pewnej wartości obwodu grafu g (np. $g = 6$). Przykładowo w [49] przedstawiono klasę kodów o macierzach kontrolnych utworzonych przy wykorzystaniu geometrii skończonej (*LDPC based on finite geometries*), gdzie wiersze macierzy odpowiadają liniom, a kolumny – punktom geometrii skończonej. Uzyskano kody regularne o $g = 6$, przy stopach kodu rzędu $R = 0,75 \dots 0,9$. Konstrukcje geometryczne zostały wykorzystane również przez autorów prac [71, 72], w których przedstawiono kody o stopniach wierzchołków bitowych (odpowiednio) 2 i 3, mniejszej stopie kodu i większym obwodzie grafu ($g = 12 \dots 16$). W [25, 110] zaprezentowano metody tworzenia kodów oparte na tzw. konfiguracjach kombinatorycznych (BIBD – *Balanced Incomplete Block Designs*, Systemy Steinera i Kirkmana), dla których obwód grafu $g = 6$. Z tego względu metody te są odpowiednie dla kodów o dużej stopie oraz o małych długościach bloku N , gdyż dla tych przypadków uzyskanie $g > 6$ jest niemożliwe żadną metodą. Na uwagę zasługują również kody opisane w [87], gdzie wykorzystano tzw. grafy Cayleya oraz Ramanujana (grafy regularne, stopa kodu $R = 0.5$), które są tzw. dobrymi ekspanderami (por. [96]), tzn. dowolny podzbiór wierzchołków bitowych jest połączony z dużą liczbą wierzchołków kontrolnych grafu.

Zasadniczym ograniczeniem wymienionych wyżej metod deterministycznych jest fakt, że są one przeznaczone w zasadzie tylko dla kodów regularnych, o ściśle określonych długościach bloku N i stopie R . Ograniczenia tego nie mają algorytmy generacji macierzy (grafów) wykorzystujące element losowości. W pracy [69] przedstawiono metodę poszukiwania „dobrych” kodów polegającą na losowym wygenerowaniu pew-

nej liczby macierzy kontrolnych (o założonej dystrybucji stopni wierzchołków grafu), spośród których wybierana jest macierz o największej wartości średniej obwodu \bar{g} skojarzonego grafu. Podobną koncepcję zaprezentowano w [106], gdzie przedstawiono metodę generacji grafów o małej liczbie krótkich cykli polegającą na wyborze spośród losowo wygenerowanych grafów tego, który ma najlepszą wartość parametru będącego ważoną sumą liczby cykli elementarnych o poszczególnych długościach. Autorzy pracy [48] proponują generację macierzy poprzez wstawianie kolejno po jednej kolumnie wybieranej z pewnej liczby wygenerowanych losowo w taki sposób, aby aktualna wartość średnia obwodu grafu \bar{g} była możliwie największa. Metoda przedstawiona w [107, 108] polega również na losowej generacji kolejnych kolumn, przy czym w kolejnych krokach wybierana jest ta kolumna, dla której w skojarzonym grafie nie występują cykle o małej wartości parametru ACE. Dostyc podobne idee zawarte są także w [79]. Z przedstawionych w tych pracach wyników eksperymentów wynika, że optymalizacja grafu kodu ze względu na cykle o małej wartości ACE (EMD) pozwala uzyskać system o mniejszej bitowej stopie błędów (szczególnie dla dużych wartości SNR) niż przy zastosowaniu optymalizacji tylko ze względu na długość cykli, czyli wartość średnią obwodu \bar{g} .

5.3.1 Algorytm PEG

Algorytmy losowej generacji macierzy dają dobre rezultaty dla kodów o bardzo dużej długości bloku [16, 60]. Dla kodów o małej i średniej wielkości bloku (rzędu kilkuset – kilku tysięcy bitów), znakomite wyniki można uzyskać przy wykorzystaniu algorytmu PEG (*Progressive Edge Growth*, [37, 38]) polegającego na konstruowaniu grafu Tannera krawędź po krawędzi w taki sposób, aby zminimalizować obecność krótkich cykli. Algorytm PEG jest cytowany w wielu pozycjach literaturowych stanowiąc jeden z najlepszych algorytmów tworzenia macierzy kontrolnych kodów LDPC i zapewniając zwykle nie gorsze wyniki niż wymienione wyżej algorytmy. Wiele spośród kodów umieszczonych w bazie MacKaya [64] zostało wygenerowanych z jego wykorzystaniem. Po analizie znanych rozwiązań algorytmów generacji kodów LDPC, zmodyfikowany algorytm PEG został wybrany do generacji macierzy bazowej (grafu bazowego) kodu AA-LDPC. Zostało to podyktowane uniwersalnością algorytmu (możliwość tworzenia kodów o dowolnych długościach bloków, stopach i dystrybucjach wierzchołków bitowych grafu), jak również znakomitymi własnościami uzyskiwanych grafów, dla małej i średniej liczby wierzchołków (graf bazowy kodu ma stosunkowo niewielką liczbę wierzchołków). Dostyc podobny do PEG algorytm został zaproponowany niezależnie w pracy [6].

W algorytmie PEG [37, 38] graf Tannera kodu konstruowany jest poprzez sukcesywne wstawianie krawędzi, przy czym kolejno umieszczane są krawędzie incydentne do poszczególnych wierzchołków bitowych grafu. Dla danego wężła bitowego wyszukiwany jest najbardziej odległy wierzchołek kontrolny i umieszczana jest krawędź łącząca te dwa wierzchołki, dzięki czemu (ewentualne) powstające cykle mają na danym etapie największą możliwą długość. Algorytm PEG przedstawiony jest formalnie jako algorytm 5.2.

Algorytm 5.2: Algorytm PEG	
Dane wejściowe:	N, M , dystrybucja stopni wierzchołków bitowych grafu \mathbf{d}_b
Dane wyjściowe:	Macierz kontrolna $\mathbf{H}_{M \times N}$
1	Utworzyć graf, $\mathcal{V}_b = \{b_1, \dots, b_N\}$, $\mathcal{V}_c = \{c_1, \dots, c_M\}$, $\mathcal{E} = \emptyset$
2	DLA $n = 1, \dots, N$ (dla każdego wierzchołka bitowego)
3	DLA $k = 1, \dots, d_{b_n}$
4	JEŻELI $k = 1$
5	$\mathcal{E} := \mathcal{E} \cup (b_n, c_m)$, gdzie c_m to wierzchołek kontrolny wybrany spośród tych o najmniejszym stopniu w aktualnej postaci grafu.
6	W PRZECIWNYM WYPADKU
7	Wyznaczyć zbiór Φ_{b_n} wierzchołków kontrolnych najbardziej odległych od b_n w aktualnej postaci grafu
8	$\mathcal{E} := \mathcal{E} \cup (b_n, c_m)$, gdzie c_m to wierzchołek kontrolny wybrany spośród tych o najmniejszym stopniu, należących do zbioru Φ_{b_n} .
9	Utworzyć macierz kontrolną $\mathbf{H}_{M \times N}$ skojarzoną z uzyskanym grafem $(\mathcal{V}, \mathcal{E})$.

Odległość między dwoma wierzchołkami w grafie to długość najkrótszego łańcucha łączącego owe wierzchołki. Zbiór wierzchołków kontrolnych najbardziej odległych od wierzchołka b_n , oznaczony przez Φ_{b_n} i wyznaczany w wierszu 7 algorytmu definiowany jest następująco:

1. W grafie niespójnym Φ_{b_n} jest podzbiorem takich wierzchołków kontrolnych, dla których nie istnieje łańcuch łączący je z wierzchołkiem b_n .
2. W grafie spójnym Φ_{b_n} jest podzbiorem wierzchołków kontrolnych o największej wartości odległości do b_n , tzn. $c_m \in \Phi_{b_n}$ wtedy, gdy nie istnieje c'_m o większej wartości odległości do b_n niż odległość między c_m a b_n .

Praktyczny sposób wyznaczania zbioru Φ_{b_n} wykorzystuje drzewo grafu tworzonego począwszy od korzenia b_n , co jest szczegółowo opisane np. w [37].

W oryginalnym algorytmie PEG [37], wybór wierzchołka spośród tych o najmniejszym stopniu w Φ_{b_n} (w wierszu 8 algorytmu 5.2) wykonywany jest w sposób losowy, bądź też wybierany jest wierzchołek o najmniejszym indeksie (czyli „pierwszy z brzegu”). W pracy [115] zaproponowano zawężenie zbioru możliwości wyboru do tych wierzchołków, do których po wstawieniu krawędzi nowo utworzone cykle mają możliwie największą wartość parametru ACE. Dzięki temu otrzymany graf zawiera mniej cykli o małej wartości ACE, a otrzymany kod ma lepsze własności korekcyjne. Podejście to zostało wykorzystane w niniejszej pracy, przy czym zamiast parametru ACE wyznaczana jest dokładna wartość parametru EMD. W literaturze proponowane są również inne sposoby zawężające zbiór Φ_{b_n} , np. [84, 91, 115]. Algorytm konstrukcji macierzy bazowej, oparty na zmodyfikowanym PEG przedstawiony będzie w kolejnym podrozdziale.

5.4 Algorytm tworzenia macierzy bazowej E-PEG

Jak już wspomniano, graf bazowy kodu AA-LDPC powinien charakteryzować się jak najlepszymi własnościami. Dlatego metodę konstrukcji macierzy bazowej, przedstawioną jako algorytm 5.3, oparto na algorytmie PEG, w którym wprowadzono pewne modyfikacje.

Wybór krawędzi wstawianych jest wykonywany w taki sposób, aby wartości parametru EMD nowo powstających cykli były jak największe. Cykle znajdowane są za pomocą algorytmu wyszukiwania cykli, który zostanie przedstawiony w dalszej części pracy. Wektor dystrybucji wierzchołków bitowych jest wstępnie sortowany, a do grafu wstawiane są najpierw krawędzie incydentne do węzłów o małych stopniach, dzięki czemu minimalizowana jest liczba cykli zawierających tylko węzły o małych stopniach (cykli o małej wartości parametru ACE). Kolejną modyfikacją jest uwzględnienie ograniczeń implementacyjnych kodera, tzn. macierz przekształcana jest do postaci prawie dolnotrójkątnej, jak opisano to w rozdziale 2.1. Oczywiście przekształcenie to można również wykonać dla macierzy kontrolnej po etapie ekspansji. Jednakże jak wykazały eksperymenty, znacznie lepszy efekt (mniejszą wartość luki determinującej złożoność obliczeniową kodera) można uzyskać wykonując owe przekształcenie dwuetapowo: najpierw dla macierzy bazowej, a następnie zamieniając jedynie wiersze w ramach podmacierzy, po etapie ekspansji. Ostatnią zaproponowaną i zweryfikowaną praktycznie modyfikacją jest końcowa optymalizacja macierzy dla celów efektywnego przetwarzania potokowego w dekodерze TDMP (algorytm 4.4 omówiony w podrozdziale 4.6.6).

Algorytm 5.3: Algorytm konstrukcji macierzy bazowej E-PEG	
Dane wejściowe: D, L , dystrybucja stopni wierzchołków bitowych grafu \mathbf{d}_b	
Dane wyjściowe: Macierz bazowa $\mathbf{W}_{D \times L}$	
1	Posortować \mathbf{d}_b , od wartości największej (d_{b_1}) do najmniejszej (d_{b_L})
2	Utworzyć graf, $\mathcal{V}_b = \{b_1, \dots, b_L\}$, $\mathcal{V}_c = \{c_1, \dots, c_D\}$, $\mathcal{E} = \emptyset$
3	DLA $n = L, \dots, 1$ (dla każdego wierzchołka bitowego)
4	DLA $k = 1, \dots, d_{b_n}$
5	JEŻELI $k = 1$
6	$\mathcal{E} := \mathcal{E} \cup (b_n, c_m)$, gdzie c_m to wierzchołek kontrolny wybrany spośród tych o najmniejszym stopniu w aktualnej postaci grafu.
7	W PRZECIWNYM WYPADKU
8	Wyznaczyć zbiór Φ_{b_n} wierzchołków kontrolnych najbardziej odległych od b_n w aktualnej postaci grafu.
9	Dla każdego z wierzchołków $c_{m'}$ o najmniejszym stopniu w zbiorze Φ_{b_n} wyznaczyć cykle zawierające krawędź $(b_n, c_{m'})$ oraz wartości EMD tych cykli.
10	$\mathcal{E} := \mathcal{E} \cup (b_n, c_m)$, gdzie (b_n, c_m) to krawędź generująca najmniejszą liczbę cykli o małej wartości parametru EMD.
11	Utworzyć macierz kontrolną $\mathbf{W}_{D \times L}$ skojarzoną z uzyskanym grafem $(\mathcal{V}, \mathcal{E})$.
12	Przekształcić macierz \mathbf{W} dla celów efektywnego kodowania, zgodnie z algorytmem 2.1.
13	Wyznaczyć macierz \mathbf{W}' dla optymalizacji przetwarzania potokowego w dekodерze, zgodnie z algorytmem 4.4.

5.5 Ekspansja macierzy bazowej kodu

Ostatni etap tworzenia macierzy kontrolnej kodu AA-LDPC to ekspansja macierzy bazowej. Zastosowanie starannie dopracowanych algorytmów na tym etapie jest bardzo istotne, gdyż wyznaczenie odpowiednich macierzy permutacji jest kluczowe dla osiągnięcia dobrych własności kodu. Zasadniczymi elementami opracowanego algorytmu ekspansji są analiza grafu bazowego w celu poszukiwania cykli i podgrafów indukowanych przez zbiory \mathcal{AS} oraz wyznaczenie takich podmacierzy permutacji, dla których jak największa liczba wspomnianych struktur jest „usuwana” z grafu kodu AA-LDPC. Odpowiednie algorytmy opracowane do realizacji tych zadań zostaną przedstawione w dalszej części podrozdziału.

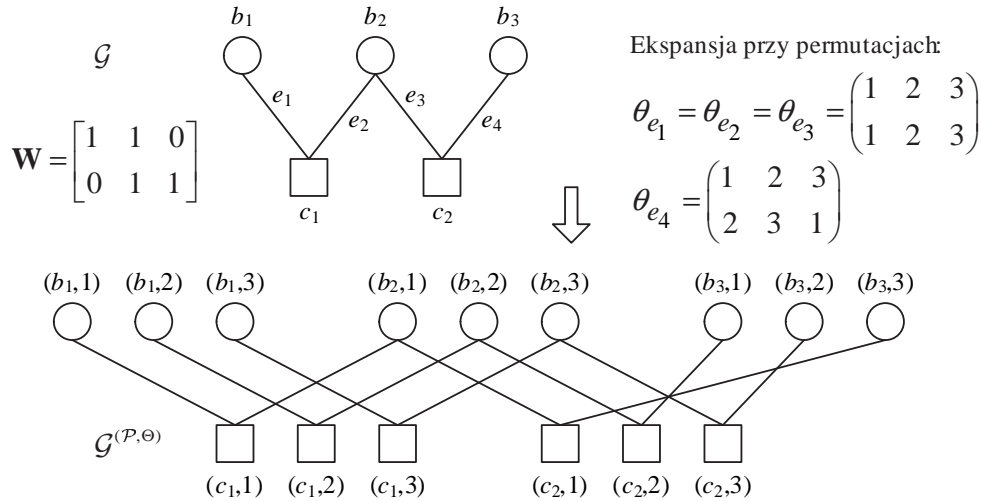
5.5.1 Definicja ekspansji grafu bazowego

Ekspansja macierzy bazowej $\mathbf{W}_{D \times L}$ polega na zastąpieniu każdego elementu $w_{d,l} = 0$ podmacierzą zerową $\mathbf{0}_{P \times P}$, a każdego elementu $w_{d,l} = 1$ pewną permutacją macierzy jednostkowej $\mathbf{I}_{\theta(p)}$, w wyniku czego otrzymuje się macierz kontrolną kodu AA-LDPC $\mathbf{H}_{DP \times LP}$ o strukturze zgodnej z (4.1).

Graf bazowy kodu (skojarzony z macierzą bazową) $\mathcal{G} = (\mathcal{V}_b \cup \mathcal{V}_c, \mathcal{E})$ składa się ze zbiorów wierzchołków $\mathcal{V}_b = \{b_1, b_2, \dots, b_L\}$, $\mathcal{V}_c = \{c_1, c_2, \dots, c_D\}$ oraz krawędzi $\mathcal{E} = \{e_1, e_2, \dots, e_{|\mathcal{E}|}\}$, gdzie $e_i = (b_l, c_d) \in \mathcal{E}$ wtedy, gdy $w_{d,l} = 1$. Ekspansji macierzy bazowej odpowiada ekspansja grafu bazowego definiowana w sposób przedstawiony poniżej.

Niech \mathcal{P} będzie zbiorem o P elementach: $\mathcal{P} = \{1, 2, \dots, p, \dots, P\}$. Z każdą krawędzią $e \in \mathcal{E}$ należy skojarzyć pewną permutację zbioru \mathcal{P} , $\theta_e(p)$. Ekspansja grafu bazowego $\mathcal{G} = (\mathcal{V}_b \cup \mathcal{V}_c, \mathcal{E})$ polega na utworzeniu grafu o zbiorach wierzchołków $\mathcal{V}_b \times \mathcal{P}$, $\mathcal{V}_c \times \mathcal{P}$ oraz krawędzi $\mathcal{E} \times \mathcal{P}$, gdzie dla $e_i = (b_l, c_d) \in \mathcal{E}$, krawędź $(e_i, p) = (b_l, c_d, p) \in \mathcal{E} \times \mathcal{P}$ jest incydentna do wierzchołków (b_l, p) oraz $(c_d, \theta_{e_i}(p))$. Konkretny sposób ekspansji grafu jest jednoznacznie definiowany przez parametr P (równy rozmiarowi podmacierzy w (4.1)) oraz zbiór permutacji $\Theta = \{\theta_e : e \in \mathcal{E}\}$. Graf otrzymany w wyniku ekspansji \mathcal{G} oznaczany będzie jako $\mathcal{G}^{(P, \Theta)}$:

$$\mathcal{G}^{(P, \Theta)} = ((\mathcal{V}_b \times \mathcal{P}) \cup (\mathcal{V}_c \times \mathcal{P}), \mathcal{E} \times \mathcal{P}) \quad (5.9)$$



Rys. 5.4: Przykładowy graf bazowy oraz graf uzyskany po ekspansji

Na rysunku 5.4 przedstawiono przykład ekspansji grafu skojarzonego z macierzą bazową $\mathbf{W} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$, dla $P = 3$, dla przypadku, gdy permutacje są następujące: $\theta_{e_1} =$

$\theta_{e_2} = \theta_{e_3} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$ (permutacje tożsamościowe) oraz $\theta_{e_4} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$. Zastosowano tu dwuwierszowy sposób zapisu permutacji zbioru [47].

5.5.2 Warunek usunięcia cyklu na etapie ekspansji grafu

Twierdzenie 5.1, którego dowód znajduje się w [76], mówi o zależności pomiędzy cyklami w grafie bazowym i grafie powstałym po ekspansji.

Twierdzenie 5.1. *Graf $\mathcal{G}^{(\mathcal{P}, \Theta)}$ zawiera k -cykl $(e_1, p_1) \sim (e_2, p_2) \sim \dots \sim (e_k, p_k) \sim$ wtedy i tylko wtedy, gdy w grafie bazowym \mathcal{G} istnieje cykl $e_1 \sim e_2 \sim \dots \sim e_k \sim$ oraz p_1 jest punktem stałym złożenia permutacji:*

$$\theta_{e_k}^{-1} \circ \theta_{e_{k-1}} \circ \dots \circ \theta_{e_4}^{-1} \circ \theta_{e_3} \circ \theta_{e_2}^{-1} \circ \theta_{e_1} \quad (5.10)$$

Z twierdzenia 5.1 wynika wprost wniosek, że dla cyklu $e_1 \sim e_2 \sim \dots \sim e_k \sim$ w grafie bazowym \mathcal{G} , nie istnieje odpowiadający mu k -cykl w grafie $\mathcal{G}^{(\mathcal{P}, \Theta)}$ wtedy i tylko wtedy, gdy złożenie permutacji (5.10) nie zawiera punktu stałego. W przeciwnym razie, graf zawiera pewną liczbę k -cykli równą liczbie punktów stałych złożenia permutacji.

Niech wszystkie permutacje w zbiorze Θ są translacjami, tzn.:

$$\theta_e(p) \equiv p + s_e \pmod{P} \quad (5.11)$$

gdzie $s_e \in \{0, 1, 2, \dots, P-1\}$ jest wartością przesunięcia cyklicznego skojarzonego z krawędzią $e \in \mathcal{E}$. Złożenie translacji $\theta_{e_2} \circ \theta_{e_1}$ jest translacją o wartość $s_{e_1} + s_{e_2}$, a zbiór wszystkich translacji stanowi grupę abelową ze względu na złożenie permutacji. Grupa ta jest podgrupą grupy wszystkich permutacji (zwanej grupą symetryczną, [47]), a jedyną translacją zawierającą punkty stałe jest permutacja tożsamościowa, będąca elementem jednostkowym grupy. Dla permutacji tożsamościowej wartość przesunięcia cyklicznego wynosi $s_e = 0$.

Na podstawie twierdzenia 5.1 można wyciągnąć następujący wniosek. Jeśli wszystkie permutacje θ_e są translacjami o wartość s_e , to dla cyklu $e_1 \sim e_2 \sim \dots \sim e_k \sim$ w grafie bazowym \mathcal{G} , nie istnieje odpowiadający mu k -cykl w grafie $\mathcal{G}^{(\mathcal{P}, \Theta)}$ wtedy i tylko wtedy, gdy wartości przesunięć cyklicznych spełniają warunek:

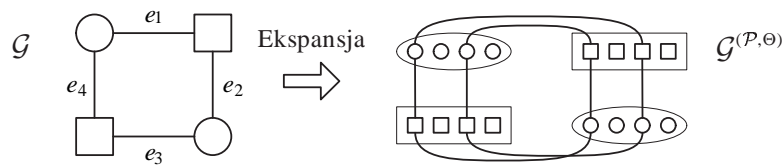
$$s_{e_1} - s_{e_2} + s_{e_3} - s_{e_4} + \dots + s_{e_{k-1}} - s_{e_k} \not\equiv 0 \pmod{P} \quad (5.12)$$

W przeciwnym razie, tzn. jeśli $s_{e_1} - s_{e_2} + \dots + s_{e_{k-1}} - s_{e_k} \equiv 0 \pmod{P}$, graf $\mathcal{G}^{(\mathcal{P}, \Theta)}$ zawiera P k -cykli odpowiadających cyklowi w grafie bazowym, gdyż permutacja tożsamościowa zbioru \mathcal{P} zawiera P punktów stałych.

W opracowanych algorytmach tworzenia macierzy kontrolnych kodów AA-LDPC, wykorzystywany zbiór permutacji ograniczono do grupy translacji, co jest uzasadnione następującymi spostrzeżeniami:

- Jediną permutacją w grupie translacji zawierającą punkty stałe jest permutacja tożsamościowa. Istnieje wiele podgrup grupy symetrycznej spełniających ten warunek, jednakże można wykazać, że wszystkie te podgrupy zawierają co najwyżej P elementów, a wśród nich podgrupy P -elementowe są izomorficzne do grupy translacji.
- Jeśli tylko złożenie permutacji skojarzonych z krawędziami cyklu grafu bazowego \mathcal{G} nie jest permutacją tożsamościową, to w grafie $\mathcal{G}^{(P,\Theta)}$ nie istnieje odpowiadający mu cykl.
- Prosty warunek (5.12) na brak cyklu w grafie kodu umożliwia zaproponowanie „praktycznych” algorytmów wyznaczania permutacji, poprzez optymalizację wartości przesunięć cyklicznych s_e .
- Pamięć permutacji w implementacji sprzętowej dekodera (rys. 4.5) może zostać zastąpiona odpowiednio inicjalizowanymi licznikami modulo- P .

Należy podkreślić, że w twierdzeniu 5.1 mowa jest o dowolnym cyklu $e_1 \sim e_2 \sim \dots \sim e_k \sim$ (niekoniecznie elementarnym) w grafie bazowym. Przykładowo na rysunku 5.5 pokazano 8-cykl (elementarny) w grafie $\mathcal{G}^{(P,\Theta)}$, któremu odpowiada w grafie bazowym następujący cykl: $e_1 \sim e_2 \sim e_3 \sim e_4 \sim e_1 \sim e_2 \sim e_3 \sim e_4 \sim$, który jest cyklem nieelementarnym. Z tego też względu, na etapie analizy grafu bazowego należy rozpatrywać nie tylko elementarne (nawet – jak w tym przypadku – 8-cykle składające się z dwukrotnie powtórzonej ścieżki tworzącej 4-cykl).



Rys. 5.5: Przykładowy 4-cykl, który po ekspansji tworzy 8-cykl

5.5.3 Warunek usunięcia zbioru \mathcal{AS} na etapie ekspansji grafu

Jeśli w grafie bazowym \mathcal{G} istnieje zbiór $\mathcal{AS}_{(a,b)}$, to w grafie $\mathcal{G}^{(P,\Theta)}$ otrzymanym po ekspansji może pojawić się P (lub mniej) odpowiadających mu $\mathcal{AS}_{(a,b)}$ o identycznej

strukturze podgrafu indukowanego. Warunek na zaistnienie takiej sytuacji został sformułowany następująco:

Twierdzenie 5.2. *Jeśli wszystkie permutacje w zbiorze Θ są translacjami, to graf $\mathcal{G}^{(\mathcal{P},\Theta)}$ zawiera zbiór $\mathcal{AS}_{(a,b)} = \{(b_1, p_1), (b_2, p_2), \dots, (b_a, p_a)\}$ odpowiadający zbiorowi w grafie bazowym $\mathcal{AS}_{(a,b)} = \{b_1, b_2, \dots, b_a\}$ wtedy i tylko wtedy, gdy dla wszystkich cykli w podgrafie indukowanym przez \mathcal{AS} złożenie permutacji (5.10) jest permutacją tożsamościową.*

Dowód:

1. Jeśli dla jakiegokolwiek cyklu w podgrafie grafu \mathcal{G} złożenie permutacji (5.10) nie ma punktu stałego, to zgodnie z twierdzeniem 5.1 w grafie $\mathcal{G}^{(\mathcal{P},\Theta)}$ nie istnieje skojarzony z nim cykl. Nie może wówczas istnieć podgraf grafu $\mathcal{G}^{(\mathcal{P},\Theta)}$ o identycznej strukturze jak w \mathcal{G} . Jediną translacją zawierającą punkt stały jest permutacja tożsamościowa.
2. Jeśli złożenie translacji dla wszystkich cykli podgrafu indukowanego przez \mathcal{AS} jest permutacją tożsamościową, to w grafie $\mathcal{G}^{(\mathcal{P},\Theta)}$ istnieją odpowiadające im cykle, łączące się w P podgrafów o identycznej strukturze jak podgraf indukowany przez \mathcal{AS} w \mathcal{G} .

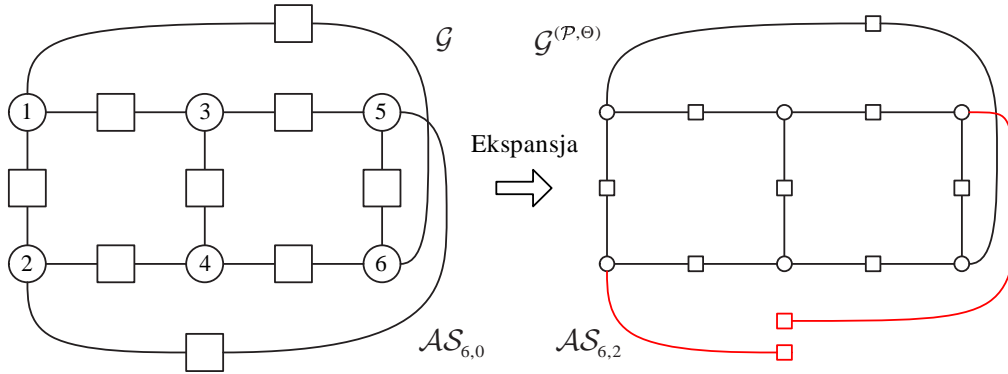
□

Dla każdego spośród zidentyfikowanych zbiorów \mathcal{AS} należy zatem zapewnić, aby na etapie ekspansji spełniony został warunek usunięcia przynajmniej jednego cyklu tworzonego przez \mathcal{AS} . W przeciwnym razie w grafie kodu AA-LDPC pojawi się odpowiadający mu zbiór \mathcal{AS} , o identycznej strukturze podgrafu indukowanego.

Zbiorowi $\mathcal{AS}_{(a,b)}$ w grafie \mathcal{G} może również odpowiadać zbiór $\mathcal{AS}_{(a',b')}$ przy $a' \geq a$, $b' \geq b$ w grafie $\mathcal{G}^{(\mathcal{P},\Theta)}$. Nie są znane ogólne twierdzenia określające obecność takich zbiorów w $\mathcal{G}^{(\mathcal{P},\Theta)}$. Zasadne wydaje się jednak założenie, że „usunięcie” na etapie ekspansji jak największej liczby cykli należących do danego $\mathcal{AS}_{(a,b)}$ daje mniejsze prawdopodobieństwo zaistnienia w grafie zbiorów $\mathcal{AS}_{(a',b')}$ o małych wartościach a' , b' .

Przykładowo na rys. 5.6 przedstawiono podgraf indukowany przez $\mathcal{AS}_{(6,0)}$ w grafie \mathcal{G} . Jeśli na etapie ekspansji zostanie usunięty tylko jeden spośród 8-cykli, to w grafie $\mathcal{G}^{(\mathcal{P},\Theta)}$ istnieje $\mathcal{AS}_{(6,2)}$ pokazany po prawej stronie rysunku 5.6. Nie jest on już tak „szkodliwy” jak $\mathcal{AS}_{(6,0)}$, tym niemniej usunięcie większej liczby cykli zapobiegłoby powstaniu $\mathcal{AS}_{(6,2)}$. Jest to uzasadnieniem celowości „usuwania” w pierwszej kolejności wszystkich cykli należących do podgrafów indukowanych przez małe zbiory

\mathcal{AS} , co jest jednym z elementów algorytmu, który zostanie przedstawiony w dalszej części rozdziału.



Rys. 5.6: Ekspansja podgrafu indukowanego przez $\mathcal{AS}_{6,0}$

5.5.4 Wyszukiwanie cykli oraz zbiorów \mathcal{AS} w grafie bazowym

Graf bazowy kodu utworzony np. za pomocą algorytmu 5.3 poddawany jest analizie, w celu wyznaczenia krótkich cykli oraz podgrafów indukowanych przez małe zbiory \mathcal{AS} , które mogą być źródłem identycznych struktur w grafie $\mathcal{G}^{(P,\Theta)}$. Do tego celu opracowano odpowiednie algorytmy wyszukiwania tych szkodliwych struktur.

Algorytm 5.4 umożliwia wyszukanie w grafie bazowym \mathcal{G} cykli o długości $k \leq K_{\max}$. W tym celu dla każdego wierzchołka bitowego, budowane jest drzewo grafu i identyfikowane są powtórzenia wierzchołków pojawiających się na kolejnych poziomach drzewa. Jeśli korzeniem drzewa jest węzeł b_l , to zbiór węzłów bitowych b_n , które są wstawiane do drzewa na kolejnych etapach ograniczany jest do tych o indeksach $n \geq l$. Dzięki temu każdy cykl znajdujący się tylko raz (dla drzewa z korzeniem w wierzchołku o najmniejszym indeksie). Cykle umieszczane są na liście LIST_{CYC} , przy czym listą określaną będzie uporządkowany zbiór. Należy zauważyć, że wyszukiwane są wszystkie cykle (nie tylko elementarne), przykładowo 8-cykle złożone z dwukrotnie powtórzonej ścieżki stanowiącej 4-cykl.

Jak zaobserwowano, liczba ścieżek w drzewie (szczególnie dla kodów o wysokiej stopie R oraz kodów nieregularnych) gwałtownie rośnie dla $k \geq 4$. W celu ograniczenia złożoności algorytmu, z drzewa mogą być wówczas usuwane ścieżki o wartości parametru EMD większej od pewnego progu EMD_{\max} . Stąd też, jeśli zostanie ustalona skończona wartość parametru EMD_{\max} , to algorytm umożliwi wyszukiwanie wszystkich $(2k)$ -cykli dla $k \leq 4$ oraz $(2k)$ -cykli o małych wartościach EMD dla

$k > 4$. Przykładowo, eksperymentalnie ustalono typowe wartości dla kodów o bloku N rzędu kilku tysięcy bitów: $K_{\max} = 12$, $\text{EMD}_{\max} = 5$ dla kodów nieregularnych oraz kodów o stopie $R > 0.6$, w pozostałych przypadkach $K_{\max} = 14$, $\text{EMD}_{\max} = 6$. Dla tych wartości liczba wyszukanych cykli (rzędu kilku-kilkudziesięciu tysięcy) jest zwykle większa niż liczba możliwych do „usunięcia” na etapie ekspansji grafu, poszukiwanie dłuższych cykli jest więc bezcelowe.

Wartość EMD cyklu (wiersz 7 algorytmu) wyznaczana jest poprzez utworzenie podmacierzy składającej się z kolumn macierzy bazowej odpowiadających wierzchołkom bitowym należącym do cyklu, a następnie wyznaczenie liczby wierszy tej podmacierzy o nieparzystej liczbie jedynek. Łatwo wykazać, że liczba ta jest równa wartości EMD podzbioru wierzchołków należących do cyklu.

Algorytm 5.4: Algorytm wyszukiwania cykli	
	Dane wejściowe: Graf $\mathcal{G} = (\mathcal{V}_b \cup \mathcal{V}_c, \mathcal{E})$, $\mathcal{V}_b = \{b_1, \dots, b_L\}$, $\mathcal{V}_c = \{c_1, \dots, c_D\}$, parametry K_{\max} , EMD_{\max}
	Dane wyjściowe: Lista cykli LIST_{cyc} oraz lista wartości EMD, LIST_{EMD}
1	DLA $l = 1, \dots, L$ (dla każdego wierzchołka bitowego)
2	Utworzyć graf składający się z wierzchołka b_l . Graf nazywany będzie drzewem, a wierzchołek b_l jest korzeniem oraz początkowo jedynym liściem.
3	DLA $k = 1, \dots, K_{\max}/2$
4	Rozbudować drzewo o jeden poziom: dla każdego liścia wstawić do drzewa wierzchołki odpowiadające wierzchołkom sąsiadującym w grafie bazowym \mathcal{G} oraz odpowiednie krawędzie; w zależności od k do drzewa wstawiane są wierzchołki bitowe b_n (dla $n \geq l$) lub kontrolne c_m (dla dowolnego m)
5	Nowo wstawione wierzchołki stają się liśćmi drzewa.
6	Jeśli dwa liście drzewa odpowiadają tym samym wierzchołkom grafu bazowego, to w grafie bazowym istnieje $(2k)$ -cykl składający się z połączenia łańcuchów od korzenia drzewa b_l do tych liści. Wszystkie cykle umieszczane są na liście LIST_{cyc}.
7	Wyznaczyć wartość EMD cykli i umieścić na liście LIST_{EMD}
8	Jeśli $k \geq 4$, z drzewa grafu usunąć łańcuchy (od korzenia do liścia) o wartości parametru $\text{EMD} > \text{EMD}_{\max}$.

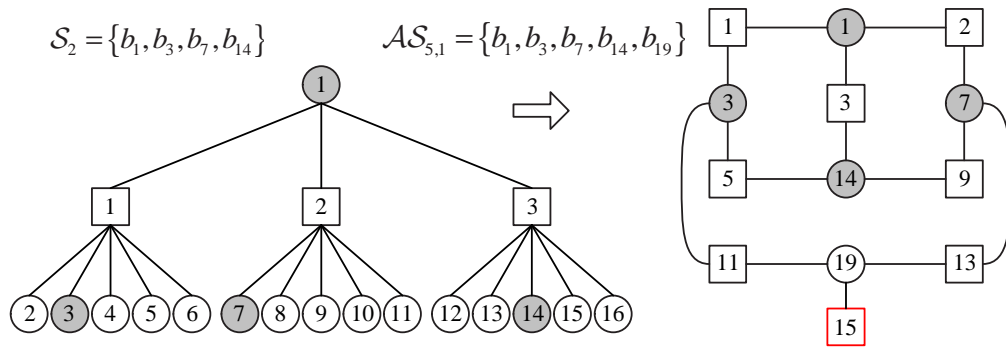
Drugim etapem analizy grafu bazowego jest wyszukiwanie zbiorów \mathcal{AS} (\mathcal{TS}). Sposób dokładnego przeszukiwania grafu Tannera w celu wyznaczenia tych zbio-

rów, polegający na sprawdzaniu wszystkich możliwych kombinacji węzłów bitowych, nawet dla niewielkich rozmiarów grafu jest bardzo czasochłonny. W niniejszej pracy zastosowano nieco odmienne podejście, polegające na identyfikacji kombinacji błędnych bitów w procesie dekodowania odpowiednio utworzonej porcji wektorów wejściowych. Autor pracy [80] proponuje oznaczenie jako \mathcal{TS} takich zbiorów wierzchołków, dla których skojarzone bity po pewnej liczbie iteracji dekodowania BP nie zostały skorygowane, przy zastosowaniu modelu kanału AWGN. Efektywniejsza metoda opisana w [7, 18, 19], zamiast kanału AWGN wykorzystuje zakłócenie pewnej liczby odpowiednio wybranych bitów (tzw. *error impulse*). Metoda ta została zaadaptowana na potrzeby niniejszej pracy. Jej zaletami są znacznie krótszy czas obliczeń w porównaniu z dokładnym przeszukiwaniem grafu, jak również zapewnienie wstępnej weryfikacji „szkodliwości” znalezionych podgrafów indukowanych przez \mathcal{TS} . Zbiory $\mathcal{TS}_{a,b}$ wyszukiwane są bowiem na drodze symulacji, tak więc potwierdzone jest, że (dla pewnego określonego zakłócenia) proces dekodowania zostaje faktycznie zatrzymany, przy a bitach (skojarzonych z wierzchołkami \mathcal{TS}) przekłamanymi oraz b niespełnionych równaniach kontrolnych. We wspomnianych publikacjach, wyszukiwanie zbiorów \mathcal{TS} jest wykorzystywane jako fragment algorytmu estymacji wartości BER systemu kodowania LDPC, a w niniejszej pracy zostało wykorzystane do analizy grafu bazowego kodu dla celów optymalnej ekspansji grafu.

Metoda wyszukiwania zbiorów \mathcal{AS} w grafie bazowym (algorytm 5.5) polega na przeprowadzeniu pewnej liczby symulacji procesu dekodowania dla kodu o macierzy kontrolnej \mathbf{W} , przy zakładanym prawidłowym słowie kodowym składającym się z samych zer. W pierwszej kolejności określany jest zbiór zakłóceń słowa kodowego, który będzie oznaczony przez \mathcal{S}_2 . Dla każdego wierzchołka bitowego b_n wyznaczone są dwa poziomy drzewa grafu bazowego, przy wierzchołku b_n będącym korzeniem drzewa. Do zbioru \mathcal{S}_2 dołączane są wszystkie kombinacje: wierzchołka bitowego b_n oraz po jednym wierzchołku wybranym spośród sąsiadujących z każdym z wierzchołków kontrolnych na pierwszym poziomie drzewa. Przykładowo na rysunku 5.7 przedstawiono drzewo, którego korzeniem jest wierzchołek b_1 , a do zbioru \mathcal{S}_2 należeć będą kombinacje wierzchołków $\{b_1, b_2, b_7, b_{12}\}$, $\{b_1, b_2, b_7, b_{13}\}$, ... itd.

Dla kodu regularnego o stopniach wierzchołków (d_b, d_c) , liczba kombinacji wynosi $|\mathcal{S}_2| = N(d_c - 1)^{d_b}$. Dla każdej z nich określone jest odpowiednie zakłócenie słowa kodowego, które polega na przekłamaniu bitów na pozycjach równych indeksom wierzchołków bitowych danego elementu \mathcal{S}_2 . Przekłamanie polega na zamianie wartości bitu, a tak utworzone zakłócone słowo wykorzystywane jest do wyznaczenia prawdopodobieństw *a priori* inicjalizujących zmienne algorytmu dekodowania. Wykonywanych jest zatem $|\mathcal{S}_2|$ symulacji, podczas których obserwowane są twarde

decyzje po kolejnych iteracjach. Dla przypadków gdy dekodowanie zakończone jest niepowodzeniem, wśród zapamiętanej historii twardej decyzji identyfikowane są te, dla których liczba niespełnionych równań kontrolnych jest minimalna, a wierzchołki odpowiadające bitom, które są wówczas przekłamane, stanowią zbiory \mathcal{TS} . Liczba niespełnionych równań kontrolnych równa jest wadze Hamminga syndromu błędu, tzn. $b = w_H(\mathbf{H}\hat{\mathbf{x}}^{(i)T})$, gdzie $w_H(\mathbf{x})$ to waga Hamminga wektora \mathbf{x} (liczba jego niezerowych elementów). Każdy $\mathcal{TS}_{(a,b)}$ dodawany jest do listy zbiorów $\text{LIST}_{\mathcal{AS}}$, jeśli tylko jest on zbiorem $\mathcal{AS}_{(a,b)}$ oraz parametry (a, b) mają małą wartość, czyli mniejszą od pewnego założonego maksimum a_{\max}, b_{\max} .



Rys. 5.7: Drzewo grafu o korzeniu b_1 oraz znaleziony $\mathcal{AS}_{(5,1)}$

Przykład 8.

Przy zakłóceniu bitów $\{b_1, b_2, b_7, b_{14}\}$ kodu o drzewie grafu z korzeniem b_1 przedstawionym na rysunku 5.7, za pomocą algorytmu 5.5 został wyznaczony zbiór $\mathcal{AS}_{(5,1)}$. Podgraf indukowany przez ten zbiór przedstawiony jest po prawej stronie rysunku 5.7. Wstępne zakłócenie bitów $\{b_1, b_2, b_7, b_{14}\}$ powoduje (po kilku iteracjach dekodowania) przekłamanie bitu b_{19} . Wówczas tylko jedno równanie kontrolne (skojarzone z wierzchołkiem c_{15}) jest niespełnione.

Ten sam $\mathcal{AS}_{(5,1)}$ został również wyznaczony dla innych zakłóceń, np. $\{b_7, b_1, b_{14}, b_{19}\}$ (drzewo o korzeniu b_7).

Algorytm 5.5: Algorytm wyszukiwania zbiorów \mathcal{AS}	
	Dane wejściowe: Macierz bazowa $\mathbf{W}_{D \times L}$, parametry a_{\max}, b_{\max}
	Dane wyjściowe: $\text{LIST}_{\mathcal{AS}}$ czyli lista zbiorów $\mathbf{AS}_{(a,b)}$, dla $a \leq a_{\max}, b \leq b_{\max}$
1	Określić zbiór zakłóceń słowa kodowego \mathcal{S}_2 : $\mathcal{S}_2 = \{b_l, \{b_n : n \in \mathcal{N}(m) \setminus l, m \in \mathcal{M}(l)\}, l = 1, \dots, L\}$
2	DLA każdego $s \in \mathcal{S}_2$
3	Do dekodera dostarczyć przekłamanane słowo $\mathbf{y} = [y_1, y_2, \dots, y_L]$, gdzie $(y_l = 1) \Leftrightarrow (b_l \in s)$ oraz $(y_l = 0) \Leftrightarrow (b_l \notin s)$ (zakładane prawidłowe słowo kodowe składa się z samych zer).
4	DLA $i = 1, \dots, i_{\max}$
5	Wykonać iterację dekodowania BP (algorytm 2.4) i zapamiętać wartość twardych decyzji $\hat{\mathbf{x}}^{(i)}$.
6	JEŻELI dekodowanie nie zakończone sukcesem
7	Do listy $\text{LIST}_{\mathcal{AS}}$ dodawane są zbiory wierzchołków bitowych skojarzonych z przekłamanymi (niezerowymi) bitami, dla $\hat{\mathbf{x}}^{(i)}$, $i = 1, \dots, i_{\max}$ spełniających następujące warunki: <ul style="list-style-type: none"> 1. Liczba niespełnionych równań kontrolnych $b = w_H(\mathbf{H}\hat{\mathbf{x}}^{(i)T})$ jest minimalna, tzn. $w_H(\mathbf{H}(\hat{\mathbf{x}}^{(i)})^T) = \min_i w_H(\mathbf{H}(\hat{\mathbf{x}}^{(i)})^T)$. 2. $b \leq b_{\max}$ oraz liczba bitów przekłamanych $a \leq a_{\max}$. 3. Konfiguracja $(l_{\text{odd}} : l_{\text{even}})$ każdego z wierzchołków bitowych w podgrafie indukowanym spełnia warunek $l_{\text{even}} > l_{\text{odd}}$ (czyli spełniona jest definicja zbioru \mathcal{AS}).

Uzasadnienie metody doboru kombinacji zakłóceń (zbiór \mathcal{S}_2) przedstawiono w [19]. Jednym z elementów tego uzasadnienia jest twierdzenie mówiące o tym, że dla kodów regularnych $\mathcal{C}^{3,6}(N, K)$ o obwodzie grafu $g \geq 6$, podzbiorem każdego $\mathcal{TS}_{(a,b)}$, $a > b$ są wierzchołki przynajmniej jednej kombinacji ze zdefiniowanego zbioru \mathcal{S}_2 . Dla kodów nieregularnych, w zbiorze \mathcal{S}_2 umieszczane są jedynie kombinacje uzyskane dla drzew o korzeniach w wierzchołkach o dwóch najmniejszych wartościach stopni, co wynika ze spostrzeżenia, że zdecydowana większość zbiorów \mathcal{TS} zawiera głównie wierzchołki niskiego stopnia. Dzięki temu ograniczana jest liczba elementów \mathcal{S}_2 , która dla kodów nieregularnych może być stosunkowo wielka. Skuteczność algorytmu 5.5 została potwierdzona licznymi eksperymentami.

5.5.5 Algorytmy ekspansji macierzy bazowej

Ekspansja macierzy bazowej polega na wyznaczeniu wartości przesunięć cyklicznych s_e dla każdej krawędzi grafu bazowego kodu w taki sposób, aby w grafie $\mathcal{G}^{(\mathcal{P},\Theta)}$ istniało jak najmniej krótkich cykli oraz małych zbiorów \mathcal{AS} . Algorytm ekspansji składa się z 3 podstawowych etapów: analizy grafu bazowego, sortowania listy cykli oraz poszukiwania takich wartości s_e , dla których warunek usunięcia cyklu na etapie ekspansji (5.12) jest spełniony dla jak największej liczby cykli z listy, przy uwzględnieniu ich kolejności.

Przeprowadzono eksperymenty dla kilku sposobów wyznaczania priorytetowej listy ustalającej kolejność usuwania cykli. W niniejszej pracy przedstawione będą wyniki dla dwóch spośród nich, dla których uzyskano najlepsze rezultaty, przy czym w pierwszym algorytmie etap analizy ograniczony jest tylko do poszukiwania cykli, a w drugim poszukiwane są także zbiory \mathcal{AS} .

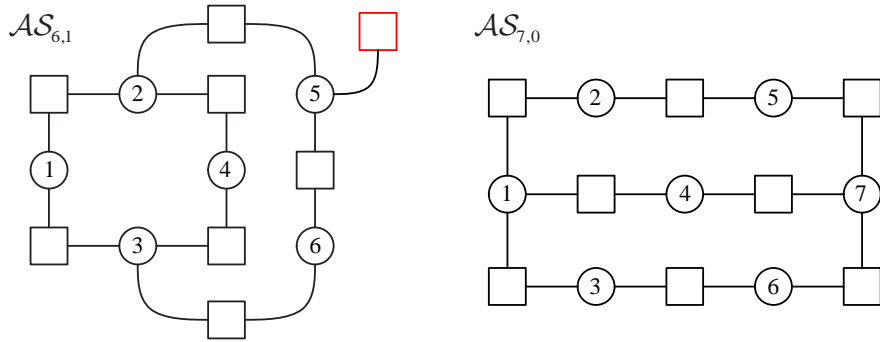
Pierwszy z opracowanych algorytmów przedstawiony jest jako algorytm 5.6 [100]. Na początku listy umieszczane są wszystkie 4-cykle, gdyż mają one najbardziej negatywny wpływ na jakość kodu. (W pewnych publikacjach definicja kodu LDPC formułowana jest w taki sposób, że skojarzony graf z definicji powinien być pozbawiony 4-cykli [57].) Następne w kolejności są wszystkie k -cykle o wartości parametru $\text{EMD} = 0$, gdyż same w sobie stanowią one zbiory $\mathcal{AS}_{(k,0)} = \mathcal{LD}_a$, a co się z tym wiąże – ograniczają wartość odległości minimalnej d_{\min} kodu. Następnie na liście umieszczane są wszystkie 6-cykle, sortowane według wartości ich parametrów EMD. Wreszcie pozostałe k -cykle, $k \geq 8$, przy czym ich kolejność zależy od długości k oraz parametru EMD, a ściślej od sumy $(k/2 + 1.1 \text{ EMD})$. Wzór ten zaproponowany został dla uwzględnienia w pierwszej kolejności „szkodliwości” cyklu (EMD), a następnie jego długości k (ściślej: liczby wierzchołków bitowych w cyklu równej $k/2$). Jak wykazały eksperymenty, wartości współczynników we wzorze nie są krytyczne dla skuteczności algorytmu, tzn. zbliżone wyniki otrzymuje się przy zastosowaniu podobnych wartości współczynników przy k oraz EMD, np. $(k/2 + 1.5 \text{ EMD})$ bądź też $(k/2 + \text{EMD})$.

Algorytm 5.6: Algorytm I ekspansji macierzy bazowej**Dane wejściowe:** Macierz bazowa $\mathbf{W}_{D \times L}$, parametry P , K_{\max} , EMD_{\max} **Dane wyjściowe:** Macierz kontrolna $\mathbf{H}_{DP \times LP}$

- 1 Wyznaczyć listę cykli LIST_{CYC} o długości $k \leq K_{\max}$, za pomocą algorytmu 5.4, z parametrami K_{\max} , EMD_{\max} .
- 2 Posortować cykle na liście LIST_{CYC} według następującej kolejności:
 - 1) 4-cykle, w kolejności od tych o najmniejszej wartości EMD,
 - 2) pozostałe cykle o parametrze $\text{EMD}=0$, w kolejności od najkrótszych,
 - 3) pozostałe 6-cykle, w kolejności od tych o najmniejszej wartości EMD,
 - 4) pozostałe k -cykle, w kolejności od tych o najmniejszej wartości sumy $(k/2 + 1.1 \text{EMD})$.
- 3 Wyznaczyć wartości przesunięć cyklicznych s_e skojarzonych ze wszystkimi krawędziami grafu bazowego $e \in \mathcal{E}$, za pomocą algorytmu 5.8, do którego przekazywana jest lista cykli LIST_{CYC}
- 4 Dokonać ekspansji macierzy bazowej: dla każdego $w_{d,l} = 1$, skojarzona podmacierz permutacji $\mathbf{P}_{d,l}$ jest przesunięciem cyklicznym kolumn macierzy jednostkowej w lewo o s_{e_i} , gdzie $e_i = (b_l, c_d)$, por. rys. 5.9.

W wyniku przeprowadzonych analiz oraz eksperymentów okazało się, że możliwa jest poprawa osiąganych rezultatów poprzez rozbudowanie algorytmu o etap bezpośredniego poszukiwania podgrafów indukowanych przez zbiory $\mathcal{AS}_{(a,b)}$ w grafie bazowym. Doprowadziło to do powstania algorytmu 5.7, gdzie dodatkowo na liście priorytetowej, która jest tu oznaczana $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$, uwzględniane są cykle należące do podgrafów indukowanych przez znalezione zbiory $\mathcal{AS}_{(a,b)}$. Usunięcie krótkich cykli oraz cykli o małej wartości EMD nie gwarantuje bowiem usunięcia małych zbiorów $\mathcal{AS}_{(a,b)}$. Przykładowo na rysunku 5.8 przedstawiono podgrafy indukowane przez zbiory \mathcal{AS} , które znaleziono w grafie bazowym skojarzonym z macierzą kodu nieregularnego o rozmiarze 32×64 . W przedstawionych grafach najkrótsze cykle mają długość (odpowiednio) 8 i 10. Tymczasem w tym przypadku dla np. $P = 16$ nie jest możliwe usunięcie na etapie ekspansji wszystkich 8-cykli, a tym bardziej 10-cykli, żadną metodą. Usunięcie jednego z 10-cykli zawartych w podgrafie indukowanym przez $\mathcal{AS}_{(7,0)}$ po prawej stronie rys. 5.8 jest natomiast praktycznie zagwarantowane przy zastosowaniu algorytmu 5.7, gdyż cykle należące do $\mathcal{AS}_{(a,0)}$ umieszczane są bardzo wysoko na liście priorytetowej. Również duże jest prawdopodobieństwo usunięcia jednego z cykli podgrafu po lewej stronie rysunku 5.8.

Zgodnie z twierdzeniem 5.2 usunięcie pojedynczego cyklu należącego do podgrafu indukowanego przez $\mathcal{AS}_{(a,b)}$ w grafie \mathcal{G} powoduje, że w grafie $\mathcal{G}^{(P,\Theta)}$ nie istnieje odpo-



Rys. 5.8: Podgrafy indukowane przez zbiory $\mathcal{AS}_{6,1}$ oraz $\mathcal{AS}_{7,0}$

wiadający mu $\mathcal{AS}_{(a,b)}$. Tak więc na liście $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$ w algorytmie 5.7 umieszczane są pojedyncze (losowo wybrane) cykle należące do \mathcal{AS} . Najwyżej na liście umieszczane są cykle należące do $\mathcal{AS}_{(a,0)}$, gdyż zbiory te równoważne są zbiorom \mathcal{LD}_a , tak więc ograniczają odległość minimalną kodu do wartości a (por. podrozdział 5.2). W dalszej kolejności na liście umieszczane są 6-cykle, a następnie rozpatrywane są zbiory $\mathcal{AS}_{(a,b)}$, kolejno od tych o najmniejszej wartości $a + 1.1b$. Wzór ten jest uogólnieniem wzoru $(k/2 + 1.1 \text{EMD})$ określającego „szkodliwość” cyklu, ponieważ – jak łatwo zauważyć – $k/2$ wierzchołków bitowych tworzących k -cykl o parametrze EMD stanowi zbiór $\mathcal{TS}_{(k/2, \text{EMD})}$.

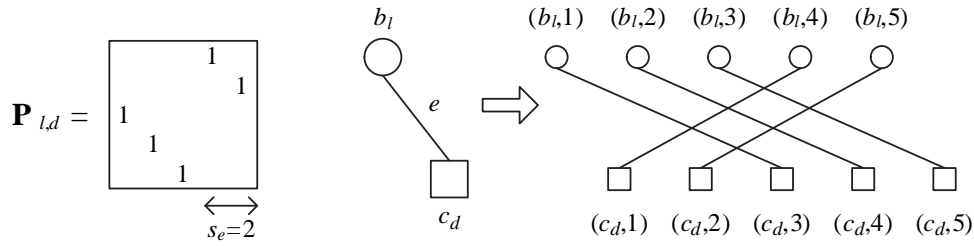
Zaproponowana kolejność umieszczania cykli na liście $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$ jest wynikiem szeregu eksperymentów. Dodatkowo, dla k -cykli o $k \geq 8$, o kolejności ich umieszczania decyduje to, czy dany cykl pojawia się jako element jednej (lub kilku) spośród zidentyfikowanych struktur \mathcal{AS} . Dzięki temu większa liczba cykli należących do małych \mathcal{AS} jest usuwana.

Algorytm 5.7: Algorytm II ekspansji macierzy bazowej

Dane wejściowe: Macierz bazowa $\mathbf{W}_{D \times L}$, parametry P , K_{\max} , EMD_{\max} ,
 a_{\max} , b_{\max}

Dane wyjściowe: Macierz kontrolna $\mathbf{H}_{DP \times LP}$

- 1 Wyznaczyć listę cykli LIST_{CYC} o długości $k \leq K_{\max}$, za pomocą algorytmu 5.4, z parametrami K_{\max} , EMD_{\max} .
- 2 Wyznaczyć listę $\text{LIST}_{\mathcal{AS}}$ zbiorów $\mathcal{AS}_{(a,b)}$, dla $a \leq a_{\max}$, $b \leq b_{\max}$, za pomocą algorytmu 5.5.
- 3 Dla każdego \mathcal{AS} z listy $\text{LIST}_{\mathcal{AS}}$, w podgrafie indukowanym wyznaczyć k -cykle o dwóch najmniejszych wartościach k .
- 4 Utworzyć nową listę $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$ i umieścić na niej cykle w następującej kolejności:
 - 1) 4-cykle, kolejno od tych o najmniejszej wartości EMD.
 - 2) Dla każdego $\mathcal{AS}_{a,0}$ z listy $\text{LIST}_{\mathcal{AS}}$, kolejno od najmniejszych a : jeśli żaden cykl należący do podgrafu indukowanego przez \mathcal{AS} nie został dotychczas umieszczony na liście $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$, to umieścić na niej losowo wybrany spośród cykli o najmniejszej długości należących do tego podgrafu.
 - 3) Pozostałe 6-cykle, w kolejności od tych o najmniejszej wartości EMD.
 - 4) Dla każdego $\mathcal{AS}_{a,b}$, $0 < b \leq b_{\max}$ z listy $\text{LIST}_{\mathcal{AS}}$, kolejno od tych o najmniejszej wartości sumy $(a + 1.1b)$: jeśli żaden cykl należący do podgrafu indukowanego przez \mathcal{AS} nie został dotychczas umieszczony na liście $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$, to umieścić na niej cykl losowo wybrany spośród cykli o najmniejszej długości należących do tego podgrafu.
 - 5) Pozostałe k -cykle, w kolejności od tych o najmniejszej wartości sumy $(k/2 + 1.1 \text{EMD})$; jeśli dwa cykle mają identyczną wartość tej sumy, to najpierw na liście umieszczany jest ten, który występuje (częściej) jako fragment podgrafów indukowanych przez zbiory z listy $\text{LIST}_{\mathcal{AS}}$.
- 5 Wyznaczyć wartości przesunięć cyklicznych s_e skojarzonych ze wszystkimi krawędziami grafu bazowego $e \in \mathcal{E}$, za pomocą algorytmu 5.8, do którego przekazywana jest lista cykli $\text{LIST}_{(\text{CYC}, \mathcal{AS})}$
- 6 Dokonać ekspansji macierzy bazowej: dla każdego $w_{d,l} = 1$, skojarzona podmacierz permutacji $\mathbf{P}_{d,l}$ jest przesunięciem cyklicznym kolumn macierzy jednostkowej w lewo o s_{e_i} , gdzie $e_i = (b_l, c_d)$, por. rys. 5.9.



Rys. 5.9: Przykładowa podmacierz przesunięcia cyklicznego o $s_e = 2$ oraz skojarzone fragmenty grafów

Algorytm wyznaczania wartości przesunięć cyklicznych

Zgodnie z definicją ekspansji macierzy, dla każdej krawędzi e_i grafu bazowego należy określić wartość skojarzonego z nią przesunięcia cyklicznego macierzy permutacji s_{e_i} , $0 \leq s_{e_i} < P$. W tym celu opracowano algorytm 5.8, polegający na sukcesywnej modyfikacji wartości s_{e_i} , w celu spełnienia warunków usunięcia kolejnych cykli z uporządkowanej listy LIST_{CYC} . Wszystkie wartości s_{e_i} , umieszczone w wektorze kolumnowym $\mathbf{S} = \{s_{e_1}, s_{e_2}, \dots, s_{e_{|\mathcal{E}|}}\}^T$, inicjalizowane są zerami. Następnie, kolejno dla każdego cyklu z listy LIST_{CYC} , modyfikowane są wartości s_{e_i} skojarzone z krawędziami należącymi do cyklu w taki sposób, aby:

- spełniony był warunek usunięcia cyklu (5.12),
- zachowane były warunki usunięcia cykli rozpatrywanych we wcześniejszych krokach,
- suma wartości s_{e_i} , równa normie L1 wektora $\|\mathbf{S}\|_1$ była jak najmniejsza.

Ostatni spośród powyższych warunków ma na celu pozostawienie jak największej swobody dla usuwania kolejnych cykli z listy. Modyfikacja wartości s_{e_i} polega bowiem na próbach jej inkrementacji (w razie potrzeby wielokrotnej) aż do spełnienia warunku usunięcia cyklu, przy zachowaniu warunków dla cykli rozpatrywanych wcześniej. Im mniejsza wartość $\|\mathbf{S}\|_1$, tym większe są możliwości inkrementacji poszczególnych elementów s_{e_i} . Przykładowo, dla pierwszego cyklu z listy, przy wszystkich wartościach s_{e_i} równych zeru, inkrementacja o 1 tylko jednej wartości s_{e_i} skojarzonej z dowolną krawędzią e_i należącą do cyklu gwarantuje spełnienie warunku usunięcia cyklu.

Wektor $|\mathcal{E}|$ -elementowy **Cyc** wyznaczany w wierszu 4 algorytmu 5.8 tworzony jest w celu określenia warunku usunięcia cyklu za pomocą iloczynu wektorów:

$$\mathbf{Cyc} \cdot \mathbf{S} \not\equiv 0 \pmod{P} \quad (5.13)$$

Przykład 9. Warunek usunięcia cyklu w postaci równania macierzowego

Niech $|\mathcal{E}| = 12$, $\mathbf{S} = \{s_{e_1}, s_{e_2}, \dots, s_{e_{12}}\}^T$.

Rozpatrzmy cykl: $e_2 \sim e_4 \sim e_9 \sim e_5 \sim$

Warunek usunięcia cyklu jest wówczas wyrażony wzorem: $s_{e_2} - s_{e_4} + s_{e_9} - s_{e_5} \not\equiv 0 \pmod{P}$, co w postaci mnożenia wektora wierszowego przez wektor kolumnowy można zapisać następująco:

$$\begin{bmatrix} 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{S} \not\equiv 0 \pmod{P}$$

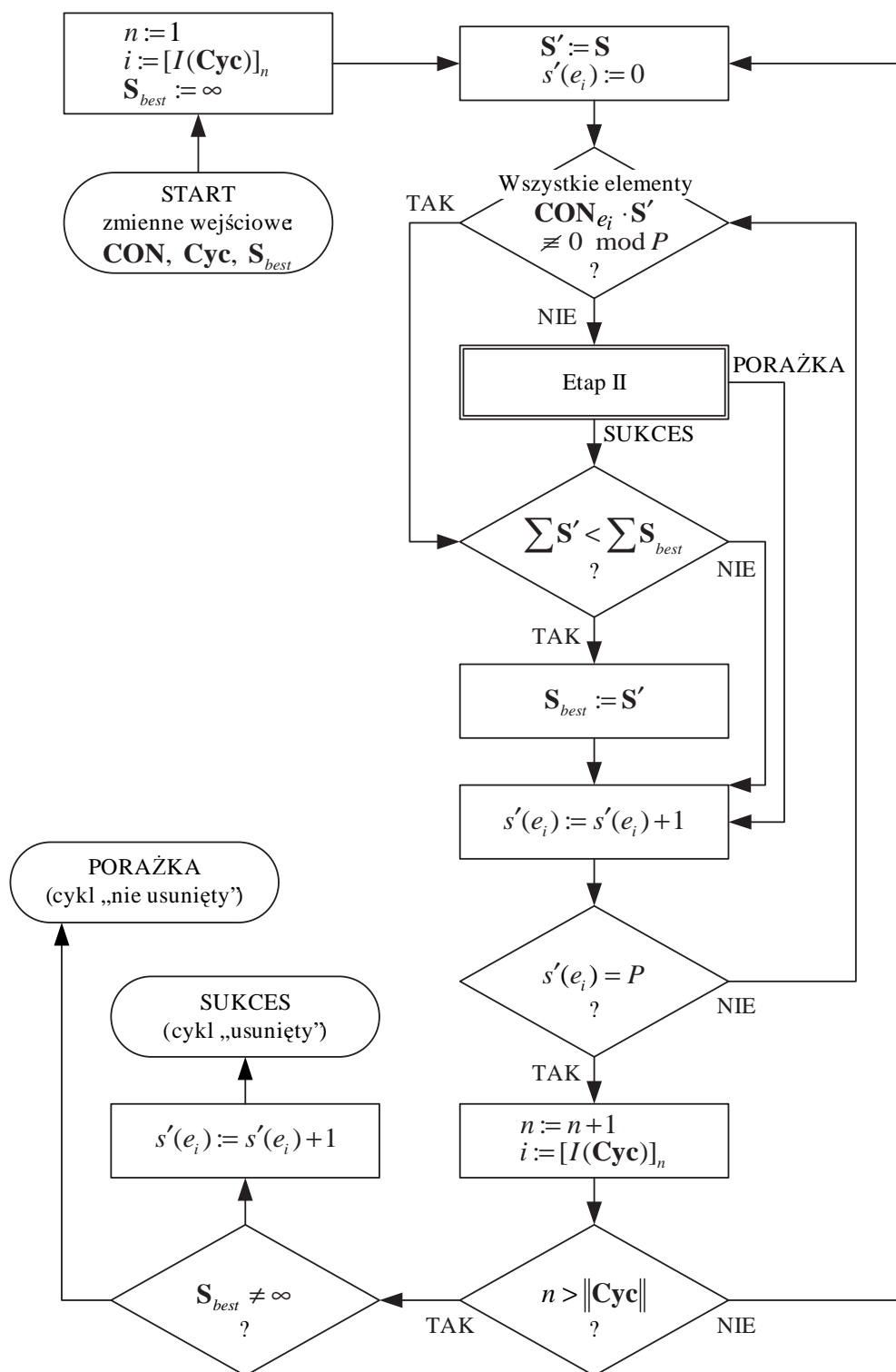
Zatem w wektorze $\mathbf{Cyc} = \begin{bmatrix} 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$, niezerowe wartości występują na pozycjach odpowiadających indeksom krawędzi cyklu (na przemian 1 i -1 , dla kolejnych krawędzi cyklu).

Dla każdej krawędzi e_i , $i = 1, \dots, |\mathcal{E}|$ tworzona jest macierz ograniczeń (*constraints*) \mathbf{CON}_{e_i} , w której poszczególne wiersze określają warunki usunięcia wszystkich rozpatrzonych do danego etapu cykli zawierających krawędź e_i . Każda z macierzy \mathbf{CON}_{e_i} jest zatem uzupełniana o wiersz równy **Cyc**, jeśli tylko e_i należy do rozpatrywanego cyklu. Macierze ograniczeń wykorzystywane są w celu kontroli warunków usunięcia cykli zawierających krawędź e_i . Jeśli wszystkie elementy wektora będącego wynikiem mnożenia ($\mathbf{CON}_{e_i} \cdot \mathbf{S}$) nie przystają do $0 \pmod{P}$, to spełnione są ograniczenia zawierające wartość s_{e_i} . Po każdej modyfikacji wartości s_{e_i} należy zatem skontrolować wynik mnożenia macierzowego ($\mathbf{CON}_{e_i} \cdot \mathbf{S}$).

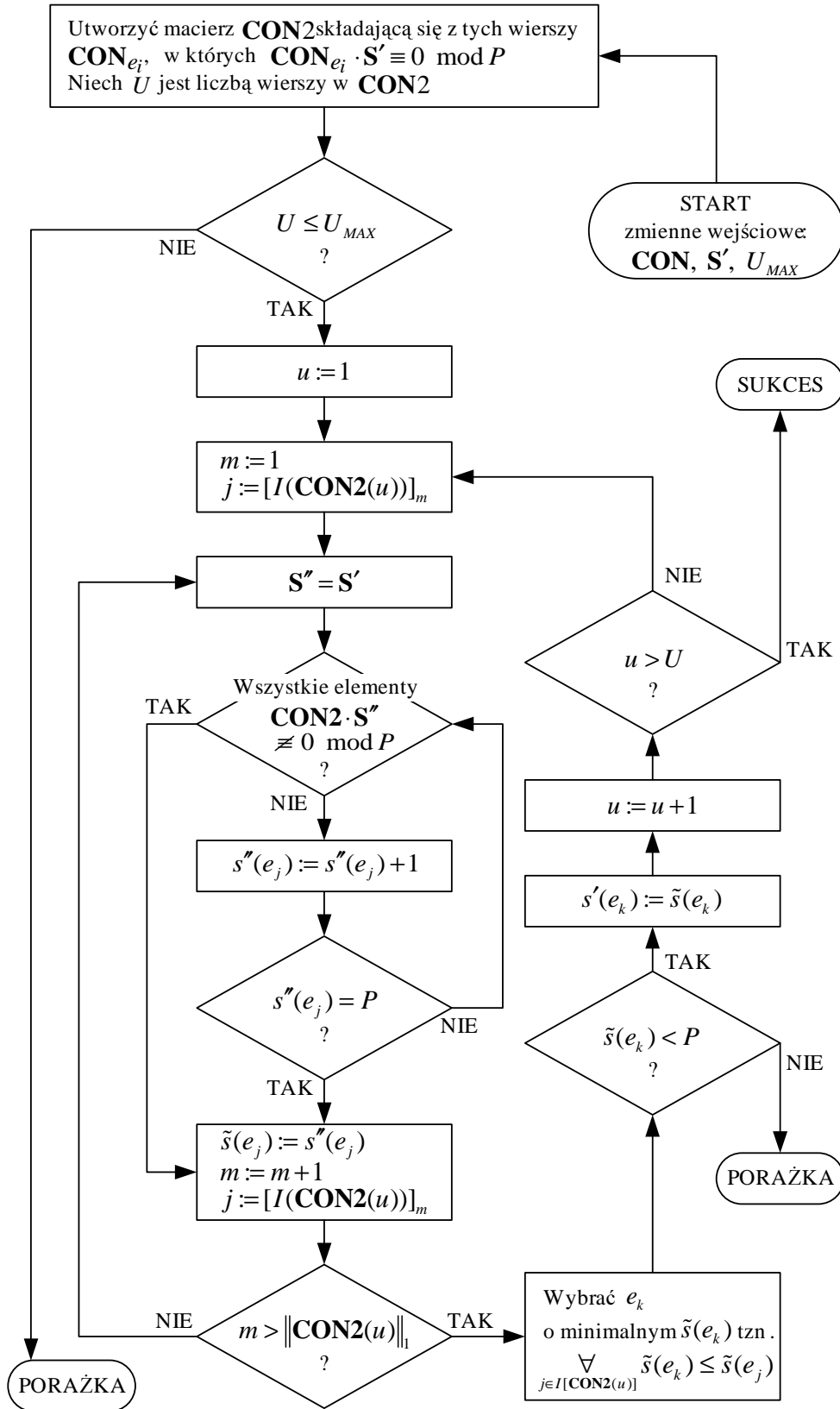
Fragment algorytmu 5.8, w którym modyfikowana jest wartość wektora **S** prowadząca do usunięcia pojedynczego cyklu, przedstawiony jest na rysunkach 5.10 i 5.11. Aktualna wartość wektora **S** umieszczana jest w tymczasowej zmiennej **S'**, dla której wykonywane są próby modyfikacji wartości $s'(e_i)$ dla kolejnych krawędzi e_i należących do cyklu. Zbiór indeksów tych krawędzi oznaczono przez $I(\mathbf{Cyc})$, a n -ty indeks oznaczony jest przez $[I(\mathbf{Cyc})]_n$. Liczba krawędzi cyklu równa jest liczbie niezerowych elementów wektora **Cyc**, tak więc oznaczono ją przez $\|\mathbf{Cyc}\|_1$.

Algorytm 5.8: Algorytm wyznaczania wartości przesunięć cyklicznych	
Dane wejściowe: Zbiór krawędzi \mathcal{E} , lista cykli LIST_{CYC} , rozmiar podmacierzy P	
Dane wyjściowe: Wartości przesunięć cyklicznych s_e , dla $e \in \mathcal{E}$	
1	Wektor przesunięć cyklicznych $\mathbf{S} = \{s_{e_1}, s_{e_2}, \dots, s_{e_{ \mathcal{E} }}\}^T := \mathbf{0}$
2	Macierze określające ograniczenia CON_{e_i} : macierze puste, dla $i = 1, \dots, \mathcal{E} $
3	DLA $i = 1, \dots, \text{LIST}_{\text{CYC}} $ (dla każdego cyklu na liście)
4	Określić $ \mathcal{E} $ -elementowy wektor wierszowy \mathbf{Cyc} , którego i -ty element jest równy 0, jeśli s_{e_i} nie występuje w warunku (5.12), w przeciwnym wypadku równy jest 1 lub -1 , w zależności od znaku z jakim s_{e_i} występuje w warunku (5.12).
5	Dla każdego e_i należącego do cyklu, macierz CON_{e_i} uzupełnić o wiersz równy wektorowi \mathbf{Cyc} .
6	Zmodyfikować wartości wektora \mathbf{S} , zgodnie z algorytmem przedstawionym w postaci schematów blokowych na rys. 5.10 i 5.11.

Algorytm podzielono na dwie części. Podstawowa część (rysunek 5.10) polega na wielokrotnej inkrementacji elementów $s'(e_i)$ wektora \mathbf{S}' skojarzonych z krawędziami należącymi do cyklu i równoczesnym sprawdzaniu, czy spełnione są ograniczenia definiowane przez CON_{e_i} . Jeśli pewna liczba ograniczeń jest niespełniona, następuje przejście do etapu II (rysunek 5.11), w którym przeprowadzana jest próba modyfikacji innych wartości $s'(e_i)$ w celu spełnienia tych ograniczeń. Jeśli ograniczenia zostaną spełnione (z wykorzystaniem etapu II lub bez), to sprawdzana jest wartość $\|\mathbf{S}'\|$ będąca sumą wszystkich elementów \mathbf{S}' . Jeśli jest ona mniejsza od najlepszego z dotychczas uzyskanych wyników, tzn. $\|\mathbf{S}'\| < \|\mathbf{S}_{best}\|$, to \mathbf{S}' jest zapamiętywane jako nowa wartość \mathbf{S}_{best} . Najlepszy uzyskany wynik podstawiany jest na koniec procesu jako nowa wartość \mathbf{S} .



Rys. 5.10: Algorytm wyznaczania przesunięć cyklicznych



Rys. 5.11: Etap II algorytmu wyznaczania przesunięć cyklicznych

5.5.6 Weryfikacja skuteczności algorytmów ekspansji

Przeprowadzono szereg różnorodnych eksperymentów w celu weryfikacji skuteczności opracowanych algorytmów oraz porównania przedstawionych dwóch metod ekspansji grafu bazowego (alg. 5.6 i alg. 5.7). Polegały one na symulacji systemu transmisji wykorzystującego model nadajnika i kanału AWGN w środowisku MATLAB oraz dekodera implementowany w układzie FPGA. Wyznaczone w ten sposób wykresy bitowej stopy błędów zostaną porównane z wykresami uzyskanymi dla kodów AA-LDPC utworzonych za pomocą innych metod ekspansji.

Przedstawione zostanie porównanie utworzonych kodów z kodami AA-LDPC uzyskanymi dwiema innymi metodami ekspansji, przy wykorzystaniu tej samej macierzy bazowej. Pierwsza metoda polegała na generacji macierzy permutacji w sposób losowy. W ten sposób tworzono pewną liczbę (100) macierzy kontrolnych o identycznych rozmiarach, spośród których wybierano macierz o minimalnej wartości średniej obwodu grafu Tannera \bar{g} (podobnie jak w algorytmie generacji kodów LDPC przedstawionym w [69]). Na wykresach uzyskane tym sposobem kody oznaczane będą jako „Random”.

Druga metoda odniesienia wykorzystuje algorytm przedstawiony w pracy [117] i zapewnia uzyskanie kodu o pewnym założonym obwodzie grafu. Algorytm [117] umożliwia wyznaczenie macierzy kontrolnej o takiej wielkości podmacierzy P , dla której można osiągnąć zadaną wartość obwodu g . Na potrzeby niniejszej pracy, algorytm ten wykorzystano w nieco zmienionej formie: dla zadanej wielkości P , tworzono za jego pomocą macierz kontrolną kodu o maksymalnej możliwej do uzyskania wartości obwodu (typowo $g = 6$ dla kodów o stosunkowo dużych stopach R lub $g = 8$ dla kodów o mniejszych stopach). Na wykresach uzyskane w ten sposób kody oznaczane będą jako „Girth conditioned”.

Wyniki eksperymentów uzyskano za pomocą utworzonego środowiska symulacji systemów kodowania LDPC, przy następujących założeniach:

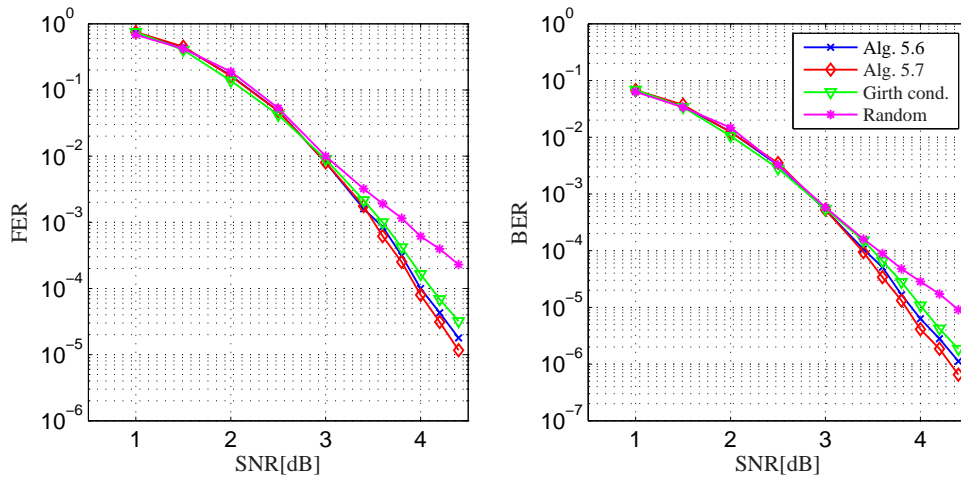
- zastosowano model kanału BIAWGN, a prawdopodobieństwa *a priori* dostarczane do dekodera obliczane zgodnie ze wzorem (1.11);
- wartość FER jest wyznaczana jako stosunek liczby błędnie zdekodowanych bloków do liczby wszystkich bloków, podobnie wartość BER to stosunek liczby błędnie zdekodowanych bitów do liczby wszystkich bitów;
- każdy z wyników symulacji (punktów na wykresie $\text{BER} = f(\text{SNR})$) uzyskano przy przynajmniej 30 wykrytych błędnych blokach;

- dekodery kodów AA-LDPC implementowano w układzie FPGA; wykorzystano algorytm Corrected-Min-Sum o parametrach $B = 7$, $Z_{th} = 22$ (dobór takich parametrów wynika z eksperymentów przedstawionych w rozdziale 4);
- maksymalna liczba iteracji wynosiła w każdym przypadku $i_{max} = 50$.

Wykorzystując opracowane algorytmy utworzono szereg kodów, o zróżnicowanych długościach bloku N , stopach R i dystrybucjach stopni wierzchołków grafu $\mathbf{d}_b, \mathbf{d}_c$. Parametry kodów, dla których wyniki zaprezentowane będą poniżej dobrano w taki sposób, aby w bazie internetowej prof. MacKaya [64] istniały ich odpowiedniki (nie zorientowane na implementację). Macierze kontrolne wszystkich utworzonych kodów zamieszczone są na stronie internetowej autora [101].

Kody regularne (273,143)

Jako przykład kodów o stosunkowo małej długości bloku ($N = 273$) posłużą kody (273,143) (stopa $R = 0.52$) o macierzy bazowej $\mathbf{W}_{10 \times 21}$ i rozmiarze podmacierzy $P = 13$. Macierz bazową utworzono algorytmem E-PEG. Wierzchołki bitowe grafów są stopnia $d_b = 3$, natomiast wierzchołki kontrolne mają stopień $d_c = 6$ lub $d_c = 7$. Ściśle rzecz biorąc nie są to zatem kody regularne, jednakże ze względu na jednolitą dystrybucję wierzchołków bitowych ich własności korekcyjne odpowiadają własnościom kodów regularnych.



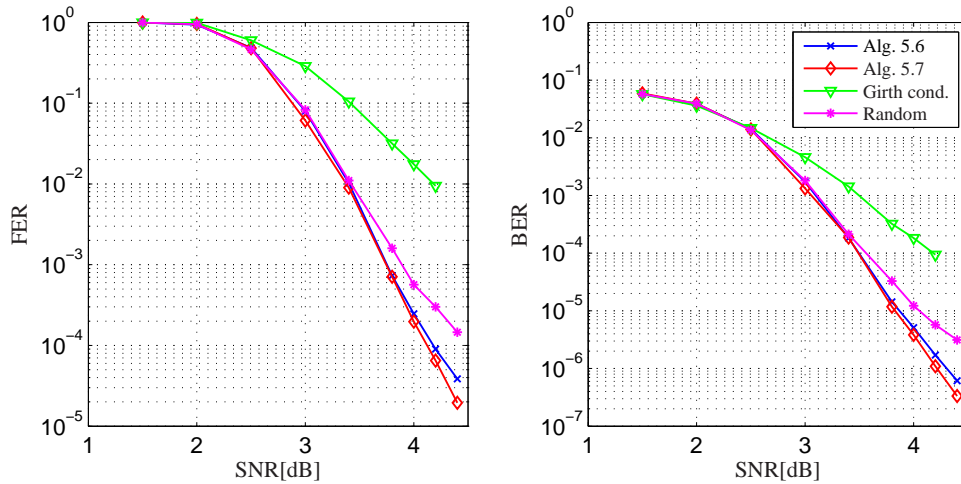
Rys. 5.12: Kody AA-LDPC (273,143) regularne

Na rys. 5.12 przedstawiono wykresy blokowej oraz bitowej stopy błędów systemu wykorzystującego kody AA-LDPC utworzone algorytmami 5.6 i 5.7 oraz kody uży-

skane za pomocą wspomnianych metod odniesienia. Kod utworzony metodą wyboru spośród losowo wygenerowanych jest wyraźnie gorszy od pozostałych. Dobre właściwości ma kod utworzony algorytmem [117] („Girth conditioned”, $g = 8$), gdyż dla stosunkowo niewielkiego grafu, usunięcie na etapie ekspansji wszystkich 4- i 6-cykli pozwala uzyskać kod o konkurencyjnych właściwościach. Tym niemniej kody utworzone opracowanymi algorytmami są nieznacznie lepsze (rys. 5.12 – alg. 5.6 i 5.7).

Kody regularne (1056,816)

Utworzone kody regularne (1056,816) o $d_b = 3$ stanowią przykład kodów o wyższej stopie ($R = 0.77$). Rozmiar podmacierzy wynosi $P = 24$, a macierz bazowa $\mathbf{W}_{10 \times 44}$ utworzona została algorytmem E-PEG. Na rys. 5.13 przedstawiono wyniki uzyskane dla kodów AA-LDPC utworzonych algorytmami 5.6 i 5.7 oraz metodami odniesienia. W tym przypadku metoda „girth conditioned” nie pozwala na uzyskanie kodu o obwodzie $g = 8$ (tzn. niemożliwe okazało się usunięcie wszystkich 6-cykli). Na rys. 5.13 przedstawiono zatem wyniki dla kodu o obwodzie $g = 6$, który jest (jak widać) znacznie gorszy od kodów uzyskanych algorytmami 5.6 i 5.7. Usunięcie na etapie ekspansji tylko wszystkich 4-cykli nie pozwala bowiem uzyskać kodu o dobrych właściwościach korekcyjnych.

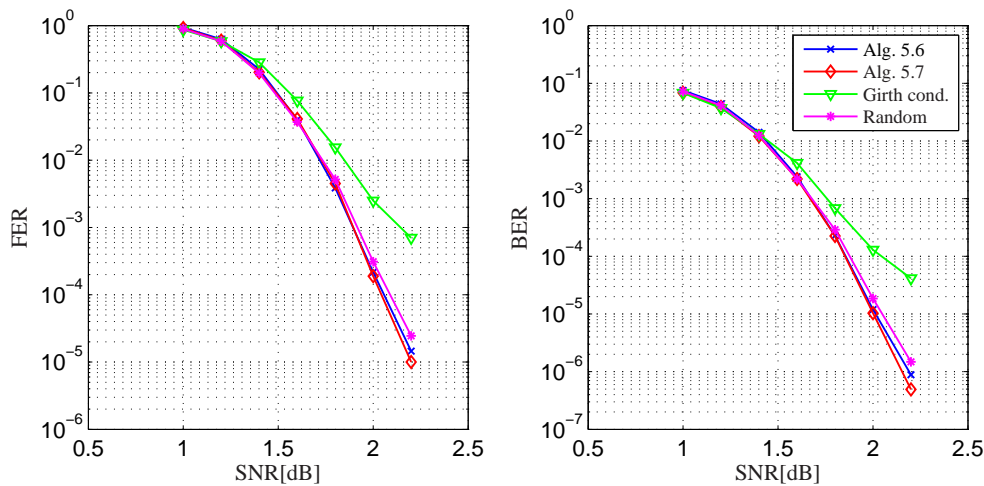


Rys. 5.13: Kody AA-LDPC (1056,816) regularne

Algorytm 5.7 daje nieco lepsze rezultaty niż 5.6 (co zresztą jest zauważalne dla prawie wszystkich wyników prezentowanych w niniejszym rozdziale).

Kody regularne (2640,1320)

Dobre własności mają również utworzone kody regularne o długości bloku rzędu kilku tysięcy i stopie $R = 0.5$. Przykładem są kody (2640,1320) o stopniach wierzchołków $d_b = 3$, $d_c = 6$, których wartości FER i BER przedstawiono na rys. 5.14. Rozmiar podmacierzy wynosi $P = 24$, a macierz bazową wygenerowano algorytmem E-PEG. W tym przypadku kod o macierzy wybranej spośród wygenerowanych losowo („Random”) ma własności zbliżone do kodów utworzonych algorytmami 5.6 i 5.7. Jest to związane ze znaną własnością metod losowych: są one skuteczniejsze dla kodów o większych długościach bloków, jak również są skuteczniejsze dla kodów regularnych (ze względu na mniejsze zagęszczenie krawędzi w grafie, a więc mniejsze prawdopodobieństwo powstania szkodliwych cykli). Tym niemniej algorytmy 5.6 i 5.7 również w tym przypadku dają nieco lepsze rezultaty.

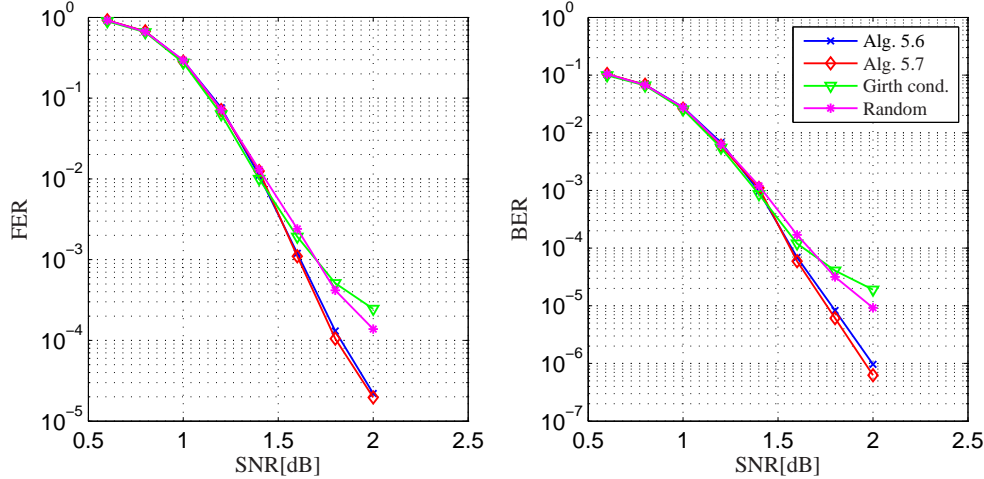


Rys. 5.14: Kody AA-LDPC (2640,1320) regularne

Kody nieregularne (2048,1024)

Na rysunku 5.15 przedstawiono wyniki uzyskane dla kodów nieregularnych (2048,1024) o maksymalnym stopniu wierzchołka bitowego $d_{b,max} = 15$ oraz następującej względnej dystrybucji krawędzi do wierzchołków bitowych: $\kappa_2 = 0.272$, $\kappa_3 = 0.259$, $\kappa_6 = 0.155$, $\kappa_7 = 0.09$, $\kappa_{14} = 0.03$, $\kappa_{15} = 0.194$ (jest to dystrybucja w przybliżeniu równa dystrybucji pewnego kodu odniesienia z bazy [64]). Rozmiar podmacierzy wynosił w tym przypadku $P = 32$, a macierz bazowa $\mathbf{W}_{32 \times 64}$ wygenerowana została algorytmem E-PEG. Jak widać utworzone kody mają ponownie lepsze własności niż kody

AA-LDPC uzyskane metodami odniesienia („Random” oraz „Girth conditioned”: $g = 8$). Algorytmy 5.6 i 5.7 dały w tym przypadku bardzo zbliżone rezultaty.



Rys. 5.15: Kody AA-LDPC (2048,1024) nieregularne $d_{b_{max}} = 15$

Analizując wyniki przedstawione na rys. 5.12-5.15 można zauważyć, że algorytm 5.7 daje nieco lepsze wyniki niż 5.6, jednakże w wielu przypadkach ta różnica jest dosyć niewielka. Zaletą algorytmu 5.6 jest jednak mniejsza złożoność koncepcyjna oraz krótszy o około 30% czas obliczeń. Z tego względu zdecydowano się zamieścić w niniejszej pracy szczegółowy opis obydwu rozwiązań.

5.6 Kompleksowy algorytm tworzenia kodu AA-LDPC

Końcowym efektem prac przedstawianych w niniejszym rozdziale jest kompleksowy algorytm tworzenia kodu AA-LDPC, obejmujący wszystkie etapy generacji macierzy kontrolnej. Algorytm wykazuje dużą skuteczność dla szerokiego zakresu parametrów kodów. Zaimplementowano go w narzędziu programowym, które umożliwi automatyczne przeprowadzenie pełnego procesu generacji „dobrego” kodu AA-LDPC, nie wymagając gruntownej wiedzy na temat przedstawionych w pracy zagadnień. Jednocześnie jest możliwość wykorzystania tylko pewnych etapów algorytmu, bądź też wykorzystania na którymś z etapów innego niż proponowane rozwiązanie. Przyjęta metodologia tworzenia kodu AA-LDPC oraz opracowane środowisko projektowo-symulacyjne sprawia, że efekty pracy mogą być natychmiast wykorzystane do dalszych badań, jak również w praktyce inżynierskiej.

Przyglądając się wynikom zaprezentowanym w poprzednim podrozdziale, jak również wynikom szeregu dalszych eksperymentów, zauważono, że algorytm ekspansji macierzy bazowej 5.7 daje zawsze nie gorsze rezultaty niż 5.6. Mając to na uwadze, w pełnym algorytmie tworzenia kodu AA-LDPC zdecydowano się skorzystać z algorytmu 5.7.

Algorytm tworzenia kodu AA-LDPC, obejmujący wszystkie etapy generacji macierzy kontrolnej kodu, przedstawiony jest jako algorytm 5.9. Stanowi on połączenie trzech algorytmów opracowanych dla realizacji kolejnych etapów: 5.1, 5.3 oraz 5.7.

Algorytm 5.9: Algorytm tworzenia kodu AA-LDPC

Dane wejściowe: Rozmiary macierzy kontrolnej M , N oraz podmacierzy P , maksymalny stopień wierzchołka bitowego $d_{b_{max}}$

Dane wyjściowe: Macierz kontrolna kodu AA-LDPC, $\mathbf{H}_{M \times N}$

- 1 Wyznaczyć dystrybucję stopni wierzchołków bitowych \mathbf{d}_b : algorytm 5.1, dane wejściowe $D = M/P$, $L = N/P$, $d_{b_{max}}$.
- 2 Utworzyć macierz bazową $\mathbf{W}_{D \times L}$: algorytm 5.3, dane wejściowe D , L , \mathbf{d}_b .
- 3 Utworzyć macierz kontrolną dokonując ekspansji macierzy bazowej: algorytm 5.7, dane wejściowe $\mathbf{W}_{D \times L}$, P oraz: K_{max} , EMD_{max} , a_{max} , b_{max} dobrane z pomocą tab. 5.2.

W tabeli 5.2 przedstawiono wyznaczone empirycznie zalecane wartości maksymalnych długości cykli oraz rozmiarów zbiorów $\mathcal{AS}_{a,b}$ poszukiwanych w grafie bazowym. Przy zastosowaniu parametrów dobranych zgodnie z tab. 5.2 liczba zidentyfikowanych cykli i podgrafów indukowanych przez $\mathcal{AS}_{a,b}$ jest większa niż liczba tych struktur możliwych do usunięcia na etapie ekspansji. Użycie wartości większych niż podane w tab. 5.2 jest oczywiście możliwe, jednakże powoduje tylko zwiększenie czasu obliczeń przy niezmiennych efektach algorytmu.

Tab. 5.2: Zalecane parametry dla algorytmu 5.7

Parametry kodu	K_{max}	EMD_{max}	a_{max}	b_{max}
$R \simeq 0.5$, $N < 1000$	10	5	10	2
$R \simeq 0.5$, $1000 \leq N < 2000$	12	6	12	3
$R \simeq 0.5$, $2000 \leq N < 4000$	14	6	14	4
$R \simeq 0.75$, $N < 1000$	8	5	10	2
$R \simeq 0.75$, $1000 \leq N < 2000$	10	6	12	3

Im mniejsza stopa kodu R oraz im większa długość bloku N , tym dłuższe cykle i większe zbiory $\mathcal{AS}_{a,b}$ mogą zostać usunięte na etapie ekspansji. Mając to na

uwadze oraz korzystając z tab. 5.2, można oszacować wartości parametrów dla kodów o dowolnych wartościach stopy R i długości bloku N . Można także kilkakrotnie przeprowadzić proces generacji macierzy, dla rosnących wartości poszczególnych parametrów, aż do uzyskania zadowalającego wyniku.

Opracowane narzędzie programowe oferuje również całkowicie automatyczny dobór wartości maksymalnych długości cykli oraz rozmiarów zbiorów $\mathcal{AS}_{a,b}$.

5.7 Własności korekcyjne uzyskanych kodów

Dla ostatecznej weryfikacji skuteczności opracowanego algorytmu tworzenia kodów AA-LDPC, własności korekcyjne uzyskanych kodów (reprezentowane przez bitową i blokową stopę błędów systemu transmisji w funkcji SNR) porównano z kodami odniesienia o tych samych (lub bardzo zbliżonych) parametrach N , R , \mathbf{d}_b , \mathbf{d}_c . Jako odniesienie posłużyły kody LDPC (nie zorientowane na implementację) umieszczone w internetowej bazie prof. MacKaya [64], jak również kod AA-LDPC użyty w standardzie IEEE 802.16e [41]. Prezentowane wyniki eksperymentów uzyskano za pomocą środowiska symulacji systemów kodowania LDPC, przy założeniach identycznych do tych poczynionych w podrozdziale 5.5.6. Dekoder kodów LDPC nie zorientowanych na implementację modelowano z wykorzystaniem środowiska MATLAB, przy dekodowaniu algorytmem LLR-BP z harmonogramem szeregowym, wykorzystującym wiadomości w postaci liczb zmiennoprzecinkowych.

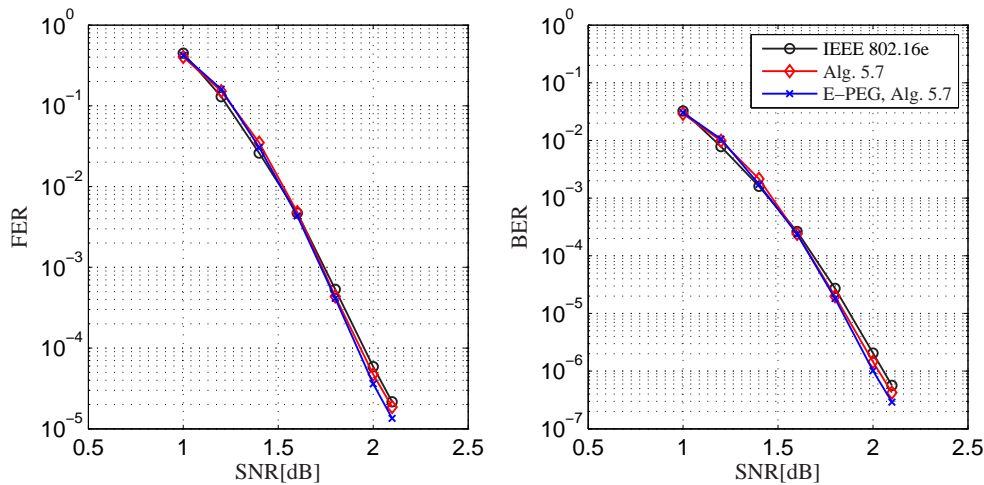
Poniżej przedstawione zostaną wyniki uzyskane dla kodów przedstawionych w podrozdziale 5.5.6 oraz kilku innych typów kodów, na tle wspomnianych kodów odniesienia, pochodzących z materiałów źródłowych.

Kody nieregularne (2304,1152)

W pierwszej kolejności przedstawione zostaną wyniki eksperymentów pozwalających zweryfikować skuteczność opracowanych algorytmów przy porównaniu do jednego z kodów wykorzystywanych w sieciach WiMAX, a zdefiniowanych w standardzie IEEE 802.16e [41]. Utworzono zatem kody o identycznych parametrach, jak kod nieregularny (2304,1152) (stopa $R = 0.5$, $d_{b_{max}} = 6$), którego macierz kontrolna zdefiniowana jest na stronie 628 w [41]. Kod ten spełnia definicje kodu AA-LDPC, a rozmiar podmacierzy wynosi $P = 96$.

Na rys. 5.16 przedstawiono wykresy bitowej stopy błędów otrzymane dla owego kodu oraz dwóch utworzonych kodów: pierwszy uzyskano przy wykorzystaniu macierzy bazowej identycznej jak ta zastosowana w kodzie [41], natomiast drugi – przy

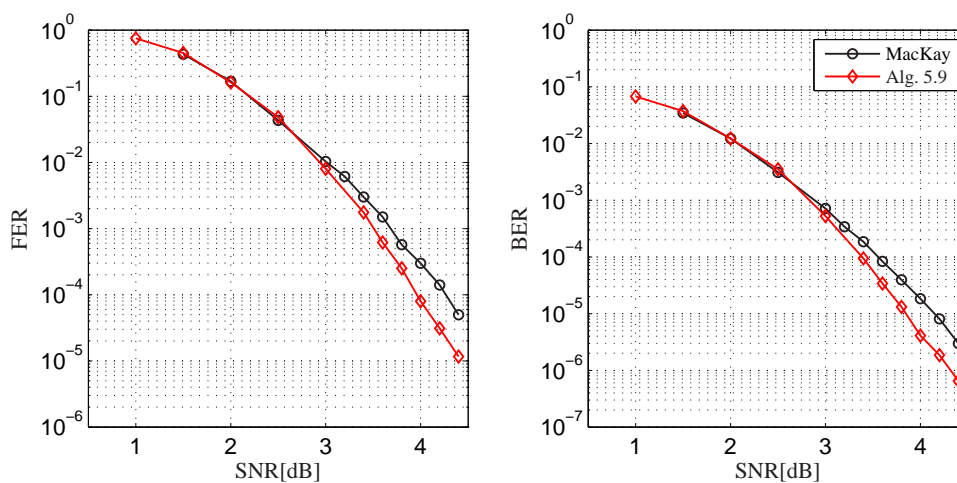
wykorzystaniu macierzy bazowej wygenerowanej algorytmem E-PEG. Jak widać, uzyskane kody mają bardzo zbliżone własności korekcyjne do kodu odniesienia. Kod uzyskany z wykorzystaniem macierzy bazowej utworzonej zaproponowanym algorytmem E-PEG wydaje się mieć nawet nieco lepsze własności niż kod przedstawiony w [41]. Autorowi nie jest znana metoda, jaką utworzono rozpatrywany kod [41], niemniej z całą pewnością autorzy standardu dołożyli starań, aby miał on dobre własności korekcyjne. Uzyskanie kodów AA-LDPC o bardzo zbliżonych własnościach wydaje się być zatem sporym sukcesem.



Rys. 5.16: Kody AA-LDPC (2304,1152) nieregularne oraz kod IEEE 802.16e [41]

Kody regularne (273,143)

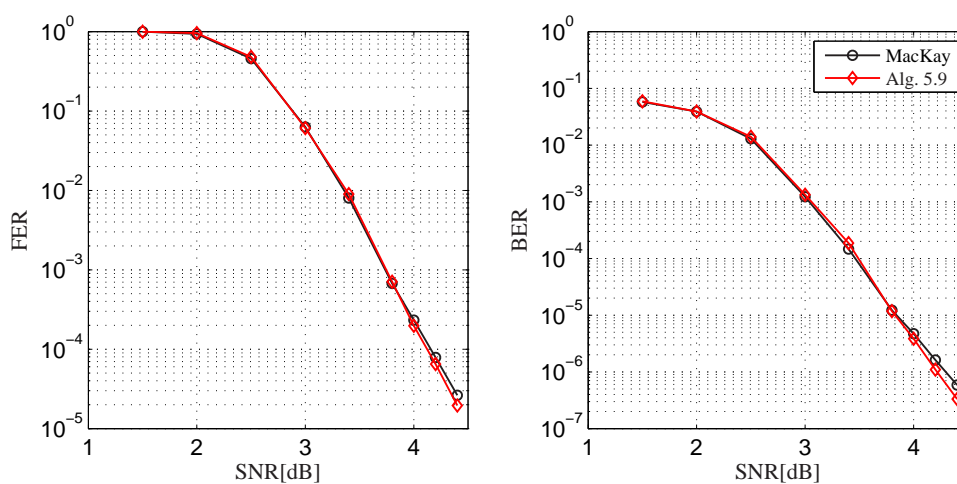
Utworzony regularny kod AA-LDPC (273,143) porównano z kodem LDPC o zbliżonych parametrach (271,144) (stopa $R = 0.53$) pochodzącym z bazy prof. MacKaya [64], o identycznej jednolitej dystrybucji wierzchołków bitowych $d_b = 3$. Jak widać na rys. 5.17, kod utworzony za pomocą algorytmu 5.9 ma zauważalnie lepsze własności korekcyjne niż kod odniesienia. Należy jednak zauważyć, że stopa kodu odniesienia jest minimalnie większa, co ma pewien niewielki udział w widocznie gorszych własnościach korekcyjnych. Kod odniesienia nie jest jednak kodem AA-LDPC – implementacja modułu szeregowo-równoległego dekodera jest niemożliwa, co stanowi o przewadze utworzonego kodu.



Rys. 5.17: Kod AA-LDPC (273,143) oraz LDPC (271,144) [64]

Kody regularne (1056,816)

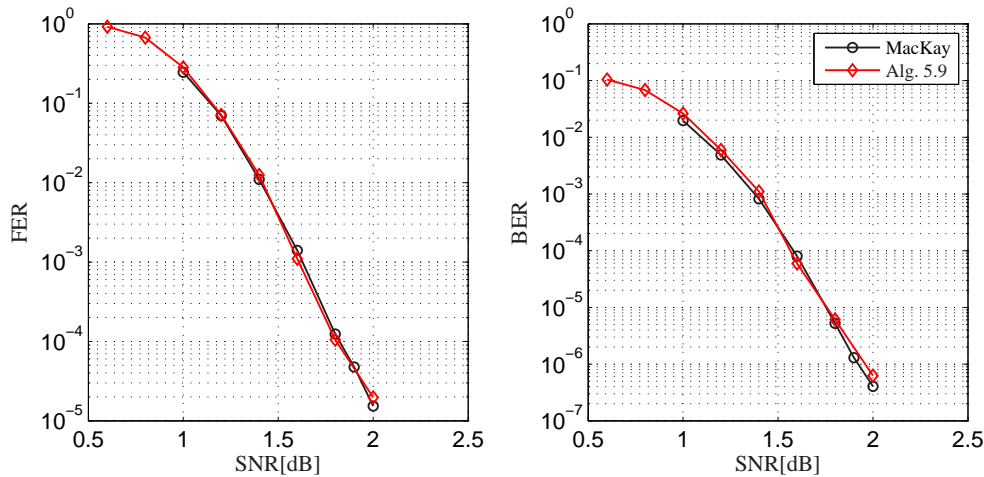
Na rys. 5.18 pokazano, że kod AA-LDPC (1056,816) utworzony algorytmem 5.9 ma bardzo zbliżone własności do kodu LDPC (1057,813) pochodzącego z bazy [64]. Niewielka różnica parametrów nie powinna mieć znaczenia dla tego porównania (utworzony kod AA-LDPC ma nawet minimalnie większą stopę, tak więc jego oczekiwane własności korekcyjne powinny być nieco gorsze).



Rys. 5.18: Kod AA-LDPC (1056,816) oraz LDPC (1057,813) [64]

Kody nieregularne (2048,1024)

Na rys. 5.19 przedstawiono wyniki uzyskane dla kodów nieregularnych (2048,1024) o maksymalnym stopniu wierzchołka bitowego $d_{b_{max}} = 15$. Jak widać, utworzony algorytmem 5.9 kod jest konkurencyjny do kodu LDPC (2048,1030) [64] o bardzo zbliżonych rozmiarach macierzy kontrolnej oraz dystrybucji wierzchołków (utworzony kod AA-LDPC ma nawet nieco większą stopę).



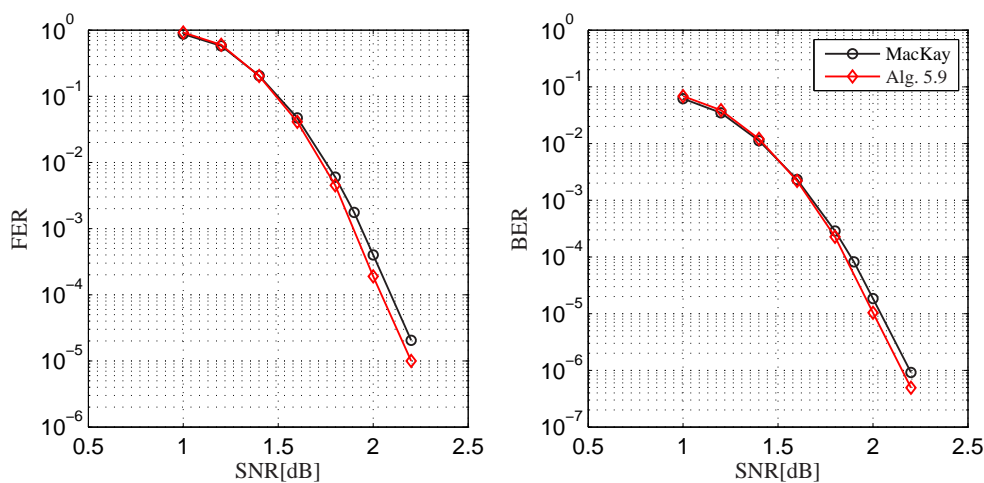
Rys. 5.19: Kod AA-LDPC (2048,1024) oraz LDPC (2048,1030) [64] nieregularne $d_{b_{max}} = 15$

Kody regularne (2640,1320)

Znakomite własności ma również utworzony kod regularny o długości bloku $N = 2640$ i stopie $R = 0.5$ (rys. 5.20). Kod AA-LDPC utworzony algorytmem 5.9 jest nieco lepszy od kodu LDPC o identycznych parametrach (rozmiary macierzy i stopnie wierzchołków grafu) pochodzącego z bazy [64] (oznaczonego tam jako „Margulis2640.1320.3”).

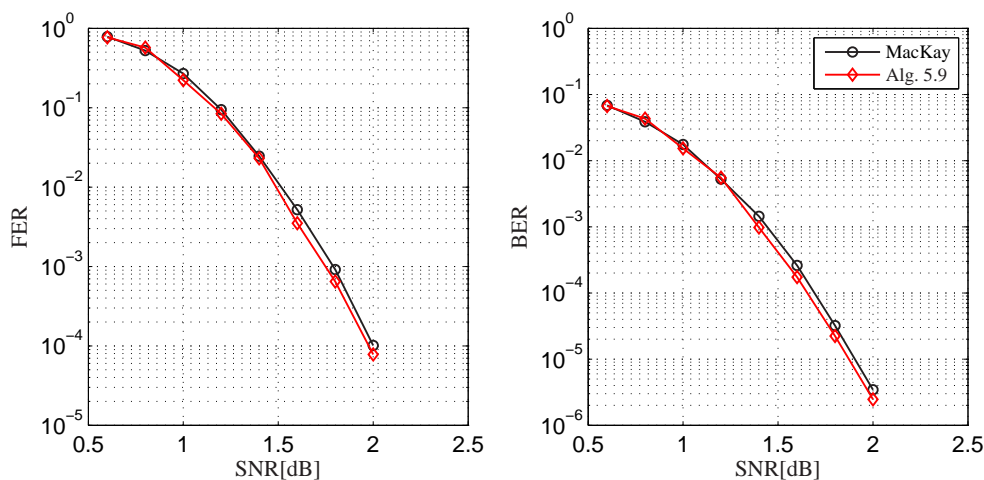
Kody nieregularne (1920,640)

Kody nieregularne (1920,640) o wierzchołkach bitowych stopnia 2 i 3 stanowią przykład kodów o stosunkowo niskiej stopie ($R = 0.33$). Kodem odniesienia był kod oznaczony w bazie [64] jako „1920.1280.3.303”. Na rys. 5.21 przedstawiono porównanie wyników uzyskanych dla tego kodu oraz kodu AA-LDPC uzyskanego algorytmem 5.9, o identycznym rozmiarze macierzy kontrolnej i rozmiarze podmacierzy



Rys. 5.20: Kody (2640,1320) AA-LDPC oraz LDPC [64]

$P = 40$. Wyniki przedstawione na rys. 5.21 potwierdzają skuteczność opracowanego algorytmu również dla kodów o niskich stopach.



Rys. 5.21: Kody (1920,640) AA-LDPC oraz LDPC [64] nieregularne $d_b = 2 \dots 3$

5.8 Podsumowanie

Przedstawione wyniki eksperymentów wskazują na dużą skuteczność opracowanych algorytmów generacji kodów AA-LDPC dla szerokiego spektrum parametrów

(długości bloku, stopy i dystrybucji stopni wierzchołków grafu). Uzyskane kody we wszystkich przypadkach są konkurencyjne, bądź nawet lepsze od odpowiadających im kodów LDPC pochodzących z bazy prof. MacKaya [64]. Przy tym kody z bazy [64] nie są zorientowane na implementację – struktura ich macierzy kontrolnych jest całkowicie nieuporządkowana, a implementacja dekodera szeregowo-równoległego praktycznie niemożliwa. Stanowi to o dużej przewadze kodów uzyskanych z wykorzystaniem opracowanych algorytmów, które zapewniając nie gorsze własności korekcyjne są jednocześnie efektywnie dekodowane sprzętowo.

Przedstawiono również porównanie własności utworzonych kodów z kodami AA-LDPC uzyskanymi innymi metodami. We wszystkich przypadkach kody utworzone za pomocą algorytmów 5.6 i 5.7 miały porównywalne, lepsze, bądź znacznie lepsze własności od kodów wybranych spośród wygenerowanych losowo oraz za pomocą metody „Girth conditioned”. Dowodzi to skuteczności opracowanych algorytmów ekspansji macierzy bazowej. Co prawda w pewnych przypadkach (dla stosunkowo dużych długości bloku) dobre rezultaty daje również metoda losowa, jednakże nie zmienia to faktu, że w ogólnym przypadku algorytmy 5.6 i 5.7 dają nie gorsze rezultaty niż inne metody. Spostrzeżenie to zostało również potwierdzone faktem uzyskania kodu konkurencyjnego do kodu wykorzystanego w standardzie IEEE 802.16e [41].

Podsumowując należy podkreślić, że eksperymenty potwierdziły znakomite własności kodów AA-LDPC uzyskanych za pomocą starannie opracowanych wieloetapowych algorytmów tworzenia macierzy kontrolnych.

6. Podsumowanie

Kody LDPC są jednymi z najlepszych znanych klas kodów wykorzystywanych do korekcji błędów w kanale telekomunikacyjnym. Skonstruowanie szybkich i skutecznych sprzętowych modułów kodera / dekodera jest zadaniem nietrywialnym. Istota pracy sprowadzała się więc do opracowania kodów LDPC, które zapewniając dobre własności korekcyjne umożliwiają efektywną implementację dekodera w dostępnych obecnie strukturach programowalnych. W pierwszej kolejności zaprezentowano zatem strukturę dekodera efektywnie implementowanego w strukturach programowalnych. Skupienie uwagi na module dekodera wynikało z faktu większej złożoności koncepcyjnej i obliczeniowej procesu dekodowania niż kodowania. Dodatkowo, w przypadku transmisji rozsiewczej (np. sygnału telewizji cyfrowej), koder występuje tylko w centralnym nadajniku, a odbiorników zawierających dekodery jest wiele, stąd też staranne zaprojektowanie dekodera jest istotniejsze. Efektywna implementacja sprzętowa dekodera narzuca pewne ograniczenia na strukturę macierzy kontrolnej kodu. W pracy przedstawiono precyzyjną definicję owych ograniczeń, a kody o macierzy kontrolnej spełniającej tą definicję nazywano kodami AA-LDPC.

Drugim, zasadniczym elementem pracy było opracowanie algorytmów generacji kodów AA-LDPC, które są konkurencyjne pod względem możliwości korekcji błędów do znanych kodów LDPC. Do weryfikacji skuteczności opracowanych algorytmów posłużyły przeprowadzone eksperymenty. Wyselekcjonowany został w ten sposób najlepszy spośród badanych algorytmów.

Realizacja dwóch głównych celów pracy, tzn. specyfikacja dekodera sprzętowego oraz opracowanie algorytmów tworzenia „dobrych” kodów AA-LDPC, została omówiona odpowiednio w rozdziałach 4 i 5.

W rozdziale 4 przedstawiono realizację pierwszych czterech zadań cząstkowych. Podstawowe efekty zaprezentowanych tam prac są następujące:

- opracowano w języku VHDL, przenaszalną na różne platformy sprzętowe specyfikację uniwersalnego, konfigurowalnego modułu dekodera regularnych, jak i nieregularnych kodów AA-LDPC,
- opracowano sposoby automatycznego dostosowania syntezowalnego opisu mo-

dułu dekodera do kodu o danej macierzy kontrolnej,

- stworzono środowisko testowania i symulacji systemów kodowania LDPC, oparte na implementacji dekodera w układzie FPGA firmy Xilinx oraz modelach pozostałych elementów systemu transmisyjnego (koder, modulator, demodulator i kanał transmisyjny) w środowisku MATLAB.

Zaprojektowany dekodery jest konkurencyjny ze względu na przepustowość oraz wymagane zasoby strukturalne do rozwiązań prezentowanych w literaturze.

Opracowanie efektywnego modułu dekodera sprzętowego wymagało rozwiązania szeregu nietrywialnych problemów. Jednym z owych problemów były błędy obliczeń wynikające z reprezentacji wiadomości w postaci liczb stałoprzecinkowych o silnie ograniczonej precyzji. Całkowicie oryginalnymi elementami pracy są przedstawiona analiza błędów obciążenia wartości wiadomości przesyłanych pomiędzy jednostkami obliczeniowymi w formacie stałoprzecinkowym o silnie ograniczonej długości słowa oraz zaproponowane modyfikacje układu dekodera pozwalające na zmniejszenie owych błędów. Wynikiem zaproponowanych rozwiązań jest znacząca poprawa możliwości korekcyjnych systemu, szczególnie dla dużych wartości stosunku sygnału do szumu w kanale.

Zastosowanie przetwarzania potokowego umożliwiło uzyskanie wysokiej częstotliwości zegarowej modułu dekodera. Jednakże dla pełnego wykorzystania zalet przetwarzania potokowego (tzn. znaczącego zwiększenia przepustowości) konieczne okazało się opracowanie algorytmu modyfikacji macierzy kontrolnej kodu prowadzącej do minimalizacji czasów bezczynności dekodera. Analiza wpływu struktury macierzy kontrolnej na czasy bezczynności doprowadziła do zaproponowania odpowiedniej metody optymalizacji macierzy. Opracowany algorytm cechuje się dużą skutecznością, pozwalając w wielu przypadkach na całkowite wyeliminowanie cykli bezczynności elementów dekodera.

Cennym elementem pracy jest również przeprowadzona analiza i porównanie własności systemów kodowania LDPC w warunkach dekodowania o ograniczonej precyzji. Analiza przepustowości, wymaganych zasobów układu programowalnego oraz bitowej stopy błędów systemu transmisji pozwala na wybór algorytmów dekodowania i ich charakterystycznych parametrów gwarantujących odpowiednie własności systemu transmisji.

Druga część pracy dotyczyła algorytmów tworzenia „dobrych” kodów AA-LDPC (rozdział 5), a podstawowe jej efekty są następujące:

- opracowano kompleksowe algorytmy generacji kodów AA-LDPC, które obejmują:

- wyznaczanie optymalnych dystrybucji stopni węzłów grafu Tannera dla nieregularnych kodów AA-LDPC,
 - generację macierzy bazowej kodu AA-LDPC, z uwzględnieniem ograniczeń implementacyjnych kodera oraz wspomnianej optymalizacji dla minimalizacji czasów bezczynności dekodera,
 - ekspansję macierzy bazowej; zaproponowano algorytmy ekspansji umożliwiające minimalizację liczby „szkodliwych” struktur w grafie Tannera kodu mających negatywny wpływ na własności korekcyjne kodu,
- przeprowadzono badania eksperymentalne własności kodów wygenerowanych z wykorzystaniem opracowanych metod oraz porównano je z kodami AA-LDPC wygenerowanymi pewnymi metodami odniesienia, jak również znanymi kodami LDPC pochodzącymi z bazy prof. MacKaya oraz kodami zamieszczonymi w specyfikacjach standardów transmisji danych; macierze kontrolne uzyskanych kodów zamieszczone są na stronie internetowej autora [101].

Dla uzyskania kodów o dobrych własnościach konieczne było zastosowanie odpowiednich algorytmów na wszystkich etapach tworzenia kodu. Szczególnie istotny jest etap ekspansji macierzy bazowej, dla którego opracowano całkowicie oryginalne algorytmy. Dołożono starań, aby opracowane algorytmy były uniwersalne, tzn. umożliwiały tworzenie kodów regularnych, jak i nieregularnych, o dowolnej długości bloku N , stopie kodu R oraz rozmiarze podmacierzy P .

Przedstawiono szereg eksperymentów dla kodów o różnych parametrach. Opierając się na uzyskanych wynikach wyselekcjonowano najlepszy spośród zaproponowanych algorytmów tworzenia macierzy kontrolnej. Za pomocą tego algorytmu, we wszystkich przypadkach uzyskano kody AA-LDPC nie gorsze ze względu na ich możliwości korekcyjne niż odpowiadające im inne znane z materiałów źródłowych kody LDPC. W związku z tym można stwierdzić, iż **istnieje możliwość generowania macierzy kontrolnych kodów LDPC efektywnie dekodowanych w strukturach programowalnych, konkurencyjnych do znanych kodów LDPC pod względem własności korekcyjnych**. Tym samym tezę pracy można uznać za udowodnioną.

Do najważniejszych oryginalnych efektów pracy należy zaliczyć: opracowanie implementacji dekodera kodów LDPC w strukturach programowalnych; analizę wpływu błędów obcięcia wartości wiadomości o silnie ograniczonej długości słowa na proces dekodowania oraz zaproponowanie modyfikacji układu dekodera pozwalających na zmniejszenie owych błędów; opracowanie algorytmu optymalizacji macierzy

kontrolnej kodu prowadzącej do minimalizacji czasów bezczynności dekodera z przetwarzaniem potokowym; przeprowadzenie szeregu eksperymentów umożliwiających porównanie własności korekcyjnych systemu wykorzystującego różne rozwiązania dekoderek; opracowanie wieloetapowego algorytmu tworzenia macierzy kontrolnej „dobrych” kodów LDPC efektywnie dekodowanych sprzętowo; utworzenie za pomocą opracowanych algorytmów szerokiego zbioru kodów i eksperymentalne wyznaczenie ich możliwości korekcyjnych.

Niniejsza praca zamyka pewien etap badań nad kodami AA-LDPC. Należy podkreślić, że wszystkie opracowane algorytmy zintegrowano w narzędziu programowym, które umożliwia szybkie przeprowadzenie wszystkich etapów procesu generacji „dobrego” kodu AA-LDPC oraz natychmiastowe przeprowadzenie symulacji z wykorzystaniem dekodera implementowanego w układzie typu FPGA. Efektem pracy jest więc nie tylko implementacja uniwersalnego dekodera i opracowanie algorytmów tworzenia kodów, ale również stanowisko uruchomieniowo-symulacyjne pozwalające na sprawne przeprowadzenie szeregu dalszych prac. Wymienione poniżej pomysły stanowią proponowane kierunki dalszych badań nad systemami transmisji wykorzystującymi kodowanie LDPC.

- Dla celu dalszego zwiększenia przepustowości możliwe wydaje się opracowanie dekodera kaskadowego, tzn. składającego się z pewnej liczby modułów dekoderek o miękkich decyzjach, gdzie każdy z modułów realizuje tylko pewną (niewielką) liczbę iteracji dekodowania. Implementacja takiego dekodera oraz zaproponowanie odpowiedniej metody doboru algorytmów kolejnych dekoderek w kaskadzie wydaje się być interesującym problemem.
- Dla kodów o stosunkowo niewielkich długościach bloków, możliwe jest zwiększenie możliwości korekcyjnych poprzez zastosowanie kodów LDPC definiowanych na ciele $GF(q)$, $q > 2$ (nazywanych również kodami $GF(q)$ -LDPC, [20, 21]). Wykorzystanie tego typu kodów wydaje się w wielu wypadkach wartym rozważenia rozwiązaniem. Literatura dotycząca sprzętowych implementacji dekodera tego typu kodów jest bardzo uboga. Pożądane byłoby zatem przystosowanie implementacji dekodera do tej klasy kodów.
- Kolejnym etapem prac mogłoby być rozwinięcie opracowanych metod generacji kodów na klasę $GF(q)$ -LDPC: zaproponowanie definicji kodu $GF(q)$ -LDPC zorientowanego na implementację oraz opracowanie odpowiednich algorytmów tworzenia tego typu kodów.

- Cenne byłoby również rozważenie możliwości implementacji sprzętowego modułu dekodera zintegrowanego z innymi elementami systemu transmisji, w szczególności układami demodulacji i korekcji charakterystyki kanału (konceptje zaprezentowane np. w pracy [52]).

Liczne doświadczenia zdobyte w trakcie realizacji niniejszej pracy są dla autora niezwykle cenne i stanowią będą motywację do kontynuowania prac w dziedzinie będącej jedną z podwalin rozwoju współczesnych środków komunikacji elektronicznej.

Bibliografia

- [1] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, „Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate”, *IEEE Transactions on Information Theory*, 20, No. 2, pp. 284–287, March 1974.
- [2] L. Bazzi, T. J. Richardson and R. L. Urbanke, „Exact Thresholds and Optimal Codes for the Binary-Symmetric Channel and Gallager’s Decoding Algorithm A”, *IEEE Transactions on Information Theory*, 50, No. 9, pp. 2010–2021, September 2004.
- [3] E. N. Berlekamp, R. J. McEliece and H. C. A. Van Tilborg, „On the Inherent Intractability of Certain Coding Problems”, *IEEE Transactions on Information Theory*, IT-24, No. 3, pp. 384–386, May 1978.
- [4] A. J. Blanksby and C. J. Howland, „A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder”, *IEEE Journal of Solid-State Circuits*, 37, No. 3, pp. 404–412, March 2002.
- [5] A. Byrne, E. M. Popovici and M. E. O’Sullivan, „Versatile Architectures for Decoding a Class of LDPC Codes”, in *European Conference on Circuit Theory and Design*, 3, pp. 269–272, Cork, Ireland, August, 2005.
- [6] J. Campello, D. S. Modha and S. Rajagopalan, „Designing LDPC Codes Using Bit-Filling”, in *IEEE International Conference on Communications*, 1, pp. 55–59, Helsinki, Finland, June, 2001.
- [7] E. Cavus, C. L. Haymes and B. Daneshrad, „An IS Simulation Technique for Very Low BER Performance Evaluation of LDPC Codes”, in *IEEE International Conference on Communications*, pp. 1095–1100, Istanbul, Turkey, June, 2006.
- [8] H. Chen and Z. Cao, „A Modified PEG Algorithm for Construction of LDPC Codes with Strictly Concentrated Check-Node Degree Distributions”, in *IEEE*

- Wireless Communications and Networking Conference*, pp. 564–568, Kowloon, China, March, 2007.
- [9] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier and X.-Y. Hu, „Reduced-Complexity Decoding of LDPC Codes”, *IEEE Transactions on Communications*, 53, No. 8, pp. 1288–1299, August 2005.
- [10] J. Chen and M. P. C. Fossorier, „Density Evolution for BP-Based Decoding Algorithms of LDPC Codes and Their Quantized Versions”, in *IEEE Globecom*, pp. 1378–1382, Taipei, Taiwan, August, 2002.
- [11] J. Chen and M. P. C. Fossorier, „Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes”, *IEEE Transactions on Communications*, 50, No. 3, pp. 406–414, March 2002.
- [12] J. Chen and M. P. C. Fossorier, „Density Evolution for Two Improved BP-Based Decoding Algorithms of LDPC Codes”, *IEEE Communications Letters*, 6, No. 5, pp. 208–210, May 2002.
- [13] Y. Chen and D. E. Hocevar, „A FPGA and ASIC Implementation of Rate 1/2, 8088-b Irregular Low Density Parity Check Decoder”, in *IEEE Globecom*, pp. 113–117, San Francisco, USA, December, 2003.
- [14] S. K. Chilappagari, S. Sankaranarayanan and B. Vasic, „Error Floors of LDPC Codes on the Binary Symmetric Channel”, in *IEEE International Conference on Communications*, pp. 1089–1094, Istanbul, Turkey, June, 2006.
- [15] J. Chojcan and J. Rutkowski, *Zbiór zadań z teorii informacji i kodowania*, Wydawnictwo Politechniki Śląskiej, Gliwice, 1997.
- [16] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson and R. L. Urbanke, „On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit”, *IEEE Communications Letters*, 5, No. 2, pp. 58–60, February 2001.
- [17] S.-Y. Chung, T. J. Richardson and R. L. Urbanke, „Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 657–670, February 2001.
- [18] C. A. Cole, S. G. Wilson, E. K. Hall and T. R. Giallorenzi, „Analysis and Design of Moderate Length Regular LDPC Codes with Low Error Floors”, in

- 40th Annual Conference on Information Sciences and Systems*, pp. 823–828, Princeton, USA, March, 2006.
- [19] C. A. Cole, S. G. Wilson, E. K. Hall and T. R. Giallorenzi, „A general method for finding low error rates of ldpc codes”, May 2006, <http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0605051>.
- [20] M. C. Davey and D. J. C. MacKay, „Low-Density Parity Check Codes over $GF(q)$ ”, *IEEE Communications Letters*, 2, No. 6, pp. 165–167, June 1998.
- [21] D. Declercq and M. Fossorier, „Decoding Algorithms for Nonbinary LDPC Codes Over $GF(q)$ ”, *IEEE Transactions on Communications*, 55, No. 4, pp. 633–643, April 2007.
- [22] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson and R. L. Urbanke, „Finite-Length Analysis of Low-Density Parity-Check Codes on the Binary Erasure Channel”, *IEEE Transactions on Information Theory*, 48, No. 6, pp. 1570–1579, June 2002.
- [23] J. Dielissen, A. Hekstra and V. Berg, „Low Cost LDPC Decoder for DVB-S2”, in *DATE 06 - Design, Automation and Test in Europe Conference*, pp. 1–6, Munich, Germany, March, 2006.
- [24] R. Diestel, *Graph Theory*, Springer-Verlag, Berlin, 2006.
- [25] S. Ding, Z. Yang, J. Song, C. Pan and J. Wang, „Combinatorial Construction of Low-Density Parity-Check Codes for Short Block Length and High Rate Applications”, in *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4, Wuhan, China, September, 2006.
- [26] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright and B. Nikolic, „Analysis of Absorbing Sets for Array-Based LDPC Codes”, in *IEEE International Conference on Communications*, pp. 6261–6268, Glasgow, Scotland, June, 2007.
- [27] E. Eleftheriou, T. Mittelholzer and A. Dholakia, „Reduced-Complexity Decoding Algorithm for Low-Density Parity-Check Codes”, *Electronics Letters*, 37, No. 2, pp. 102–104, January 2001.

- [28] ETSI Standard: EN 302 307 v1.1.1, *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications*, 2005.
- [29] G. D. Forney, Jr., „Codes on Graphs: Normal Realizations”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 520–548, February 2001.
- [30] R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.
- [31] R. G. Gallager, „Low-Density Parity-Check Codes”, *IRE Transactions on Information Theory*, IT-8, pp. 21–28, January 1962.
- [32] J. Hagenauer, E. Offer and L. Papke, „Iterative Decoding of Binary Block and Convolutional Codes”, *IEEE Transactions on Information Theory*, 42, No. 2, pp. 329–445, March 1996.
- [33] S. Haykin, *Systemy telekomunikacyjne*, Wydawnictwa Komunikacji i Łączności, Warszawa, 1998.
- [34] D. E. Hocevar, „LDPC Code Construction with Flexible Hardware Implementation”, in *ICC 2003 - IEEE International Conference on Communications*, 4, pp. 2708–2712, Anchorage, Alaska, May, 2003.
- [35] S. L. Howard, C. Schlegel and V. C. Gaudet, „Degree-Matched Check Node Decoding for Regular and Irregular LDPCs”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53, No. 10, pp. 1054–1058, October 2006.
- [36] C. Howland and A. Blanksby, „Parallel Decoding Architectures for Low Density Parity Check Codes”, in *ISCAS 2001 - IEEE International Symposium on Circuits and Systems*, pp. 742–745, Sydney, Australia, May, 2001.
- [37] X.-Y. Hu, E. Eleftheriou and D. M. Arnold, „Regular and Irregular Progressive Edge-Growth Tanner Graphs”, *IEEE Transactions on Information Theory*, 51, No. 1, pp. 386–398, January 2005.
- [38] X.-Y. Hu, E. Eleftheriou and D. M. Arnold, „Progressive Edge-Growth Tanner Graphs”, in *IEEE Globecom*, pp. 995–2001, San Antonio, Texas, USA, November, 2001.

- [39] X.-Y. Hu, E. Eleftheriou, D. M. Arnold and A. Dholakia, „Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes”, in *IEEE Globecom*, pp. 1036–1036E, San Antonio, Texas, USA, November, 2001.
- [40] IEEE Standard: IEEE P802.11n=D10, *Draft IEEE Standard for Local Metropolitan Networks – Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC), and Physical Layer (PHY) specifications: Enhancements for Higher Throughput*, March 2006.
- [41] IEEE Standard: IEEE Std 802.16e-2005, *IEEE Standard for Local and Metropolitan Area Networks. Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, February 2006.
- [42] J. Izydorzycy, M. Kucharczyk and J. Rutkowski, „Kody LDPC – konkurencja dla turbokodów”, *Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne*, Rocznik LXXIX, nr 2-3, pp. 65–72, 2006.
- [43] M. Jiang, C. Zhao, L. Zhang and E. Xu, „Adaptive Offset Min-Sum Algorithm for Low-Density Parity Check Codes”, *IEEE Communications Letters*, 10, No. 6, pp. 483–485, June 2006.
- [44] G. Kalai and N. Linial, „On the Distance Distribution of Codes”, *IEEE Transactions on Information Theory*, 41, No. 5, pp. 1467–1472, September 1995.
- [45] F. Kienle, T. Brack and N. Wehn, „A Synthesizable IP Core for DVB-S2 LDPC Code Decoding”, in *DATE 05 - Design, Automation and Test in Europe Conference*, 3, pp. 100–105, Munich, Germany, March, 2005.
- [46] S. Kim, G. E. Sobelman and J. Moon, „Parallel VLSI Architectures for a Class of LDPC Codes”, in *ISCAS 2002 - IEEE International Symposium on Circuits and Systems*, 2, pp. 93–96, Scottsdale, Arizona, USA, May, 2002.
- [47] M. C. Klin, R. Poschel and K. Rosenbaum, *Algebra stosowana dla matematyków i informatyków*, Wydawnictwa Naukowo Techniczne, Warszawa, 1992.
- [48] Y. J. Ko and J. H. Kim, „Girth Conditioning for Construction of Short Block Length Irregular LDPC Codes”, *Electronics Letters*, 40, No. 3, pp. 187–188, February 2004.

- [49] Y. Kou, S. Lin and M. P. C. Fossorier, „Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results”, *IEEE Transactions on Information Theory*, 47, No. 7, pp. 2711–2736, November 2001.
- [50] F. R. Kschischang, B. J. Frey and H.-A. Loeliger, „Factor Graphs and the Sum-Product Algorithm”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 498–519, February 2001.
- [51] M. K. Ku, L. H. Sheng and Y. H. Chien, „Code Design and Decoder Implementation of Low Density Parity Check Code”, in *Emerging Information Technology Conference*, p. 3, Dallas, USA, August, 2005.
- [52] M. Kucharczyk, *Kodowanie z korekcją błędów w systemach transmisyjnych z modulacją wielotonową*, rozprawa doktorska, Wydział AEiI, Politechnika Śląska, Gliwice, Polska, 2006.
- [53] J. L. Kulikowski, *Zarys teorii grafów*, Państwowe Wydawnictwo Naukowe, Warszawa, 1986.
- [54] S. Landner and O. Milenkovic, „Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes”, in *International Conference on Wireless Networks, Communications and Mobile Computing*, pp. 630–635, Maui, USA, August, 2005.
- [55] D. Levin, E. Sharon and S. Litsyn, „Lazy Scheduling for LDPC Decoding”, *IEEE Communication Letters*, 11, No. 1, pp. 70–72, January 2007.
- [56] B. Levine, R. R. Taylor and H. Schmit, „Implementation of Near Shannon Limit Error-Correcting Codes Using Reconfigurable Hardware”, in *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 217–226, Napa Valley, USA, April, 2000.
- [57] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications, 2nd Edition*, Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2004.
- [58] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi and D. A. Spielman, „Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation”, in *ISIT 98 - IEEE International Symposium on Information Theory*, p. 171, Cambridge, USA, August, 1998.

- [59] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi and D. A. Spielman, „Improved Low-Density Parity-Check Codes Using Irregular Graphs”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 585–598, February 2001.
- [60] D. J. C. MacKay, „Good Error-Correcting Codes Based on Very Sparse Matrices”, *IEEE Transactions on Information Theory*, 45, No. 2, pp. 399–431, March 1999.
- [61] D. J. C. MacKay and R. M. Neal, „Good Codes Based on Very Sparse Matrices”, in *Cryptography and Coding 5th IMA Conference*, pp. 100–111, Berlin, Germany, December, 1995.
- [62] D. J. C. MacKay and M. S. Postol, „Weaknesses of Margulis and Ramanujan–Margulis Low-Density Parity-Check Codes”, *Electronic Notes in Theoretical Computer Science*, 74, pp. 1–8, 2003.
- [63] D. J. C. MacKay, S. T. Wilson and M. C. Davey, „Comparison of Constructions of Irregular Gallager Codes”, in *Proceedings of the 1998 Allerton Conference on Communication, Control, and Computing*, September, 1998.
- [64] David MacKay’s Gallager code resources, <http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html>.
- [65] M. M. Mansour, „A Turbo-Decoding Message-Passing Algorithm for Sparse Parity-Check Matrix Codes”, *IEEE Transactions on Signal Processing*, 54, No. 11, pp. 4376–4392, November 2006.
- [66] M. M. Mansour and N. R. Shanbhag, „Turbo Decoder Architectures for Low-Density Parity-Check Codes”, in *IEEE Globecom*, pp. 1383–1388, Taipei, Taiwan, August, 2002.
- [67] M. M. Mansour and N. R. Shanbhag, „High Throughput LDPC Decoders”, *IEEE Transactions on Very Large Scale Integration Systems*, 11, No. 6, pp. 976–996, December 2003.
- [68] M. M. Mansour and N. R. Shanbhag, „A 640-Mb/s 2048-Bit Programmable LDPC Decoder Chip”, *IEEE Journal of Solid-State Circuits*, 41, No. 3, pp. 684–698, March 2006.
- [69] Y. Mao and A. H. Banihashemi, „A Heuristic Search for Good Low-Density Parity-Check Codes at Short Block Lengths”, in *IEEE International Conference on Communications*, 1, pp. 41–44, Helsinki, Finland, June 2001.

- [70] Y. Mao and A. H. Banihashemi, „Decoding Low-Density Parity-Check Codes with Probabilistic Scheduling”, *IEEE Communication Letters*, 5, No. 10, pp. 414–416, October 2001.
- [71] J. M. F. Moura, J. Lu and H. Zhang, „Structured Low-Density Parity-Check Codes”, *IEEE Signal Processing Magazine*, 12, No. 1, pp. 42–55, January 2004.
- [72] J. M. F. Moura and H. Zhang, „The Design of Structured Regular LDPC Codes with Large Girth”, in *IEEE Globecom*, pp. 4022–4027, San Francisco, CA, USA, December, 2003.
- [73] R. M. Neal, „Sparse Matrix Methods and Probabilistic Inference Algorithms, Part 1: Faster Encoding for Low-Density Parity-Check Codes Using Sparse Matrix Methods”, in *IMA Summer Program: Codes, Systems and Graphical Models, Week 1: Codes on Graphs and Iterative Decoding*, Minneapolis, USA, August, 1999.
- [74] S. Olcer, „Decoder Architecture for Array-Code-Based LDPC Codes”, in *IEEE Globecom*, pp. 2046–2050, San Francisco, USA, December, 2003.
- [75] M. E. O’Sullivan, „Construction of LDPC Codes from Affine Permutation Matrices”, in *40th annual Allerton Conference of Communication, Control and Computing*, Illinois, USA, 2002.
- [76] M. E. O’Sullivan, „Algebraic Construction of Sparse Matrices with Large Girth”, *IEEE Transactions on Information Theory*, 52, No. 2, pp. 718–727, February 2006.
- [77] L. Ping and W. K. Leung, „Decoding Low Density Parity Check Codes with Finite Quantization Bits”, *IEEE Communication Letters*, 4, No. 2, pp. 62–64, February 2000.
- [78] J. G. Proakis, *Digital Communications, 4th Edition*, McGraw-Hill, New York, 2000.
- [79] A. Ramamoorthy and R. Wesel, „Construction of Short Block Length Irregular Low-Density Parity-Check Codes”, in *IEEE International Conference on Communications*, pp. 410–414, Paris, France, June, 2004.
- [80] T. Richardson, „Error Floors of LDPC Codes”, in *Proc. 41st Annual Allerton Conference on Communications, Control, and Computing*, pp. 1426–1435, Monticello, USA, October, 2003.

- [81] T. J. Richardson, M. A. Shokrollahi and R. L. Urbanke, „Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 619–637, February 2001.
- [82] T. J. Richardson and R. L. Urbanke, „Efficient Encoding of Low-Density Parity-Check Codes”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 638–656, February 2001.
- [83] T. J. Richardson and R. L. Urbanke, „The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding”, *IEEE Transactions on Information Theory*, 47, No. 2, pp. 599–618, February 2001.
- [84] G. Richter and A. Hof, „On a Construction Method of Irregular LDPC Codes Without Small Stopping Sets”, in *IEEE International Conference on Communications*, pp. 1119–1124, Istanbul, Turkey, June, 2006.
- [85] G. Richter, G. Schmidt, M. Bossert and E. Costa, „Optimization of a Reduced-Complexity Decoding Algorithm for LDPC Codes by Density Evolution”, in *IEEE International Conference on Communications*, pp. 642–646, May, 2005.
- [86] P. Robertson, E. Vilebrun and P. Hoher, „A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain”, in *IEEE International Conference on Communications*, pp. 1009–1013, Seattle, USA, June, 1995.
- [87] J. Rosenthal and P. O. Vontobel, „Construction of LDPC codes using Ramanujan graphs and ideas from Margulis”, in *Proceedings of the 38th Allerton Conference on Communications, Control, and Computing*, pp. 248–257, Monticello, USA, October, 2000.
- [88] M. Rovini, N. E. L’Insalata, F. Rossi and L. Fanucci, „VLSI Design of a High-Throughput Multi-Rate Decoder for Structured LDPC Codes”, in *DSD 2005 - Euromicro Conference on Digital System Design*, pp. 202–209, Porto, Portugal, August, 2005.
- [89] C. E. Shannon, „Communication in the Presence of Noise”, *Proceedings of the I.R.E.*, 37, pp. 10–21, January 1949.
- [90] C. E. Shannon, „A Mathematical Theory of Communication”, *Bell Systems Technical Journal*, 27, pp. 379–423, 623–656, July, October 1948.

- [91] E. Sharon and S. Litsyn, „A Method for Constructing LDPC Codes with Low Error Floor”, in *IEEE International Symposium on Information Theory*, pp. 2569–2573, Seattle, Washington, USA, July, 2006.
- [92] E. Sharon, S. Litsyn and J. Goldberger, „Efficient Serial Message-Passing Schedules for LDPC Decoding”, *IEEE Transactions on Information Theory*, 53, No. 11, pp. 4076–4091, November 2007.
- [93] E. Sharon, S. Litsyn and J. Goldberger, „An Efficient Message-Passing Schedule for LDPC Decoding”, in *23rd IEEE Convention of Electrical and Electronics Engineers in Israel*, pp. 223–226, Tel-Aviv, Israel, September, 2004.
- [94] X. Y. Shih, C. Z. Zhan, C. H. Lin and A. Y. Wu, „An 8.29 mm² 52mW Multi-Mode LDPC Decoder Design for Mobile WiMAX System in 0.13 μ m CMOS Process”, *IEEE Journal of Solid-State Circuits*, 43, No. 3, pp. 672–683, March 2008.
- [95] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga and G. Satoshi, „Partially-Parallel LDPC Decoder Based on High-Efficiency Message-Passing Algorithm”, in *ICCD 2005 - International Conference on Computer Design*, pp. 503–510, San Jose, California, USA, October, 2005.
- [96] M. Sipser and D. A. Spielman, „Expander Codes”, *IEEE Transactions on Information Theory*, 42, No. 6, pp. 1710–1722, November 1996.
- [97] W. Sułek, „Configurable AA-LDPC Decoder Implementation”, in *Signal Processing Symposium*, pp. 1–6, Jachranka, Poland, May, 2007.
- [98] W. Sułek, „Simulation of LDPC Codes Using Programmable Logic Devices”, in *IWCIT 2007 - International Workshop Control and Information Technology*, pp. 177–182, Ostrava, Czech Republic, September, 2007.
- [99] W. Sułek, „Implementacja modułu sprzętowego dekodera kodów AA-LDPC”, *Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne*, pp. 1229–1240, September 2008.
- [100] W. Sułek and D. Kania, „Code Construction Algorithm for Architecture Aware LDPC Codes with Low Error Floor”, in *SIBIRCON 2008 - IEEE Region 8 International Conference on Computational Technologies in Electrical and Electronics Engineering*, pp. 1–6, Novosibirsk, Russia, July, 2008.

- [101] Kody LDPC, baza Wojciecha Sułka, <http://iele.polsl.pl/~wojsu>.
- [102] L. Sun and V. Kumar, „Field Programmable Gate Array Implementation of a Generalized Decoder for Structured Low-Density Parity Check Codes”, in *IEEE International Conference on Field-Programmable Technology*, pp. 17–24, Brisbane, Australia, December, 2004.
- [103] L. Sun, H. Song, Z. Keirn and K. Vijaya, „Field Programmable Gate Array (FPGA) for Iterative Code Evaluation”, *IEEE Transactions on Magnetics*, 42, No. 2, pp. 226–231, February 2006.
- [104] Y. Sun, M. Karkooti and J. R. Cavallaro, „VLSI Decoder Architecture for High Throughput, Variable Block-size and Multi-rate LDPC Codes”, in *ISCAS 2007 - IEEE International Symposium on Circuits and Systems*, pp. 2104–2107, New Orleans, USA, May, 2007.
- [105] R. M. Tanner, „A Recursive Approach to Low Complexity Codes”, *IEEE Transactions on Information Theory*, IT-27, pp. 533–547, September 1981.
- [106] J. Thorpe, „Design of LDPC Graphs for Hardware Implementation”, in *IEEE International Symposium on Information Theory*, p. 483, Lausanne, Switzerland, July, 2002.
- [107] T. Tian, C. Jones, J. D. Villasenor and W. R. D., „Construction of Irregular LDPC Codes with Low Error Floor”, in *IEEE International Conference on Communications*, 5, pp. 3125–3129, Anchorage, Alaska, May, 2003.
- [108] T. Tian, C. Jones, J. D. Villasenor and R. D. Wesel, „Selective Avoidance of Cycles in Irregular LDPC Code Construction”, *IEEE Transactions on Communications*, 52, No. 8, pp. 1242–1247, August 2004.
- [109] A. Vardy, „The Intractability of Computing the Minimum Distance of a Code”, *IEEE Transactions on Information Theory*, 43, No. 6, pp. 1757–1766, November 1997.
- [110] B. Vasic, E. M. Kurtas and A. V. Kuznetsov, „Kirkman Systems and Their Application in Perpendicular Magnetic Recording”, *IEEE Transactions on Magnetics*, 38, No. 4, pp. 1705–1710, July 2002.
- [111] F. Verdier and D. Declercq, „A Low-Cost Parallel Scalable FPGA Architecture for Regular and Irregular LDPC Decoding”, *IEEE Transactions on Communications*, 54, No. 7, pp. 1215–1223, July 2006.

- [112] Z. Wang and Z. Cui, „Low-Complexity High-Speed Decoder Design for Quasi-Cyclic LDPC Codes”, *IEEE Transactions on Very Large Scale Integration Systems*, 15, No. 1, pp. 104–114, January 2007.
- [113] S. B. Wicker and S. Kim, *Fundamentals of Codes, Graphs and Iterative Decoding*, Kluwer Academic Publishers, London, 2003.
- [114] R. J. Wilson, *Wprowadzenie do teorii grafów*, Wydawnictwo Naukowe PWN, Warszawa, 1998.
- [115] H. Xiao and A. H. Banihashemi, „Improved Progressive-Edge-Growth (PEG) Construction of Irregular LDPC Codes”, *IEEE Communications Letters*, 8, No. 12, pp. 715–717, December 2004.
- [116] F. Xiong, *Digital Modulation Techniques*, Artech House, Inc., 685 Canton Street, Norwood, MA 02062, 2000.
- [117] L. Yang, H. Liu and C. J. R. Shi, „Code Construction and FPGA Implementation of a Low-Error-Floor Multi-Rate Low-Density Parity-Check Code Decoder”, *IEEE Transactions on Circuits and Systems I: Regular Papers*, 53, No. 4, pp. 892–904, April 2006.
- [118] L. Yang, H. Liu and C. J. R. Shi, „VLSI Implementation of a Low-Error-Floor and Capacity-Approaching Low-Density Parity-Check Code Decoder with Multi-Rate Capacity”, in *IEEE Globecom*, pp. 1261–1266, St Louis, USA, November, 2005.
- [119] M. R. Yazdani, S. Hemati and A. H. Banihashemi, „Improving Belief Propagation on Graphs With Cycles”, *IEEE Communications Letters*, 8, No. 1, pp. 57–59, January 2004.
- [120] F. Zarkeshvari and A. H. Banihashemi, „On Implementation of Min-Sum Algorithm for Decoding Low-Density Parity-Check (LDPC) Codes”, in *IEEE Globecom*, pp. 1349–1353, Taipei, Taiwan, August, 2002.
- [121] J. Zhang and M. P. C. Fossorier, „Shuffled Iterative Decoding”, *IEEE Transactions on Communications*, 53, No. 2, pp. 209–213, February 2005.
- [122] J. Zhang, M. P. C. Fossorier, D. Gu and J. Zhang, „Improved Min-Sum Decoding of LDPC Codes Using 2-Dimensional Normalization”, in *IEEE Globecom*, pp. 1187–1192, St Louis, USA, November, 2005.

-
- [123] T. Zhang and K. Parhi, „An FPGA Implementation of (3,6)-Regular Low-Density Parity-Check Code Decoder”, *EURASIP Journal on Applied Signal Processing*, 2003, No. 6, pp. 530–542, May 2003.
- [124] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam and M. Wainwright, „Investigation of Error Floors of Structured Low-Density Parity-Check Codes by Hardware Emulation”, in *IEEE Globecom*, San Francisco, USA, November, 2006.
- [125] J. Zhao, F. Zarkeshvari and A. H. Banihashemi, „On Implementation of Min-Sum Algorithm and Its Modifications for Decoding Low-Density Parity-Check (LDPC) Codes”, *IEEE Transactions on Communications*, 53, No. 4, pp. 549–554, April 2005.
- [126] H. Zhong and T. Zhang, „Block-LDPC: A Practical LDPC Coding System Design Approach”, *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52, No. 4, pp. 766–775, April 2005.