AUTOMATYKA • 2005 • Tom 9 • Zeszyt 3

Ernest Jamro*, Kazimierz Wiatr*

Implementacja w układach FPGA silnie zrównoleglonej operacji *Look-Up Table***

1. Wprowadzenie

Przekodowanie Look-Up Table (LUT) [2] jest operacją stosunkową prostą do przeprowadzenia i często jest wykonywane w procesorze ogólnego przeznaczenia. Niestety w niektórych przypadkach transformacja LUT może wymagać sprzętowej akceleracji obliczeń [3]. Takim przypadkiem jest np. system detekcji twarzy poprzez sieci neuronowe [4]. W systemie tym na wejście sieci neuronowej podaje się fragment obrazu o rozmiarach 20×20 pikseli. Fragment (maska) obrazu jest kolejno przesuwany o 1 piksel po całym obrazie źródłowym. W szczególności dla obrazu wejściowego 512×512 pikseli należy przeprowadzić operacje na 492×492 osobnych fragmentach obrazu o rozmiarze 20×20. Obraz podawany na wejście sieci neuronowej powinien być poddany lokalnemu wyrównaniu histogramu. Jedną z operacji wchodzących w skład operacji wyrównywania histogramu jest operacja LUT. W konsekwencji, w celu przetworzenia pojedynczego obrazu 512×512 konieczne jest wykonanie $492\times492\times20\times20 \approx 10^8$ operacji LUT. Warto w tym miejscu podkreślić, że podana liczba dotyczy tylko oryginalnej skali. Dla systemu [3] wyżej wspomniane operacje muszą być również przeprowadzone na obrazie o kolejno zmniejszanej skali.

W konsekwencji, aby przyspieszyć wykonywanie operacji LUT, celowa stała się sprzętowa realizacja przekodowania LUT oraz jej silne zrównoleglenie. Niestety w przeciwieństwie do samej sieci neuronowej operacja LUT nie może być w prosty sposób zrównoleglona ze względu na konieczność zmiany zawartości pamięci LUT dla różnych fragmentów obrazu. Zmiana ta może odbywać się tylko w sposób sekwencyjny.

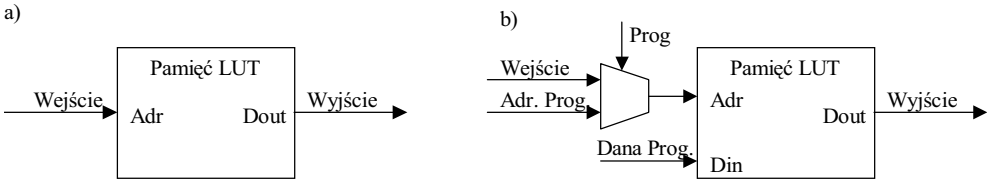
2. Sprzętowa realizacja operacji LUT

Operację LUT można w prosty sposób realizować sprzętowo. W tym celu należy na wejście adresowe pamięci podać daną wejściową, która po przekodowaniu jest otrzymywa-

* Akademia Górniczo-Hutnicza, Kraków, ACK Cyfronet AGH, Kraków; jamro/wiatr@agh.edu.pl

** Praca naukowa finansowana ze środków budżetowych na naukę jako projekt badawczy

na na wyjściu danych pamięci. Ilustruje to rysunek 1a. Rzeczywisty układ wymaga albo zastosowania pamięci ROM, dla której wartość pamięci się nie zmienia, lub też zastosowania pamięci RAM i dodatkowego interface'u umożliwiającego zmianę zawartości pamięci. W niniejszym artykule zostanie zastosowany sposób drugi, który został pokazany na rysunku 1b. Dodatkowy interface składa się głównie z multipleksera, który multipleksuje magistralę adresową: wybiera daną wejściową podczas normalnej pracy lub też adres programujący, w przypadku kiedy następuje programowanie pamięci LUT.



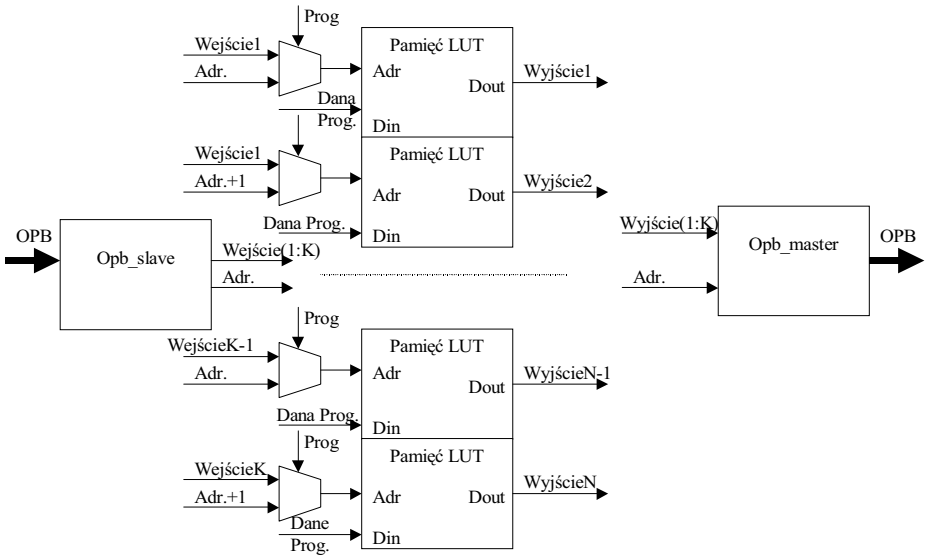
Rys. 1. Sprzętowa realizacja operacji LUT
Objaśnienia w tekście

W układach FPGA powyższa pamięć może być zaimplementowana w postaci pamięci blokowej BRAM. Na przykład w układach FPGA Virtex firmy Xilinx pamięć BRAM ma pojemność 4kb lub 18kb (Virtex2), dlatego operacja LUT dla danej wejściowej 8-bitowej i danej wyjściowej 8-bitowej może być zrealizowana za pomocą pojedynczej pamięci BRAM. Co więcej, w układach FPGA pamięć BRAM jest pamięcią 2-portową, dlatego aby lepiej wykorzystać pamięć BRAM, zalecane jest wykonywanie dwóch operacji LUT równolegle, np. na dwóch sąsiadujących ze sobą pikselach. Powoduje to podwojenie szybkości pracy układu bez wpływu na liczbę używanych pamięci BRAM.

3. Moduł LUT kompatybilny ze środowiskiem EDK

Przekodowanie LUT jest jedną z wielu operacji wykonywanych wewnątrz układu FPGA, dlatego konieczna stała się modułowa budowa całego systemu. W konsekwencji wykorzystano pakiet Embedded Development Kit (EDK) firmy Xilinx, który umożliwia łatwą budowę całego systemu z poszczególnych bloków. Jako magistralę łączącą poszczególne bloki wykorzystano magistralę On-chip Peripherial Bus (OPB) firmy IBM.

Aby moduł LUT działał poprawnie z pakietem EDK, konieczne stało się dostosowanie tego modułu do standardu magistrali OPB, co wymagało zaprojektowania specjalnych interface'ów o nazwie roboczej *opb_slave* i *opb_master*. Dane wejściowe są podawane przez interface *opb_slave*, który pracuje jako urządzenie podrzędne na magistrali OPB. Dane wyjściowe są transmitowane dalej poprzez interface *opb_master*, który pracuje jako urządzenie nadrzędne na magistrali OPB. Moduł dokonujący przekodowania LUT ze względu na jego kompatybilność z magistralą OPB będzie dalej nazywany jako *opb_lut.* Struktura blokowa moduł *opb_lut* została pokazana na rysunku 2.



Rys. 2. Struktura wewnętrzna modułu *opb_lut*

Moduł *opb_lut* oprócz inerface'ów *opb_master* i *opb_slave* składa się z wielu modułów pamięci LUT. Liczba równoległych pamięci LUT (oznaczona poprzez stałą *K* na rys. 2), podobnie jak wiele innych niewymienionych tutaj parametrów, jest określana w pakiecie EDK. W konsekwencji liczba operacji LUT wykonywanych w czasie pojedynczego taktu zegara może być łatwo zmieniona. Umożliwia to w łatwy sposób modyfikację całego systemu.

Tabela 1
Wykorzystanie zasobów FPGA przez moduł *opb_lut* dla: różnej szerokości magistrali danych magistrali wejściowej SOPB (SOPB_dwidth), magistrali wyjściowej MOPB_dwidth, liczby użytych pamięci LUT

SOPB Dwidth	# LUT = K	MOPB dwidth	# 4-input LUT	# przerzutników	# BRAM
8	1	8	102	33	1
16	1	16	186	61	1
16	2	16	125	33	1
32	2	32	249	62	1
32	4	32	163	38	2

W tym miejscu należy podkreślić, że równoległość stała się łatwa do zrealizowania, ponieważ szerokość danej wejściowej/wyjściowej pojedynczej pamięci LUT jest niewielka i wynosi z reguły 8 bitów. Z kolei magistrala OPB może mieć zmienną szerokość magistrali danych równą: 8, 16, 32 lub 64 bity. Wykorzystanie zasobów układu FPGA przez moduł *opb_lut* dla różnych parametrów został przedstawiony w tabeli 1. Moduły interface'ów

opb_master i *opb_slave* mają największy wpływ na ogólną powierzchnię całego modułu *opb_lut* (oczywiście z wyjątkiem pamięci BRAM). Jest to zauważalne w przypadku dokonywania konwersji szerokości magistrali danych.

3.1. Moduły *opb_master* i *opb_slave*

Największą trudnością podczas projektowania modułu *opb_lut* było zaprojektowanie modułów interface'ów *opb_master* i *opb_slave*. Moduły te są modułami uniwersalnymi i zostały również użyte przy projektowaniu innych modułów kompatybilnych z magistralą OPB. Moduły *opb_master* i *opb_slave* są silnie sparametryzowane (za pomocą słowa kluczowego *generic* w języku opisu sprzętu VHDL) i charakteryzują je następujące cechy:

- Programowalna wielkość wewnętrznej kolejki *First-In First-Out* (FIFO). Dla modułu FIFO użyto wbudowanych (w układach Virtex) rejestrów przesuwnych (SRL).
- Implementacja (lub nie) adresowania sekwencyjnego, co wydatnie przyspiesza transfer blokowy na magistrali OPB. Opcjonalnie możliwe jest automatyczne wykrywania adresowania sekwencyjnego tylko na podstawie porównywania następujących po sobie stanów magistrali adresowej. Opcja ta umożliwia łączenie poszczególnych transferów blokowych i dodatkowo przyspiesza transfer danych na magistrali OPB.
- Zoptymalizowanie przyznawania magistrali OPB ze względu na stan buforów FIFO oraz sparametryzowania liczby przesłań podczas pojedynczego transferu blokowego.
- Sparametryzowanie szerokości magistrali danych zarówno po stronie magistrali OPB, jak i po stronie użytkownika. Zaprojektowano układ umożliwiający wewnętrzną konwersję szerokości danych, która odbywa się praktycznie bez zaangażowania projektanta. Powyższa cecha umożliwia łatwą skalowalność szybkości działania poszczególnych modułu. Na przykład, dla modułu *opb_lut* możliwe jest bardzo proste użycie tylko pojedynczego 8-bitowego modułu pamięci LUT, pomimo podłączenia modułu *opb_lut* do magistrali OPB 32-bitowej. Dodatkowe użycie kolejki FIFO umożliwia szybkie przyjmowanie wielu transferów 32-bitowych w ramach pojedynczego transferu blokowego oraz powolne opróżnianie kolejki FIFO poprzez 8-bitowy interface użytkownika (pamięci LUT). W konsekwencji pamięć LUT może ciągle wykonywać operacje LUT, pomimo to że magistrala OPB jest współdzielona przez wiele urządzeń.
- Sparametryzowane opóźnienie potokowe pomiędzy modułem *opb_slave* i *opb_master*. Powyższa cecha powoduje, że w prosty sposób, bez dodatkowej logiki kontrolnej, można dodać rejestry potokowe w celu przyspieszania maksymalnej częstotliwości pracy logiki użytkownika. W praktyce wiąże się to z przesunięciem potokowym pomiędzy magistralą danych, na której wykonywane są operacje (np. przekodowanie LUT), a magistralą adresową, na której nie występuje opóźnienie. W przypadku gdyby nie było wspomnianej logiki, dodatkowe opóźnienie potokowe wymagałoby skomplikowanego sterowania sygnałami kontrolnymi gotowości do transmisji (sygnały OPB_select i OPB_xferAck na magistrali OPB) oraz zastosowania dodatkowych buforów FIFO.

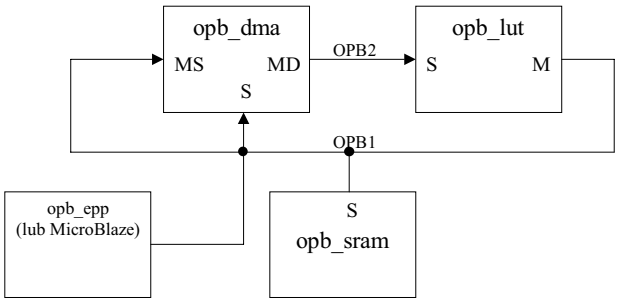
Warto w tym miejscu podkreślić, że silnie sparametryzowane moduły *opb_master* i *opb_slave* stanowią prawdziwe wyzwanie dla projektanta – ponieważ powyższe parametry są bardzo często od siebie uzależnione. Na przykład: konwersja szerokości magistrali danych wpływa na wielkość pamięci FIFO. Podobnie jest w przypadku adresowania sekwencyjnego oraz wielkości bufora FIFO, opóźnienia potokowego lub wielkości bufora FIFO. Ponadto poprzez zastosowanie tak wielu parametrów możliwe jest zbudowanie wielu różnych odbiegających od siebie modułów *opb_master* i *opb_slave*, które trzeba niezależnie przetestować. W celu przyspieszenia symulacji i testowania całego sytemu wykorzystano środowisko APSI [5].

4. System wykonujący operację LUT

Moduł *opb_lut* wymaga zastosowania innych modułów między innymi podających dane wejściowe i odbierających dane wyjściowe. Przykład takiego systemu jest pokazany na rysunku 3.

Składa się on z następujących dodatkowych modułów kompatybilnych z magistralą OPB:

- *opb_sram* – interface pamięci zewnętrznej, np. pamięci statycznej SRAM;
- *opb_dma* – Direct Memory Access (DMA) służący do bardzo szybkiego transferu pomiędzy pamięcią a modułem *opb_lut*;
- *opb_epp* – interface pomiędzy komputerem PC (port równoległy) a wewnętrzną logiką układu FPGA; moduł ten umożliwia konfiguracje modułu *opb_dma* oraz przesyłanie danych dla celów testowych.



Rys. 3. Przykład całego prostego systemu służącego do przekodowania LUT

W tym miejscu należy podkreślić, że pakiet EDK zastał zaprojektowany pod kątem współpracy z soft-procesorem MicroBlaze lub PowerPC. Dlatego wszystkie dostarczone z nim moduły nie są zoptymalizowane pod kątem szybkiego przetwarzania danych. Przykładem może być moduł *opb_cental_dma* dostarczony wraz z pakietem EDK. Posiada on

niestety tylko jedną magistralę OPB, przez co w jednej chwili czasowej może wykonywać tylko jedną czynność: albo odczytywać dane z pamięci (tutaj modułu *opb_sram*), albo zapisywać dane do modułu *opb_lut.* Opracowano więc własny moduł *opb_dma* z trzema magistralami:

1) MS_OPB – magistrala OPB sprzęg Master – pobierający dane z pamięci (*source*);
2) MD_OPB – magistrala OPB sprzęg Master – wysyłający dane do pamięci (*destination*);
3) S_OPB – magistrala OPB sprzęg Slave – służący do konfiguracji modułu: podania adresu źródła, przeznaczenia oraz wielkości transferu.

Moduł ten może równocześnie pobierać i zapisywać dane, przez co szybkość pracy całego systemu uległa zdecydowanemu przyspieszeniu.

Warto w tym miejscu podkreślić, że powyższy system bardzo często wymaga jeszcze dodatkowych modułów takich, jak moduł kontrolny dla modułu *opb_dma*, moduł zapisu danych z kamery wideo, odczytu przetworzonego obrazu itd.

Wyniki implementacji w układzie FPGA schematu przedstawionego na rysunku 3 dla *SOPB_Width=16*, *K=2, MOPB_Width=16* przedstawia tabela 2. Maksymalna częstotliwość pracy została ograniczona przez prędkość odczytu z pamięci zewnętrznej SRAM i wynosiła 50 MHz.

Tabela 2
Wyniki implementacji prostego systemu transformacji LUT
w układzie FPGA Spartan2E xc2s300e-pq208-7 i płycie XSB firmy Xess

	16×1 LUT	Przerzutniki	BRAM
opb_lut	125	33	1
opb_dma	198	66	0
opb_sram	67	74	0
opb_epp	309	119	0
całość	716	242	1

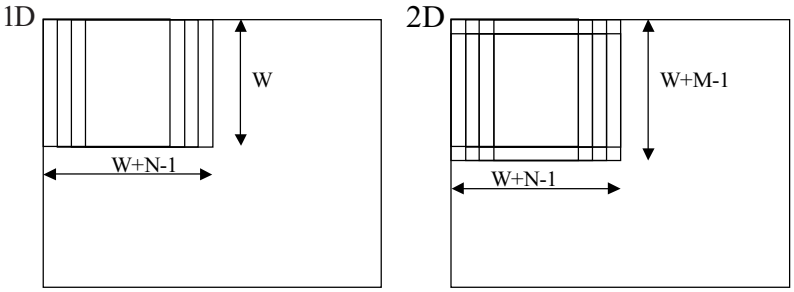
5. Zrównoleglenie operacji LUT

Częstotliwość pracy układów FPGA jest rzędu 100 MHz, dlatego w celu przyspieszenia wykonywania operacji LUT konieczne jest zrównoleglenie algorytmu. Jedna z metod zrównoleglenia została już zaimplementowana w samym module *opb_lut,* w którym możliwe jest wykonywanie wielu operacji LUT równocześnie. Niestety szerokość magistrali danych przy dostępie do pamięci zewnętrznej jest ograniczona, także ten sposób przyspieszenia wykonywania operacji LUT jest niewystarczający. Po drugie dalsze zrównoleglenie algorytmu w podany sposób powodowałoby, że czas programowania (zapisu) pamięci LUT byłby większy niż czas wykonywania samej operacji LUT.

Alternatywnym rozwiązaniem jest zrównoleglenie całego systemu podanego na rysunku 3, czyli zastosowanie *N* modułów *opb_lut, opb_dma, opb_sram.* Rozwiązanie to jest

jednak nie do przyjęcia, ponieważ liczba modułów pamięci zewnętrznej jest ograniczona najczęściej do dwóch.

Proponowanym rozwiązaniem jest zastosowanie tego samego interface'u pamięci zewnętrznej do obliczania wielu operacji LUT równocześnie, jest to możliwe ze względu na częściowe nakładanie się obrazu źródłowego dla wielu niezależnych operacji LUT. W konsekwencji, aby przekodować równolegle N niezależnych sąsiadujących ze sobą albo w poziomie, albo w pionie fragmentów obrazu $W \times W$, wystarczy dostarczyć dodatkowo $W \cdot (N-1)$ pikseli. Ilustruje to rysunek 4 (1D).



Rys. 4. Nakładanie się poszczególnych obrazów wejściowych jednostek równoległych: 1D – tylko w poziomie, 2D – w poziomie i w pionie

W celu przekodowania N oddzielnych sąsiadujących ze sobą fragmentów obrazu $W \times W$ konieczne jest wykonanie P przesłań pikseli źródłowych

$$P = W \times (W + N - 1) \tag{1}$$

Niestety nie wszystkie moduły LUT mogą pracować równocześnie (nie wszystkie piksele są ważne dla danego modułu), dlatego średnie wykorzystanie R każdego modułu LUT wynosi

$$R = \frac{W \cdot W}{P} = \frac{W}{W + N - 1} \tag{2}$$

Analizując powyższe równanie można zauważyć, że największy wzrost przepustowości ($N \cdot R$) osiąga się dla niewielkich N. Dla małych N, średnie wykorzystanie modułów LUT jest bliskie jeden i dlatego zrównoleglenie jest opłacalne. Dla $N = W$, średnie wykorzystanie poszczególnych modułów R wynosi około ½ i dlatego w praktyce górną sensowną granicą jest $N = W$. W konsekwencji dla $W = 20$ sensowny wydaje się wybór $N = 8$ ($N \cdot R = 5{,}92$) lub $N = 16$ ($N \cdot R = 9{,}14$).

Zgodnie z równaniem (2) średnie wykorzystanie modułu LUT szybko maleje ze wzrostem liczby jednostek równoległych N. Dlatego przy dużym N bardziej opłacalne jest wykorzystanie nakładania się sąsiednich pikseli zarówno w pionie, jak i w poziomie. W ogólnej sytuacji można zastosować N jednostek w poziomie i M w pionie. W konsekwencji liczba

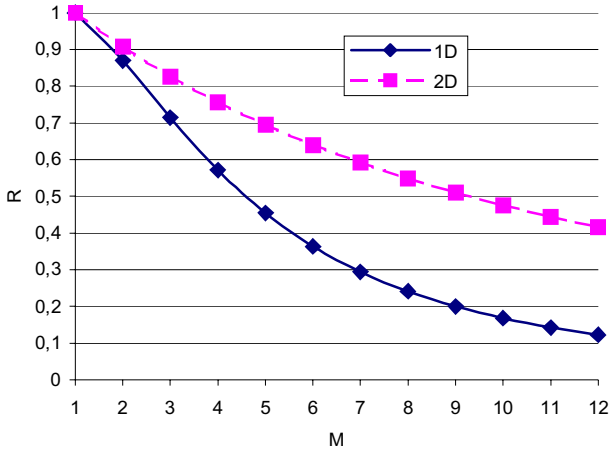
wszystkich równoległych modułów LUT wynosi $N \times M$. Dla takiego rozwiązania liczba potrzebnych pikseli źródłowych (przesłań) P wynosi

$$P_2 = (W+N-1)\times(W+M-1) \tag{3}$$

Średnie wykorzystanie poszczególnych jednostek LUT wynosi

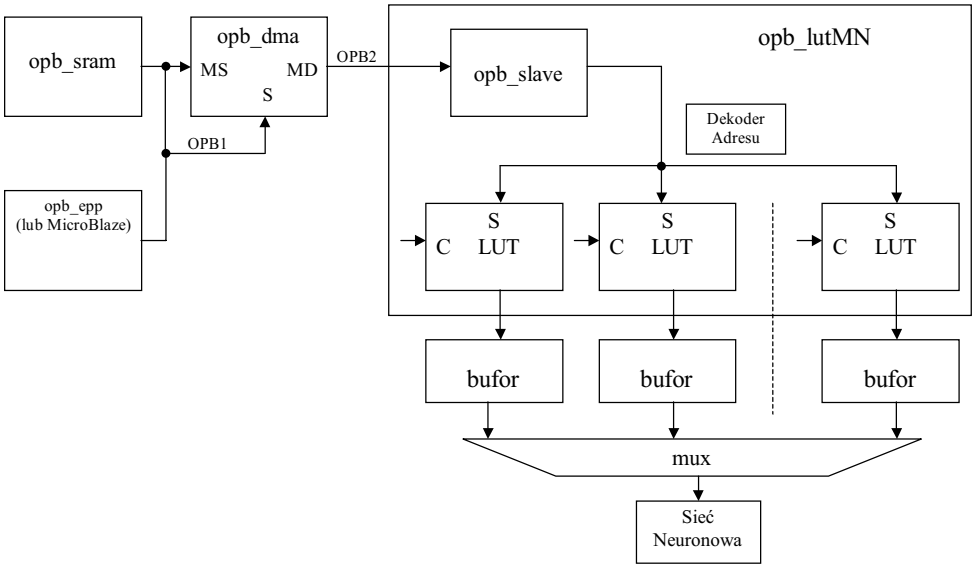
$$R_2 = \frac{W^2}{(W+N-1)\cdot(W+M-1)} \tag{4}$$

Porównanie średniego wykorzystania jednostek równoległych R *i* R_2 dla różnych metod zrównoleglenia: 1D i 2D i tej samej liczby jednostek równoległych zostało przedstawione na rysunku 5. Wynika z niego, że dla $M = 4$, czyli dla 16 równolegle pracujących jednostek, średnie wykorzystanie modułów LUT dla zrównoleglenia 1D wynosi: 0,57, natomiast dla zrównoleglenia 2D wynosi 0,76. W konsekwencji dla większej liczy jednostek równoległych bardziej optymalnym rozwiązaniem jest zrównoleglenie zarówno w poziomie, jak i w pionie. Warto w tym miejscu podkreślić, że przy małym N sensowne jest zastosowanie tylko zrównoleglenia 1D w poziomie, ze względu na to, że odczyt sekwencyjny (następujących po sobie pikseli) jest szybszy od odczytu wybiórczego (od następnej linii).



Rys. 5. Średnie wykorzystanie R w zależności od liczby jednostek równoległych: 1D) $N = M^2$, 2D) $N = M$

Schemat blokowy całego systemu dokonującego $N \times M$ równoległych niezależnych transformacji LUT został przedstawiony na rysunku 6. Moduł *opb_lut* został tutaj zastąpiony $N \times M$ równoległymi modułami LUT. Należy w tym miejscu podkreślić, że każdy z modułów LUT może wykonywać K konwersji LUT równolegle, podobnie jak to miało miejsce dla modułu *opb_lut.* Jedyną różnicą jest to, że wszystkie moduły współdzielą pojedynczy moduł interface magistrali OPB: *opb_slave.*



Rys. 6. Schemat blokowy systemu pracującego z (N·M)–równoległymi modułami konwersji LUT

6. Wpływ czasu programowania pamięci LUT

Podczas rozważania funkcjonalności całego rozwiązania należy również wziąć pod uwagę czas T potrzebny na zaprogramowanie pamięci LUT. Czas ten zależy od szerokości b (w bitach) danej wejściowej oraz od liczby K pamięci LUT znajdujących się w pojedynczym module LUT. Dla $K = 1$ wynosi on $T = 2^b$ taktów zegara, dla $K \geq 2$ wynosi on $T = 2^{b-1}$ taktów zegara. W konsekwencji zwiększenie K powyżej 2 zmniejsza tylko czas wykonywania operacji LUT, natomiast nie zmniejsza czasu programowania LUT. Podczas odczytu (wykonywania operacji LUT), każda pamięć BRAM może pracować niezależnie. W przypadku zapisu nie jest to możliwe – każda pamięć musi zawierać te same dane. Użyte pamięci LUT są pamięciami 2-portowymi, dlatego dalsze zrównoleglenie zapisu nie jest możliwe ze względu na to, że dla $K > 2$ używa się dwóch lub więcej niezależnych pamięci 2-portowych BRAM. Następnym ograniczeniem, które nie będzie poruszane w tym artykule, to trudność z obliczeniem nowej zawartości pamięci LUT.

W konsekwencji całkowite średnie wykorzystanie modułów LUT podczas wykonywania operacji LUT po uwzględnieniu czasu programowania wynosi

$$R_T = \frac{W^2}{\frac{K \cdot 2^b}{MIN(2,K)} + (W+N-1)\cdot(W+M-1)} \tag{5}$$

Dla $W = 20$, $N = M = 1$, $K \leq 2$ oraz $b = 8$ (256 odcieni szarości) $R_T = 0{,}61$, czyli czas programowania pamięci LUT zdecydowanie wpływa na efektywność obliczeniową całego

systemu. Zwiększenie K do 4 powoduje, że funkcja wartości minimalnej: $MIN(2, K)$ – zwraca 2 i konsekwentnie dla tych samych parametrów $R_T = 0{,}44$. Dlatego zalecana wartość to $K = 2$. Co więcej, w sytuacji kiedy jest to możliwe, należy zredukować liczbę poziomów szarości do np. $b = 7$, co poprawi efektywność algorytmu. Należy zaznaczyć, że dla $b < 8$ lub $W > 20$ oraz dużego M i N lepszy efekt może zostać osiągnięty poprzez zwiększenie K niż poprzez dalsze zwiększanie stopnia zrównoleglenia M i N. W każdym przypadku należy wziąć pod uwagę lepsze średnie wykorzystanie R_T modułu LUT wyrażone w równaniu (5).

7. Podsumowanie

Operację LUT można zaimplementować w relatywnie prosty sposób w układach programowalnych FPGA i zajmuje ona niewiele zasobów układu FPGA. W pierwszym przybliżeniu zajmuje ona tylko pamięć blokową BRAM, konsekwentnie – liczba równolegle pracujących jednostek jest z reguły ograniczona właśnie przez liczbę dostępnych pamięci BRAM.

Pamięć BRAM jest 2-portowa i dlatego zalecane jest wykorzystanie równolegle dwóch portów tej pamięci ($K = 2$), co skutkuje pierwszym zrównolegleniem algorytmu na poziomie samego modułu LUT. Zwiększenie równoległości na poziomie modułu LUT (K) jest jednak rzadko zalecane, ze względu na większy wpływ czasu programowania (zapisu) pamięci LUT.

Następnym etapem przyspieszenia obliczeń jest wykonywanie równolegle transformacji LUT dla wielu oddzielnych sąsiadujących ze sobą fragmentów obrazu źródłowego. Wykorzystuje się w tym celu fakt, że obrazy źródłowe sąsiednich fragmentów nakładają się, dzięki temu pojedynczy interface z pamięcią zewnętrzną może być wykorzystywany równocześnie przez wiele modułów równoległych. Niestety wraz ze wzrostem liczby jednostek równoległych część wspólna obrazu źródłowego maleje, co w dużym stopniu wpływa na efektywność wykorzystania jednostek równoległych. W konsekwencji udowodniono, że dla dużego stopnia zrównoleglenia, lepszym rozwiązaniem jest zrównoleglenie algorytmu zarówno w poziomie, jak i w pionie. W rezultacie część wspólna obrazu źródłowego jest większa, co poprawia wykorzystanie jednostek równoległych. Dla $M \times N = 64$ jednostek równoległych współczynnik wykorzystania poszczególnych jednostek jest większy niż 0,5. Podsumowując: możliwe jest efektywne zrównoleglenie wykonywania operacji LUT 3D ($K \times M \times N$) rzędu 128 przy średnim wykorzystaniu poszczególnych jednostek R_2 powyżej 0,5.

Literatura

[1] Gonzalez R., Wintz P.: *Digital Image Processing.* Addision-Wesley, 1987

[2] Wiatr K.: *Architektura potokowa specjalizowanych procesorów sprzętowych do wstępnego przetwarzania obrazów w systemach wizyjnych czasu rzeczywistego*. Kraków, Wydawnictwo AGH 1998

[3] Henry A., Rowley, Shumeet Baluja, Takeo Kanade: *Neural Network-Based Face Detection*. Ieee Transactions on Pattern Analysis and Machine Intelligence, vol. 20, No. 1, January 1998, 23–38

[4] Jamro E., Waitr K.: *Heterogeneouse Hardware-Software Prototyping System for PC-controlled FPGA-based Designs*. Proc. of IFAC Workshop on Programmable Devices and Systems PDS, Kraków, Nov. 18–19 2004, 186–191