AUTOMATYKA • 2006 • Tom 10 • Zeszyt 3

Ernest Jamro*, Kazimierz Wiatr*

Potokowe przetwarzanie obrazu w oparciu o środowisko EDK i magistralę OPB

1. Wprowadzenie

Gwałtowny rozwój układów programowalnych FPGA spowodował, że mają one coraz większe znaczenie podczas cyfrowego przetwarzania sygnałów DSP (*Digital Signal Processing*). Z drugiej strony wielkość dostępnych zasobów powoduje, że skala skomplikowania pojedynczego projektu przewyższa możliwości projektowe pojedynczej osoby oraz wymusza usystematyzowany sposób łączenia poszczególnych zasobów sprzętowych. W konsekwencji konieczne stało się zastosowanie środowiska modułowego umożliwiającego łatwe łączenie gotowych modułów, których interface oraz ogólne właściwości są łatwe do wykorzystania. Przykładem takiego środowiska jest system Embedded Development Kit (EDK) firmy Xilinx [1] oraz magistrala On-chip Peripheral Bus (OPB) [2] opracowana przez firmę IBM. W konsekwencji standard łączenia poszczególnych modułów jest zdefiniowany za pomocą magistrali OPB, a środowisko EDK umożliwia łatwe graficzne reprezentowanie poszczególnych modułów oraz definiowanie i zmienianie parametrów tych modułów. Podsumowując, w ten sposób uzyskuje się zdecydowanie przyspieszenie projektowania SoC (*System on a Chip*).

Środowisko EDK jest jednym z wielu środowisk służących do projektowania SoC. Do niedawna najpopularniejszą metodą było bezpośrednie projektowanie za pomocą schematów elektronicznych lub języka opisu sprzętu np. VHDL. Oczywiście coraz częstsze jest wykorzystywanie gotowych modułów IP (*Intellectual Property*) oraz określenie standardu łączenia poszczególnych modułów. Przykładem może być organizacja OpenCores zalecająca magistralę Wishbone oraz udostępniająca dużą bazę dostępnych gotowych modułów na stronie www.opencores.org. Niestety łączenie poszczególnych modułów na poziomie języka VHDL jest czasochłonne i przy większych projektach raczej nieczytelne.

Alternatywnym rozwiązaniem jest System Generator [3], który jest stowarzyszony z pakietem Matlab / Simulink. W konsekwencji projektowanie systemu cyfrowego przetwarzania danych DSP jest bezpośrednio zintegrowane ze środowiskiem Matlab, dzięki temu proces projektowania oraz symulacji jest zdecydowanie łatwiejszy i szybszy. Pakiet

* Katedra Elektroniki, ACK Cyfronet, Akademia Górniczo-Hutnicza w Krakowie

Matlab został stworzony głównie z myślą o aplikacjach matematycznych i jest on bardzo często używany podczas projektowania modułów DSP. Dlatego integracja, a przede wszystkim symulacja całego toru DSP jest zdecydowanie szybsza i łatwiejsza. W ramach pakietu System Generator dostępnych jest wiele gotowych modułów, np. moduły DSP: układy mnożące, filtry, FFT; moduły kontrolne: liczniki, bufory FIFO. Porównując pakiet System Generator z EDK, można odnieść wrażenie, że pakiet EDK jest pakietem nadrzędnym w stosunku do System Generator, tzn. cały projekt w System Generator może być dalej wykorzystywany jako gotowy pojedynczy moduł w pakiecie EDK. W tym miejscu należy podkreślić, że System Generator posiada odpowiednie procedury do automatycznego generowania takiego modułu. Podsumowując, główną zaletą pakietu System Generator jest łatwość testowania toru DSP, natomiast istnieje pewna trudność w generowaniu całkowicie niezależnego systemu wykorzystującego dostępne zasoby sprzętowe (szczególnie zewnętrzne w stosunku do układu FPGA) oraz hardware/ software codesign.

Myślą przewodnią niniejszego artykułu jest pokazanie nowej filozofii projektowania układów FPGA dla potrzeb cyfrowego przetwarzania sygnałów, w szczególności niskopoziomowego przetwarzania obrazu, np. filtracji, operacji Look-Up-Table. Użycie pakietu EDK oraz standardu magistrali OPB jest podstawą niniejszej filozofii. Dodatkowo wykorzystywanie wielu lokalnych magistral OPB znacząco przyspiesza transfer danych i efektywność całego systemu. Lokalne przetwarzanie danych wymusza usystematyzowanie łączenia poszczególnych modułów dlatego zaproponowano aby łączenie to odbywało się z wykorzystaniem architektury potokowej.

2. Embedded Development Kit

Pakiet Embedded Development Kit (EDK) został zaprojektowany przez firmę Xilinx głównie z myślą o systemie umożliwiającym wykorzystanie procesorów MicroBlaze i PowerPC [1]. System EDK wykorzystuje modułowy sposób projektowania, czyli projektant wykorzystujący środowisko EDK skupia się głównie na łączeniu gotowych modułów poprzez zdefiniowane magistrale. W ramach pakietu EDK można wykorzystywać wiele różnych standardów magistral, jednak najbardziej popularna jest magistrala On-chip Peripheral Bus (OPB) [2]. Środowisko EDK umożliwia graficzną prezentacje całego projektu składającego się z wielu różnych modułów oraz w łatwy sposób pozwala na łączenie poszczególnych modułów. W konsekwencji EDK jest pakietem, który zdecydowanie przyspiesza proces projektowania układów FPGA.

Pakiet EDK dostarczony przez firmę Xilinx posiada około 70 gotowych modułów, służących głównie do komunikacji pomiędzy różnymi standardami magistral, np. PLB, PCI, IIC, UART, Ethernet oraz obsługi zewnętrznych pamięci. Niestety z wyjątkiem procesorów MicroBlaze i PowerPC pakiet ten nie zawiera gotowych modułów wspomagających proces obliczeniowy. Dużą zaletą pakietu EDK jest to, że umożliwia on dodawanie własnych modułów, które są zgodne z wymaganiami pakietu EDK. Moduły własne mogą być zaprojektowane w języku opisu sprzętu, np. VHDL. Dużą zaletą języka VHDL jest możliwość projektowania z parametrem dzięki użyciu słowa kluczowego *generic.* Projektowanie z parametrem ma szereg zalet [4], z których najważniejsze jest to, że dzięki sparametryzowaniu

projektu jego funkcjonalność i elastyczność gwałtownie wzrasta. Dzięki temu pojedynczy silnie sparametryzowany moduł może być użyty do wielu różnych zadań. Przykładem może być parametryzacja szerokości bitowej bloku arytmetycznego, dzięki czemu precyzja obliczeń może być dobierana zgodnie z wymaganiami projektanta systemu. W tym miejscu należy wspomnieć, że środowisko EDK umożliwia w prosty sposób określanie parametrów i przekazywanie ich do projektu napisanego w języku VHDL, jest to integralna i bardzo ważna część wspomnianego pakietu.

Gotowe moduły IP dostarczone w ramach pakietu EDK zostały zoptymalizowane ze względu na współprace z procesorem MicroBlaze lub PowerPC. W związku z tym, jądrem całego systemu powinien być procesor, który nadzoruje i kontroluje cały system. W systemie takim pojedyncza magistrala globalna podłączona do procesora steruje większością modułów. Niestety moc obliczeniowa wspomnianych procesorów jest niewielka, dlatego taki system nie jest zalecany. Zdecydowanie większą moc obliczeniową zapewniają dedykowane moduły sprzętowe, dlatego celem niniejszej pracy jest zaprojektowanie, a następnie szybkie łączenie takich dedykowanych modułów. Wymaga to jednak zmiany filozofii projektowania w środowisku EDK oraz opracowanie własnych metod łączenia poszczególnych modułów.

3. Lokalne wykonywanie obliczeń

Przez lokalne wykonywanie obliczeń rozumie się cały system obliczeniowy posiadający wiele niezależnych magistral lokalnych, poprzez które transmitowane są dane. Dane te widoczne są z reguły tylko lokalnie, najlepiej tylko przez dwa moduły: nadawcy i odbiorcy. Liczba magistral oraz ich architektura powinna być tak dobrana, aby maksymalizować szybkość obliczeń (czyli również przesył danych). W konsekwencji tworzonych jest wiele magistral lokalnych. Lokalne przetwarzanie danych ma szereg zalet:

- Do magistrali lokalnej podłączonych jest mniej urządzeń, co powoduje mniejsze współdzielenie magistrali. W konsekwencji poszczególne jednostki obliczeniowe mogą wykorzystywać daną magistralę przez dłuższy czas, co zdecydowanie poprawia szybkość transmisji.
- Dzięki zastosowaniu magistrali lokalnej długość połączeń jest znacznie mniejsza, co skraca czas propagacji sygnałów, czyli magistrala może być taktowana z większą częstotliwością.
- Zasoby zajmowane przez magistralę ulegają zmniejszeniu, np. system arbitrażu magistrali może być prostszy (np. możliwe jest zastosowanie stałego priorytety przyznawania magistrali). W szczególnym przypadku, kiedy na danej magistrali znajduje się tylko jedno urządzenie typu master, arbitraż magistrali nie jest w ogóle potrzebny. Dodatkowo w przypadku, kiedy przesył danych odbywa się tylko w jednym kierunku (np. od urządzenia typu master do urządzenia typu slave), zasoby zajmowane przez magistralę są teoretycznie równe zero.

Podstawową wadą zastosowania lokalnego wykonywania obliczeń jest brak łatwego dostępu do danych na poszczególnych etapach obliczeniowych. Dostęp ten jest potrzebny szczególnie do celów testowych. W przypadku układów FPGA, które mogą być w łatwy

sposób rekonfigurowane, rozwiązanie tego problemu jest stosunkowo proste: do celów testowych możliwe jest tymczasowe dołączanie do wybranych magistral lokalnych dodatkowych modułów, których zadaniem jest rejestracja aktywności danych magistral.

Środowisko EDK oraz dostępne zasoby sprzętowe układów FPGA umożliwiają budowanie systemu z wieloma niezależnymi magistralami lokalnymi, dzięki temu projektant ma do dyspozycji narzędzie, w którym lokalne przetwarzanie danych nabiera zupełnie innego znaczenia. W tym miejscu należy zwrócić uwagę, że w układach FPGA znajdują się relatywnie duże zasoby pamięci blokowej BRAM. Pamięci te są z reguły za małe do składowania danych globalnych, jednakże świetnie spełniają swoją funkcję w przypadku składowania niewielkiej ilości danych lokalnych. Duża liczba niezależnych pamięci BRAM (np. 444 dla układu XC2VP100 firmy Xilinx) powoduje, że preferowane jest lokalne przetwarzanie danych.

System obliczeniowy z lokalnym przetwarzaniem danych można zaprojektować na wiele różnych sposobów. Dlatego w tym miejscu bardzo ważną rolę spełnia projektant, który dzięki swojej wiedzy może zaprojektować system o bardzo dużej mocy obliczeniowej, składający się z wielu lokalnych modułów. Moc obliczeniowa pojedynczego modułu może być relatywnie mała, ale równoległa czy też potokowa praca wszystkich modułów lokalnych powoduje, że sumaryczna moc obliczeniowa całego systemu jest olbrzymia.

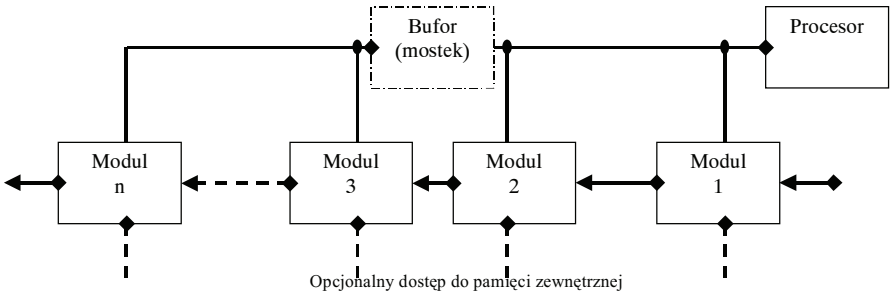
4. Potokowe łączenie modułów

Potokowe łączenie modułów obliczeniowych zostało zaproponowane z wykorzystaniem architektury DePiAr [5]. W systemie tym poszczególne moduły obliczeniowe w szczególności wstępnego przetwarzania obrazu były połączone potokowo. W konsekwencji zastosowania architektury DePiAr osiągnięto znaczne przyspieszenie wykonywania niektórych operacji wstępnego przetwarzania obrazu. System oparty ma magistrali DePiAr został zaproponowany w latach, kiedy zasoby pojedynczego układu FPGA były zbyt małe, aby możliwe było zbudowanie całego systemu w pojedynczym układzie FPGA. Dlatego poszczególne moduły obliczeniowe pracujące potokowo znajdowały się na osobnych płytach PCB, na których osadzone były układy FPGA, dedykowane układy VLSI, pamięci, bufory FIFO, itd. Osobne moduły PCB mogły być fizycznie wkładane w zależności od przeprowadzanych operacji obliczeniowych i odpowiednio konfigurowane poprzez magistralę VME oraz procesor. Głównym zadaniem procesora było przede wszystkim nadzorowanie i konfigurowanie poszczególnych modułów sprzętowych a nie przeprowadzanie obliczeń.

Dzisiaj zasoby sprzętowe pojedynczego układu FPGA są już na tyle duże, że możliwe jest zaimplementowanie całego systemu obliczeniowego w pojedynczym układzie scalonym. Dlatego zamiast poszczególnych niezależnych modułów obliczeniowych znajdujących się na osobnych płytach PCB, możliwe jest zastosowanie tylko pojedynczego układu FPGA oraz modułowego projektowania całego systemu, podobnie jak to miało miejsce w przypadku magistrali DePiAr. Magistrala DePiAr była magistralą zewnętrzną, tzn. łączyła niezależne układy znajdujące się na różnych płytach PCB. Ponadto pracowała ona z częstotliwością 15 MHz, która jest mała, jak na możliwości dzisiejszych układów FPGA. W konsekwencji zastosowanie tej magistrali do łączenia poszczególnych modułów wewnątrz pojedynczego układu FPGA wydaje się niezasadne.

Zamiast magistrali DePiAr wybrano magistralę OPB oraz środowisko EDK do modułowego łączenia poszczególnych modułów. Jednak filozofia potokowego łączenia poszczególnych modułów pozostała niezmieniona. Podsumowując, w niniejszym artykule zaproponowano nową architekturę On-chip Pipeline Architecture (OPiAr), która jest oparta na modyfikacji architektury potokowej DePiAr oraz jest przystosowana do magistrali wewnętrznej OPB.

Aby możliwe było łatwe łączenie modułów, konieczne było określenie standardowego interfejsu przesyłania danych w architekturze OPiAr. Ostatecznie określono, podobnie jak to było w przypadku architektury DePiAr, że dane wejściowe są odbierane poprzez interfejs typu slave, a dane wyjściowe (po przetworzeniu przez dany moduł) są przesyłane dalej poprzez interfejs typu master. Dodatkowy interfejs typu slave służy do konfiguracji danego urządzenia poprzez procesor ogólnego przeznaczenia. Przykład prostego systemu obliczeniowego stosującego architekturę OPiAr przedstawiono na rysunku 1. Interfejsy typu master zostały zaznaczone odwróconym prostokątem, pozostałe interfejsy są typu slave.



Rys. 1. Schemat blokowy sytemu OPiAr

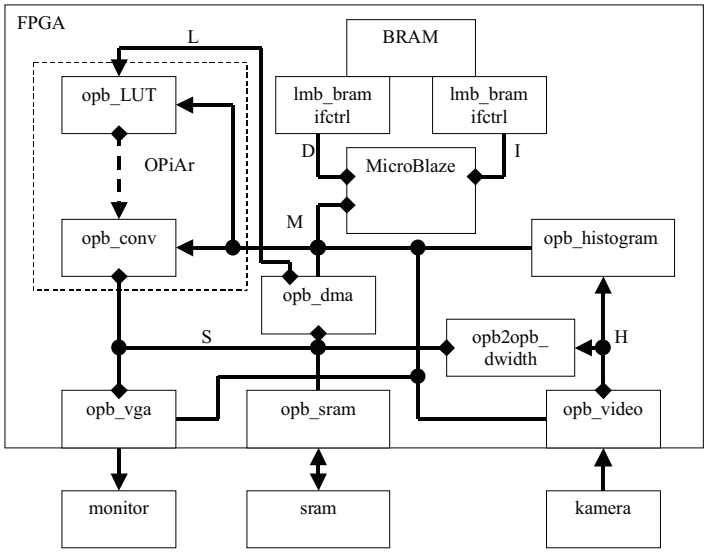
Warto w tym miejscu podkreślić, że filozofia potokowego łączenia modułów stanowi jedynie wytyczną podczas projektowania. Możliwa jest zmiana architektury całego systemu jeżeli rodzaj przeprowadzanych operacji to uzasadnia. Jest to możliwe, ponieważ w przeciwieństwie do magistrali zewnętrznej DePiAr odstępstwo od architektury potokowej w przypadku zastosowania magistrali OPB i środowiska EDK jest relatywnie łatwe do zrealizowania. Należy jednak podkreślić, że czas projektowania poszczególnych modułów obliczeniowych jest stosunkowo długi, dlatego odstępstwo od architektury OPiAr jest zalecane tylko w uzasadnionych przypadkach.

Warto również wspomnieć o pewnym ograniczeniu architektury OPiAr w porównaniu z architekturą DePiAr: architektura DePiAr składała się z wielu niezależnych płyt PCB i układów scalonych, których możliwości, szczególnie jeśli chodzi o zasoby pamięci, mogły być dużo większe niż możliwości pojedynczego układu FPGA. Dlatego ze względu na dostępne zasoby pamięci BRAM w pojedynczym układzie FPGA nie jest możliwe (w chwili obecnej) składowanie wewnątrz pojedynczego modułu architektury OPiAr całej ramki obrazu. Dlatego w takim przypadku każdy z modułów może mieć dostęp do pamięci zewnętrznej poprzez dodatkowy interfejs typu master. Oczywiście optymalnym rozwiązaniem byłoby, aby każdy z modułów, który tego wymaga, miał dostęp do niezależnej pamię-

ci. Niestety liczba interfejsów pamięci zewnętrznej jest z reguły ograniczona i w takim przypadku interfejs ten musi być współdzielony przez poszczególne moduły. Na szczęście relatywnie niewielka liczba modułów wymaga dostępu do pamięci zewnętrznej.

5. Przykład całego systemu

Przykład prostego systemu składającego się z wielu magistral lokalnych jest pokazany na rysunku 2.



Rys. 2. Schemat blokowy systemu dokonującego wyrównania histogramu

Strzałki pokazują kierunek przepływu danych, a prostokąt znajdujący się na końcu magistrali oznacza, że jest to interfejs typu master. Poszczególne magistrale służą do niżej opisanych celów.

– H (histogram) – magistrala służy do dwóch celów: do przesyłania danych z kamery video do modułu mostka *opb2opb_dwidth* i dalej do pamięci zewnętrznej SRAM oraz do obliczania histogramu w module *opb_histogram*. Moduł *opb_histogram* posiada lokalną (wewnętrzną) pamięć BRAM służącą do zapisywania obliczonego histogramu [6]. Warto w tym miejscu podkreślić, że moduł *opb_histogram* pracuje w trybie śledzenia – czyli sam nie odpowiada sygnałem *Sl_xferAck* na transfer danych [2], a jedynie śledzi i rejestruje dane, które są przesyłane do innego urządzenia (mostka). Dzięki temu liczba transferów jest dwukrotnie mniejsza oraz system jest uproszczony. Moduły *opb_video* oraz *opb_histogram* operują na danych 8-bitowych monochromatycznych. Magistrala *S* jest magistralą szerszą 16- lub 32-bitową, dlatego umieszczono dodatkowy mostek *opb2opb_dwidth* łączący wspomniane magistrale OPB. Mostek ten dokonuje konwersji szerokości magistral oraz posiada dodatkowe bufory FIFO optymalizujące transfer danych.

– S (sram) – magistrala służy do wymiany danych z pamięcią zewnętrzną np. pamięcią SRAM. Pamięć zewnętrzna jest współdzielona pomiędzy 4 zadania: zapis obrazu wideo z kamery; odczyt obrazu wideo z kamery w celu przetworzenia w architekturze OPiAr; zapis obrazu po dokonaniu obliczeń w architekturze DePiAr; odczyt danych w celu wyświetlenia na monitorze. Interfejs z pamięcią zewnętrzną może stanowić wąskie gardło całego systemu, dlatego w takim wypadku, jeżeli dysponuje się dwoma niezależnymi interfejsami pamięci zewnętrznej, należy rozdzielić fizycznie pamięć przed i po przetworzeniu w module OPiAr. Dodatkowo szerokość magistrali danych jest większa (16- lub 32-bitów), dzięki temu szybkość transmisji ulega przyspieszeniu – jest to możliwe ponieważ interfejs z pamięcią zewnętrzną jest z reguły 16- lub 32-bitowy (np. dla płyt XSV lub XSB firmy Xess).
– L (LUT wejście do architektury OPiAr) – magistrala służy do transmisji danych pomiędzy modułem *opb_dma* oraz *opb_lut* (lub innym modułem architektury OPiAr). W tym miejscu należy podkreślić, zupełnie inną budowę modułu *opb_dma* zaprojektowanego przez autorów niniejszego artykułu oraz modułu *opb_central_dma* dostarczonego przez firmę Xilinx. Moduł *opb_dma* posiada dwie niezależne magistrale typu master osobną do odczytu i zapisu danych, dzięki czemu przesyłanie danych może następować z dwukrotnie większą prędkością, niż to miało miejsce w przypadku modułu *opb_central_dma,* który posiadał tylko jedną magistralę typu master. Magistrala *L* jest typową magistralą lokalną, składająca się tylko z jednego modułu typu master i jednego modułu slave. Oczywiste jest, że w tym przypadku moduł *opb_dma* mógłby być bezpośrednio włączony do modułu *opb_lut* (czyli moduł *opb_lut* zawierałby również wewnętrzny moduł podobny do modułu *opb_dma*), jednakże nie spełniałby on zaleceń architektury OPiAr oraz elastyczność (możliwość użycia w różnych konfiguracjach) tego modułu byłaby dużo mniejsza.
– M (MicroBlaze) – jest to magistrala nadzorowana przez procesor MicroBlaze. Zadaniem tego procesora jest odczytać obliczony histogram oraz na podstawie tego histogramu odpowiednio zakodować pamięć LUT modułu *opb_lut* oraz innych modułów architektury OPiAr. Dodatkowym zadaniem jest odpowiednie zaprogramowanie modułu *opb_dma, opb_video* oraz *opb_vga.* Ze względu na skrócenie czasów propagacji sygnałów na tej magistrali, zaleca się dodanie dodatkowych mostków (zob. rys. 1)
– D (dane) – magistrala LMB służąca do dostępu do lokalnych danych procesora MicorBlaze.
– I (instrukcje) – magistrala LMB służąca do pobierania instrukcji przez procesor MicroBlaze.

6. Interface magistrali OPB

Lokalne przetwarzanie danych w środowisku EDK powoduje, że szybkość budowy całego systemu składającego się z wielu niezależnych modułów gwałtownie wzrasta. Negatywną stroną modułowego projektowania są dodatkowe zasoby sprzętowe zajmowane przez magistralę OPB łączącą poszczególne moduły. Oczywiste jest, że jeżeli na magistrali OPB znajduje się tylko jedno urządzenie typu master i tylko jedno urządzenie typu slave oraz jeżeli transfer danych odbywa się tylko od urządzenia typu master do urządzenia typu

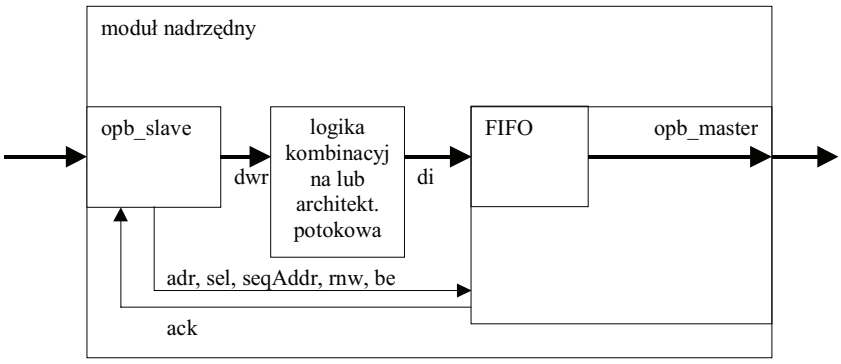
slave (jak ma to miejsce w przypadku magistrali OPiAr), to pełne możliwości magistrali OPB nie są wykorzystywane. Dlatego, z punktu widzenia zajmowanych zasobów, najlepszym rozwiązaniem byłoby zastosowanie bezpośredniego połączenia pomiędzy tymi modułami. Niestety takie połączenie byłoby mało uniwersalne, trudne do konfiguracji i rozbudowy i nie mogłoby być wykorzystywane w przypadku bardziej skomplikowanych połączeń.

Rozwiązaniem powyższego problemu jest projekt własnego modułu interfejsu magistrali OPB, który w prosty sposób wykorzystywałby tylko te zasoby, które są potrzebne w ramach poszczególnych magistral. Wspomniane moduły wykonane przez autorów niniejszej publikacji noszą nazwę *opb_master* oraz *opb_slave*. Określenie parametrów poszczególnych modułów odbywa się poprzez środowisko EDK oraz słowo kluczowe *generic* w języku VHDL. Na szczególną uwagę zasługują niżej wymienione parametry tych modułów.

- Programowalna wielkość wewnętrznej kolejki First-In First-Out (FIFO). Dla modułu FIFO użyto wbudowanych (w układach Virtex) rejestrów przesuwnych (SRL). Oczywiście możliwa jest rezygnacja w ogóle z buforu FIFO poprzez zastosowanie tylko rejestru, bezpośredniego połączenia lub też całkowitego braku połączenia. Określenie wielkości buforów FIFO lub rodzaju połączenia odbywa się niezależnie dla ścieżki odczytu i zapisu, dzięki temu w przypadku architektury OPiAr, która wykonuje tylko zapis, ścieżka odczytu nie jest w ogóle implementowana.
- Sparametryzowanie szerokości magistrali danych zarówno po stronie magistrali OPB, jak i po stronie logiki użytkownika. Zaprojektowano dodatkowy układ umożliwiający wewnętrzną konwersję szerokości danych, która odbywa się praktycznie bez zaangażowania projektanta. Powyższa cecha umożliwia nie tylko konwersję szerokości danych ale również poprawia szybkości działania poszczególnych modułów. 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 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. W konsekwencji pamięć LUT może ciągłe wykonywać operacje LUT pomimo to, że magistrala OPB jest współdzielona przez wiele urządzeń. W przypadku użycia architektury OPiAr często połączenie pomiędzy poszczególnymi modułami jest tylko 8-bitowe, dlatego możliwe jest określenie, że magistrala OPB jest tylko 8-bitowa i nie ma potrzeby dokonywania konwersji szerokości danych pomiędzy poszczególnymi modułami, dzięki temu powierzchnia zajmowana przez układ jest zdecydowanie mniejsza.
- Stosowanie (lub też nie) logiki Byte Enable – pozwala to na dalsze zoptymalizowanie zajmowanej powierzchni szczególnie w przypadku zastosowania kolejek FIFO.
- 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. Na uwagę zasługuje możliwość zastosowania buforów FIFO oraz adresowania sekwen-

cyjnego. Transfer blokowy na współdzielonej magistrali OPB często kończy się przedwcześnie, aby umożliwić dostęp do magistrali OPB innym urządzeniom. W konsekwencji jeden duży transfer blokowy często dzielony jest na wiele mniejszych transferów blokowych. Niestety każdy z tych mniejszych transferów blokowych jest dalej traktowany jako niezależny transfer blokowy. Dlatego w niektórych sytuacjach celowe jest automatyczne wykrywanie adresowania sekwencyjnego nie na podstawie sygnału *OPB_seqAddr* ale na podstawie porównania kolejnych adresów. Dzięki temu możliwe jest odtworzenie oryginalnego dużego transferu blokowego.

– Sparametryzowane opóźnienie potokowe na ścieżce danych wewnątrz każdego modułu. Powyższa cecha powoduje, że w prosty sposób, bez dodatkowej logiki kontrolnej, można dodać rejestry potokowe w celu przyspieszenia maksymalnej częstotliwości pracy logiki użytkownika. W przypadku gdyby nie było wspomnianego parametru, 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. Przykład podłączenia modułu *opb_slave* z modułem *opb_master* przez moduł obliczeniowy o różnej liczbie etapów potokowych pokazano na rysunku 3. Zmiana liczby etapów potokowych w zależności od wymaganej częstotliwości pracy układu z punktu widzenia całego projektu odbywa się bardzo prosto: wystarczy dodać dodatkowe rejestry potokowe w logice obliczeniowej użytkownika, a następnie zwiększyć odpowiedni parametr modułu *opb_master* o liczbę dodanych rejestrów potokowych.
– Zoptymalizowanie przyznawania magistrali OPB ze względu na stan buforów FIFO, oraz sparametryzowana typowa liczby przesłań podczas pojedynczego transferu blokowego.
– Dodatkowy parametr *single_master,* który określa, czy dany interfejs typu master jest jedynym interfejsem tego typu na danej magistrali OPB. Dzięki użyciu powyższego parametru moduł *opb_master* może nie używać logiki arbitrażu magistrali oraz nie stosować dodatkowych bramek AND-OR służących do multipleksacji, np. magistrali adresowej.



Rys. 3. Przykład prostego projektu z wykorzystaniem modułów *opb_master* i *opb_slave*

7. Wyniki implementacji

Jako wynik implementacji podano zasoby zajmowane przez modułu *opb_master* dla różnych jego parametrów. Ponieważ liczba parametrów jest duża i podanie wyników implementacji wszystkich możliwych kombinacji tych parametrów jest niemożliwe, wybrano podstawowe wartości tych parametrów – wzorzec przedstawiony na listingu 1. Kolejne implementacje mają wszystkie oprócz wymienionych parametry takie same jak wzorzec.

Listing 1. Parametry modułu opb_master i ich wartości wzorcowe

```
 opb_master generic(
opb_dwidth: integer:= 32; —opb master data width
dwidth: integer:= 8; — internal (user) data width, the LSBs of OPB_DBus='0'
real_dwidth: integer:= 8; — used OPB bus dwidth, (opb_dwidth-real_dwidth) LSBs= '0'
opb_awidth: integer:= 32; — opb address bus width
awidth: integer:= 16; — internal user address bus width
transfer_size: integer:= 4; — min transfer size when bus is locked
use_be: integer:= 1; — >0 use Byte Enable Logic, <=0 assume full transfer
use_seqAddr: integer:= 1; — =1 use sequential addressing, =0 -ignore seqAddr signal
use_addr_counter: integer:= 0; — internal address counter
single_master: integer:= 0; — =1 only a single master is atached to the OPB bus
fifo_wr_awidth: integer:= 0; — write fifo address width
fifo_rd_awidth: integer:= 0; — read fifo address width
wr_pipeline_latency: integer:= 0); —user (external) write logic pipeline latency
```

Analizując wyniki implementacji zamieszczone w tabeli 1, można dojść do wniosku, że ustawienie parametrów silnie wpływa nie tylko na funkcjonalność, ale również na zajmowane zasoby układu FPGA. W przypadku użycia modułu *opb_master* w architekturze OPiAr (parametry: *fifo_rd_awidth*= -1, *single_master*= 1) moduł zajmuje tylko 7 LUT. W konsekwencji dodatkowe nakłady sprzętowe zajmowane przez połączenia poszczególnych modułów są niewielkie. Na uwagę zasługuje parametr *single_master=1*, który umożliwia rezygnację z logiki arbitrażu magistrali i multipleksacji magistrali adresowej poprzez wiele urządzeń typu master. Ponadto konwersja szerokość bitowej magistrali danych zajmuje znaczące zasoby układu FPGA, dlatego należy jej unikać. Ze względu na skrócenie czasów propagacji zaleca się stosować dodatkowe moduły FIFO pomimo to, że zajmują one dodatkowe zasoby. Warto w tym miejscu podkreślić, że rezultaty podane w tabeli 1 obejmują tylko urządzenie typu master, czyli pominięto urządzenie typu slave oraz inne zasoby magistrali OPB (logikę arbitrażu, timeoutu oraz multiplekserów).

Warto jednak podkreślić, że duża liczba dostępnych zasobów układów FPGA powoduje, że coraz większe znaczenie ma szybkość projektowania i łączenia modułów, a nie wielkość zajmowanych zasobów.

Tabela 1
Wyniki implementacji dla modułu *opb_master* dla różnych parametrów

Rodzaj	FF	LUT
Listing 1 (wzór)	3	61
dwidth= 16 (konwersja z 8 na 16 bitów)	21	124
dwidth= 32 (konwersja z 8 na 32 bity)	32	160
real_dwidth= 16 (konwersja z 8 na 16 bitów)	30	155
real_dwidth= 32 (konwersja z 8 na 32 bitów)	40	206
dwidth= 16, real_dwidth= 16 (magistrala zawsze 16-bitowa)	3	69
dwidth= 32, real_dwidth= 32 (magistrala zawsze 32-bitowa)	3	88
awidth= 32 (magistrala adresowa użytkownika 32-bitowa)	3	55
transfer_size= 1 (zwalniaj magistralę, jak tylko inne urządzenie żąda magist.)	2	61
transfer_size= 16 (posiadaj magistralę przez 16 transferów)	5	66
dwidth= 32, real_dwidth= 32, use_be= 0 (bez logiki Byte Enable)	3	82
use_seqAddr= 0 (bez adresowania sekwencyjnego)	1	47
use_seqAddr= 2 (z automatyczna detekcja adresowania sekwencyjnego)	4	107
use_addr_counter= 1 (użyj wewnętrznej logiki inkrementacji adresu)	20	81
single_master= 1 (tylko jedno urządzenie master)	0	15
fifo_wr_awidth= -1 (brak ścieżki zapisu)	4	52
fifo_rd_awidth= -1 (brak ścieżki odczytu)	3	60
fifo_rd_awidth= -1, single_master= 1 (kombinacja dwóch poprzednich)	0	7
fifo_wr_awidth=1, fifo_rd_awidth=1 (dodatkowe rejestry na ścieżce danych)	27	86
fifo_wr_awidth= 4, fifo_rd_awidth=4 (bufory FIFO – 16 pozycji)	33	134
fifo_wr_awidth= 4, fifo_rd_awidth=-1 (tylko ścieżka zapisu z kolejka FIFO)	28	103

8. Podsumowanie

W niniejszym artykule zaproponowano potokowe przetwarzanie obrazu z wykorzystaniem środowiska EDK i standardu magistrali OPB. Zaproponowano nową architekturę OPiAr (On-Chip Pipeline Architecture) będącą adaptacją architektury potokowej DePiAr do pakietu EDK i pojedynczego układu FPGA. W tym miejscu należy podkreślić, że nie zamieszczono w niniejszym artykule budowy przykładowych modułów i nie podano wyników implementacji pełnego systemu z powodu braku miejsca. Niemniej większość wyników zaprezentowanych dla magistrali DePiAr [5] stosuje się również do architektury OPiAr, podstawowa różnica to częstotliwość taktowania, która uległa zwiększeniu

z 15 MHz do co najmniej 50 MHz. Stało się to możliwe, dzięki zastosowaniu nowszych i szybszych układów FPGA (XC2S300E, XCV800), oraz wewnętrznych (w układzie FPGA) a przez to szybciej propagujących sygnałów. Autorzy niniejszego artykułu dysponują na przykład następującymi modułami zgodnymi z architekturą OPiAr: Look-Up-Table, konwolwer, binaryzacja, koder Huffmana. Dodatkowe moduły są w trakcie realizacji. Przykłady pojedynczych modułów (wraz z wynikami implementacji) zgodnych z architekturą OPiAr podano, np. w [6, 7]. W tym artykule jednak usystematyzowano połączenie poszczególnych modułów oraz pokazano, w jaki sposób możliwe jest szybkie potokowe łączenie wcześniej już wykonanych modułów zgodnych z architekturą OPiAr. Dodatkowo dodano magistralę kontrolną sterowaną przez procesor, co nie zmienia znacząco wyników implementacji podanych w poprzednich artykułach, a poprawia szybkość pracy i projektowania całego systemu.

Literatura

[1] Xilinx Inc. *Embedded Development Kit EDK 8.1 Embeded System Tools Reference Manual*. www.xilinx.com, Oct. 2005
[2] IBM Inc. *On-Chip Peripherial Bus, Application Specification v.2.1*. April 2001
[3] Xilinx Inc. *Embedded Processing* http://www.xilinx.com/products/design_resources/proc_central
[4] Jamro E.: *Synteza układów z parametrem w języku VHDL na przykładzie kodeka kodu BCH*. II Krajowa Konferencja Metody i systemy komputerowe w badaniach naukowych i projektowaniu inżynierskim, Kraków 25–17 X 1999, 39–43
[5] Wiatr K.: *Sprzętowe implementacje algorytmów przetwarzania obrazów w systemach wizyjnych czasu rzeczywistego*. Kraków, UWND AGH, 2002
[6] Jamro E., Wielgosz M., Wiatr K.: *Implementacja silnie zrównoleglonej operacji obliczania histogramu w układach FPGA*. Computer Methods and Systems 14–16 Nov. 2005, Kraków, Vol. II, 71–76
[7] Jamro E., Wielgosz M., Wiatr K.: *Implementacja Adaptacyjnego Kodera Huffmana w Układach FPGA.* Reprogramowalne Układy Cyfrowe, Szczecin 12–13 Maj 2005, 207–214