INFORMATYKA
CZASOPISMO TECHNICZNE
TECHNICAL TRANSACTION
COMPUTER SCIENCE AND INFORMATION SYSTEMS
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ
1-I/2011
ZESZYT 9
ROK 108
ISSUE 9
YEAR 108

STANISŁAW DENIZIAK, ROBERT TOMASZEWSKI*

KOSYNTEZA SYSTEMÓW WBUDOWANYCH O ARCHITEKTURZE SIECI JEDNOUKŁADOWYCH

CO-SYNTHESIS OF NETWORK-ON-CHIP EMBEDDED SYSTEMS

S t r e s z c z e n i e

W artykule przedstawiono metodę kosyntezy systemów wbudowanych, której celem jest znalezienie najtańszej architektury heterogenicznej spełniającej podane ograniczenia czasowe. W odróżnieniu od typowych podejść stosowanych w kosyntezie opisywana w pracy metodologia generuje kompletną strukturę komunikacyjną pomiędzy elementami przetwarzającymi. Realizacja bazuje na architekturze sieci jednoukładowej (ang. *Network on Chip*, *NoC*), gdzie topologia i ruting dobierane są w sposób eliminujący ewentualne kolizje między transmisjami. Dzięki temu opracowany sposób tworzenia sieci jednoukładowej zapewnia spełnienie ograniczeń czasowych nałożonych na projektowaną aplikację. Przeprowadzone eksperymenty dowodzą przewagi zaprezentowanego rozwiązania nad typowymi podejściami wykorzystywanymi w metodologiach budowania sieci NoC.

Słowa kluczowe: kosynteza, wieloprocesorowe systemy wbudowane, sieci jednoukładowe, komunikacja bezkolizyjna

A b s t a r c t

The paper presents an algorithm for embedded systems co-synthesis. The goal of the co-synthesis is to find the most cost-effective heterogenic architecture complying with execution time constraints. Additionally, presented methodology creates communication infrastructure for allocated and mapped processing elements with the use of Network-on-Chip (NoC) architecture. Dedicated topology along with pre-determined routing eliminate any communication contention. Experimental results prove superiority of our solution over state-of-the-art approaches on field of real-time embedded systems co-synthesis and custom NoC generation.

Keywords: co-synthesis, multiprocessor embedded systems, Network-on-Chip, contention-free communication

* Dr hab. inż. Stanisław Deniziak, mgr inż. Robert Tomaszewski, Katedra Informatyki, Wydział Elektrotechniki, Automatyki i Informatyki, Politechnika Świętokrzyska.

Artykuł złożony do Wydawnictwa: 10.07.2010

1. Wstęp

Funkcje systemów wbudowanych są zwykle specyfikowane w postaci komunikujących się procesów. W wyniku kosyntezy [1] specyfikacja systemu jest odwzorowana w architekturę składającą się z procesorów uniwersalnych, procesorów specjalizowanych oraz łączy komunikacyjnych. Celem kosyntezy jest znalezienie najlepszej architektury spełniającej zadane ograniczenia, np. minimalna szybkość, maksymalny koszt, minimalny pobór mocy.

W niektórych pracach związanych z kosyntezą [1–3] zakłada się uproszczoną architekturę systemu składającą się z jednego procesora uniwersalnego i jednego procesora specjalizowanego w postaci układu scalonego typu ASIC lub FPGA. Wtedy problem kosyntezy sprowadza się do problemu podziału funkcji pomiędzy sprzęt a oprogramowanie. Jednak w ogólnym przypadku systemy wbudowane składają się z wielu różnych procesorów, tzn. są rozproszonymi systemami heterogenicznymi. Wtedy kosynteza składa się z następujących zadań:

- alokacja modułów obliczeniowych i kanałów komunikacyjnych,
- przyporządkowanie zadań do procesorów i transmisji do kanałów komunikacyjnych,
- szeregowanie zadań i transmisji.

Alokacja polega na zdefiniowaniu architektury systemu. W tym celu konieczne jest określenie dostępnych modułów bibliotecznych wraz z ich parametrami (koszt, szybkość wykonywania poszczególnych zadań, itp.). Przyporządkowanie zadań do procesorów jest uogólnionym problemem podziału zadań pomiędzy sprzęt a oprogramowanie. Zadanie to jest ściśle związane z alokacją i zwykle zadania te są wykonywane równocześnie. Szeregowanie zadań polega na generacji sterowania czyli określenia kolejności wykonywania poszczególnych zadań. Większość metod kosyntezy [4–7] zakłada wielomagistralową architekturę docelową. Jednak w ostatnich latach wykazano, że w przypadku systemów rozproszonych zorientowanych na intensywne obliczenia, bardziej efektywne są architektury o strukturze sieci jednoukładowej NoC [8]. W podejściu tym zakłada się, że każdy moduł przetwarzający (ang. *Processing Element*, *PE*) komunikuje się z innymi modułami za pośrednictwem sieci mikroruterów. Ze względu na ograniczenia dotyczące powierzchni układu scalonego jak również jego poboru mocy wymaga się, by pojedynczy ruter miał maksymalnie prostą konstrukcję i zajmował nie więcej niż ok. 5% powierzchni układu scalonego. Skalę zapożyczeń ze świata sieci komputerowych, sposoby rutingu, przykładowe konstrukcje ruterów, przegląd założeń i rozwiązań dotyczących architektur NoC można znaleźć w [9]. Sieci jednoukładowe wydają się być architekturami przyszłości dla wieloprocesorowych systemów wbudowanych przetwarzających w sposób równoległy.

Wśród wielu problemów badawczych i konstrukcyjnych jednym z najważniejszych jest sposób odwzorowania aplikacji w system NoC z uwzględnieniem ograniczeń dotyczących szybkości pracy i poboru mocy oraz metody radzenia sobie z natłokiem komunikacyjnym (ang. *traffic contention/congestion control*) [10]. Systemy wbudowane, ze względu na ich specjalizowany charakter, w większości przypadków cechuje przewidywalny model obliczeń i transmisji [11, 12]. Takie aplikacje można zatem opisać grafem zadań (ang. *Task Graph*, *TG*) i zastosować w docelowej architekturze NoC prostszy w implementacji ruting deterministyczny. W pracy [12] przedstawiono algorytm genetyczny przyporządkowujący węzły grafu zadań do połączonych w topologię mesh elementów PE. Celem jest minimalizacja czasu przetwarzania systemu. Ruting transmisji odbywa się według najkrótszych

ścieżek, metoda szereguje również zadania i transmisje. W [11] przeznaczeniem odwzorowania grafu zadań w zadaną topologię mesh jest minimalizacja kosztu energetycznego transmisji i zadań oraz unikanie kolizji transmisyjnych. Metodą realizacji bezkolizyjności jest szeregowanie zadań i transmisji oraz wyznaczanie nieprzecinających się tras dla komunikatów. Autorzy [13] rozszerzają koncept z pracy [11] o etap doboru elementów przetwarzających (kosyntezę). Metodologia oparta na sekwencji algorytmów genetycznych do alokacji zadań na PE, szeregowania zadań i komunikatów oraz wyznaczeniu minimalnych ścieżek transmisyjnych została zaprezentowana w [14]. Celem jest minimalizacja poboru mocy przez układ. Graf zadań aplikacji jest odwzorowywany w topologię mesh. Autorzy proponują różnicowanie przepustowości łącz międzyruterowych poprzez zmianę ich napięć zasilających. Ponadto postulują odłączanie nieużywanych łącz, aby ograniczyć statyczny pobór mocy układu. Minimalizację globalnego ruchu w mikrosieci obrali za cel autorzy pracy [15]. Graf zadań odwzorowują w zadany graf sieci NoC za pomocą metod heurystycznych. W przeciwieństwie do poprzednich prac biorą pod uwagę topologie nieregularne (np. łącza jednokierunkowe) z heterogenicznymi elementami przetwarzającymi o różnych wymiarach w układzie. Do jednego zasobu PE może być przyporządkowane tylko jedno zadanie, dozwolony ruting – tylko minimalny. Każda z powyższych prac zakłada zadaną topologię docelowej mikrosieci – regularną bądź nie.

Często stosowanym podejściem w projektowaniu systemów NoC z uwzględnieniem wymagań komunikacyjnych jest tzw. rezerwacja pasma [16]. Polega ona na takim projektowaniu systemu NoC, by łącza komunikacyjne miały pewien zapas przepustowości na najbardziej niekorzystny przypadek obciążenia transmisjami. Osiąga się to poprzez różnicowanie szerokości magistral między ruterami jak również stosowanie różnych częstotliwości (szybkości) pracy dla poszczególnych obszarów sieci. Modelem wejściowym aplikacji jest wówczas jedna z odmian tzw. *Core Graph* [16–18], gdzie węzły to procesory, a krawędzie – łączące je, zwykle dwukierunkowe, magistrale. Węzłom nie przypisuje się wag jak w grafie zadań, jedynie krawędzie etykietowane są szybkościami wykonywanych transmisji (ilość przesyłanych danych w jednostce czasu). W pracy [16] zauważono, że dla systemów wbudowanych o przewidywalnym wzorcu komunikacji bardziej odpowiednie są dedykowane topologie, natomiast dla układów wieloprocesorowych ogólnego przeznaczenia – predefiniowane topologie regularne (siatka, pierścień, torus, drzewo binarne itp.). Jak wspomniano, unikanie kolizji i zarazem spełnienie kryteriów szybkościowych systemu najpowszechniej jest realizowane metodami rezerwującymi przepustowość. W [16] osiąga się to poprzez budowanie topologii dedykowanej, zastosowanie ruterów typu *multi-local port* (jeden ruter obsługuje niemal dowolnie wiele procesorów) oraz różnicowanie łącz. Rozwiązanie przedstawione w [18] generuje za pomocą metod symulowanego wyżarzania wstępną, podobną do listy dwukierunkowej, sieć połączeń, do której następnie dodawane są dedykowane łącza między wybranymi węzłami. Stosowane rutery mają jeden port lokalny, zaś protokół rutingu pozwala na przesyłanie komunikatów trasami nieoptymalnymi (tzw. ruting nieminimalny). Zarówno praca [16], jak i [18] nie poruszają kwestii szeregowania. Warto odnotować, że koncept dodawania lub usuwania łącz międzyruterowych stosuje się w celu poprawy szybkości [19] lub energooszczędności systemu [20].

Analiza uszeregowania czasowego transmisji pozwala przewidzieć wszelkie konflikty (kolizje) komunikacyjne. Konflikt taki to nakładanie się na siebie w tym samym czasie dwóch lub więcej przesyłanych wiadomości. Wyróżniamy trzy rodzaje kolizji komunikatów [21]:

– na porcie źródłowym – w tym samym czasie jeden element PE próbuje wykonać więcej niż jedną transmisję,
– na porcie docelowym – ma miejsce, gdy dwie lub więcej wiadomości w tym samym czasie zmierza do tego samego PE,
– na trasie – sytuacja, gdy dwie lub więcej transmisji podąża trasami współdzielącymi te same połączenia międzyruterowe.

Wiedza o potencjalnych konfliktach pozwala na ich wczesną eliminację bez uciekania się do technik bazujących na buforowaniu w ruterach i/lub różnicowaniu przepustowości między ruterami w celu przyspieszenia/zwolnienia transmisji. W prezentowanej w niniejszym artykule metodologii kosynteza eliminuje wszelkie konflikty na portach poprzez ich szeregowanie, natomiast etap generowania dedykowanej architektury NoC usuwa kolizje na trasie. Realizowane jest to dzięki przydzieleniu odrębnych tras dla nakładających się w czasie transmisji. Ponadto etap budowania sieci ma możliwość dodatkowego przeszeregowania transmisji. Może to zrobić pod warunkiem zachowania ograniczeń czasowych nałożonych na zadania systemu.

Wady podejścia opartego na rezerwacji pasma w projektowaniu architektur NoC w porównaniu z analizą czasową możliwych konfliktów komunikacyjnych wykazano również w pracy [22]. Porównano tam dedykowane pod względem topologii systemy zaprojektowane według obu zasad. Przewaga wydajnościowa rozwiązania uwzględniającego kolizje była bezdyskusyjna. Jednakże metoda zastosowana w [22] nie jest pozbawiona braków – przede wszystkim zastosowanie rutingu minimalnego (najkrótsze ścieżki) zmniejsza elastyczność trasowania, będącego atutem sieci połączeń. Ponadto rozwiązywanie konfliktów poprzez dublowanie ruterów (typu *multi-local port*) dla spornych fragmentów sieci spowodowało średnio 25% wzrost energo- i zasobochłonności systemu. Dodatkowo powyższe podejście nie wykorzystuje możliwości szeregowania zadań i transmisji. Również w pracy [17] podczas tworzenia dedykowanej topologii autorzy kierują się bezpośrednio wiedzą o konfliktach komunikacyjnych uzyskaną dzięki analizie czasowej systemu. Dla kolidujących transmisji wyznaczają osobne trasy. Jednak ich podejście nie eliminuje całkowicie zatorów w mikrosieci, a tylko je minimalizuje. Wykorzystana przez nich konstrukcja rutera jest wysoce nieregularna, jeśli chodzi o liczbę portów lokalnych oraz przepustowość łącz międzyruterowych. Jest to również jeden z niewielu przypadków, gdzie zezwala się, by dwa rutery były połączone więcej niż jednym dwukierunkowym łączem międzyruterowym.

Kosynteza systemów wbudowanych o architekturze NoC wymaga opracowania nowych metod uwzględniających zarówno problemy alokacji modułów PE oraz szeregowania zadań jak i problemy alokacji kanałów komunikacyjnych i szeregowania transmisji w mikrosieci. W pracy zaprezentowano nową metodę kosyntezy rozproszonych systemów heterogenicznych o architekturze NoC. Celem metody jest znalezienie najtańszej architektury spełniającej podane ograniczenia czasowe, minimalizowany jest również pobór mocy związany z przesyłaniem komunikatów. Na wstępie wykonywana jest kosynteza systemu, zakładająca transmisje bezkolizyjne. Następnie dla tak wygenerowanej architektury stosowany jest algorytm generujący dedykowaną topologię sieci oraz szeregujący transmisje w taki sposób, aby uzyskać sieć o jak najmniejszej liczbie połączeń i jak najmniejszym poborze mocy. Kostynteza jest wykonywana za pomocą rafinacyjnego algorytmu EWA [23] przystosowanego do syntezy architektury NoC. Do generacji bezkolizyjnej sieci NoC stosowana jest ulepszona wersja metody [24]. Wśród poprawek znalazły się m.in. ścisłe powią-

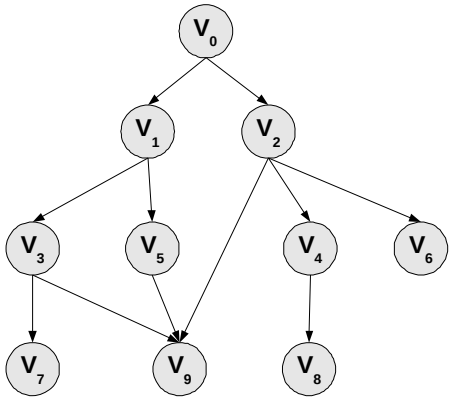
zanie z kosyntezą (nawroty metody) oraz optymalizacja zastosowanych algorytmów i struktur danych wykorzystywanych w pośrednich etapach metodologii. Według aktualnej wiedzy autorów niniejsza praca jest pierwszą, która łączy możliwości stwarzane przez topologie nieregularne z szeregowaniem transmisji i zadań aplikacji dla stworzenia wolnego od kolizji oraz efektywnego pod kątem zapotrzebowania na zasoby i energię systemu o architekturze sieci jednoukładowej.

Zaletami prezentowanej metody są: całkowita eliminacja konfliktów transmisyjnych, bardzo prosta i efektywna konstrukcja rutera (ruting deterministyczny z buforem jednoflitowym, bez kanałów wirtualnych, z metodą przełączania pakietów typu *wormhole* [9]), minimalna topologia połączeń (w sensie liczby łącz międzyruterowych) dla danego uszeregowania systemu, spełniająca ograniczenia szybkości pracy systemu oraz powiązanie metod kosyntezy z tworzeniem architektury komunikacyjnej dla rafinacji rozwiązania.

W następnym rozdziale zostanie przedstawiony przyjęty model systemu i cel kosyntezy. W rozdziale 3 zostanie opisana proponowana metodologia kosyntezy systemów NoC. Rozdział 4 zawiera wyniki wykonanych eksperymentów. Na zakończenie zostaną przedstawione wnioski i kierunki dalszych prac.

2. Sformułowanie problemu

Abstrakcyjnym modelem specyfikacji funkcji systemu jest graf zadań $G = (V, E)$. Graf zadań jest grafem skierowanym i acyklicznym [25]. Węzły v_i odpowiadają zadaniom, natomiast krawędzie e_{ij} opisują komunikację pomiędzy zadaniami odpowiadającymi węzłom v_i i v_j. Wagi d_{ij} związane z krawędziami opisują rozmiar transmisji pomiędzy komunikującymi się zadaniami. Rysunek 1 przedstawia przykładowy graf zadań.



Rys. 1. Przykładowy graf zadań

Fig. 1. Sample task graph

Wyróżniamy dwa rodzaje zasobów: procesory uniwersalne (*PP*) i procesory specjalizowane (*HC*). *PP* są traktowane jako jeden element obliczeniowy *PE*, który wykonuje sekwencyjnie wszystkie przydzielone mu zadania. *HC* jest wyspecjalizowanym

modułem sprzętowym wykonującym tylko jedno zadanie. Z każdym zasobem R_i związany jest koszt jednostkowy CU_i, niezależny od liczby zadań wykonywanych przez dany zasób. Koszt ten uwzględnia: koszt procesora, koszt interfejsu sieciowego i koszt rutera. Ponadto z każdym zadaniem związane są wektory $C_j(i)$ i $T_j(i)$ określające:

– $C_j(i)$: koszt wykonania zadania v_j przez zasób R_i,
– $T_j(i)$: czas wykonywania zadania v_j, przez zasób R_i.

Wartości parametrów $C_j(i)$ i $T_j(i)$ mogą być obliczane przy użyciu istniejących metod oszacowań na najgorszy przypadek [26]. Przykładowe parametry procesorów dla grafu z rysunku 1 są przedstawione w tabeli 1.

Tabela 1

Parametry procesorów

	R_1 (*PP*)		R_2 (*PP*)		R_3 (*HC*)		R_4 (*HC*)	
	$CU_1 = 100$		$CU_2 = 200$		$CU_3 = 500$		$CU_4 = 300$	
	T_1	C_1	T_2	C_2	T_3	C_3	T_4	C_4
V_0	30	3	10	2	3	50	4	10
V_1	50	5	20	4	6	80	5	20
V_2	20	3	10	3	3	60	5	20
V_3	10	3	8	1	1	20	2	5
V_4	30	3	15	2	4	70	10	30
V_5	50	5	30	3	5	80	5	15
V_6	40	3	15	2	10	70	12	15
V_7	30	3	15	2	5	50	8	18
V_8	20	3	5	1	2	30	4	10
V_9	10	3	5	1	3	40	4	12

Dane pomiędzy zasobami przesyłane są poprzez łącza komunikacyjne, docelowo implementowane jako trasy w sieci NoC. Podczas kosyntezy zakłada się, że topologia będzie zapewniać transmisje bezkolizyjne, tzn. z maksymalną szybkością. Dlatego przyjmuje się czas transmisji T_{ij} pomiędzy zadaniami v_i i v_j proporcjonalny do ilości danych (przy zadanej przepustowości łączy). Ponadto zakłada się, że wszystkie transmisje do/z procesora są wykonywane sekwencyjnie.

Dla potrzeb etapu budowania dedykowanej sieci połączeń o architekturze NoC został stworzony kompleksowy model. Bazuje on na rozwiązaniach podanych w literaturze dotyczącej sieci jednoukładowych oraz kosyntezy. Na model ten składają się poniższe elementy:

Atrybutowany Graf Zadań (ang. *Attributed Task Graph*, *ATG*) – jest efektem procesu kosyntezy przeprowadzonej dla danego grafu zadań TG. Oznacza to, że każdy węzeł grafu i krawędź ATG mają przypisany w wyniku uszeregowania czas rozpoczęcia. Ponadto zadania grafu są przyporządkowane do konkretnych elementów przetwarzających PE. Zakłada się, że wagi krawędzi opisane wielkościami transmisji zarazem reprezentują ich czas, mierzony w cyklach zegarowych. Założenie to wynika z faktu, że przy przełączaniu typu *wormhole* typowe opóźnienie rutera NoC to 1–2 cykle zegarowe, zaś liczba ruterów (hopów) na potencjalnej trasie rzadko przekracza 3. Z tej przyczyny w systemach ukierun-

kowanych na obliczenia i przesyłanie dużych ilości danych opóźnienie na trasie można pominąć.

Ruter – metodologia korzysta z rutera zdolnego do nawiązania wielu dwukierunkowych połączeń pomiędzy swoimi portami [27]. Ruter stosuje zalecane dla NoC przełączanie typu *wormhole* oraz ruting deterministyczny z trasą zapisaną w nagłówku każdego komunikatu (ang. *source routing*) [9]. Wyposażony jest w jeden port lokalny (ang. *Local Port, LP*), podłączony poprzez interfejs sieciowy NI (ang. *Network Interface*) do jednego modułu PE, oraz w maksymalnie cztery dwukierunkowe porty służące do tworzenia połączeń międzyruterowych (w kierunkach: Północ, Południe, Wschód, Zachód). W mikrosieci niedozwolone jest występowanie ruterów niepodłączonych do żadnego PE. W związku z brakiem kolizji w ostatecznym rozwiązaniu porty wejściowe rutera mogą posiadać co najwyżej jednoflitowe bufory. Wszystkie łącza międzyruterowe mają jednakowe parametry transmisyjne (jednakową przepustowość).

Szeregowanie, przeszeregowanie – szeregowanie oznacza przyporządkowanie czasów rozpoczęcia dla zadań i transmisji na etapie kosyntezy. Krok ten eliminuje wszelkie konflikty na portach źródłowych i docelowych. Dla kolizji na trasie nie można tego zrobić, gdyż do tego celu niezbędna jest znajomość topologii połączeń. W przypadku niemożności znalezienia bezkolizyjnej trasy lub wygenerowania dedykowanego połączenia na etapie tworzenia NoC, metodologia dokonuje próby przeszeregowania danej transmisji. W praktyce oznacza to opóźnienie czasu jej rozpoczęcia. Taka akcja jest dozwolona pod warunkiem zachowania ograniczeń czasowych dla systemu (nieprzekraczalne czasy wykonania wszystkich lub wybranych zadań) po przeszeregowaniu.

Graf Topologii Sieci (ang. *Network Topology Graph*, *NTG*) – graf skierowany nieważony, reprezentuje sieć połączeń (krawędzie) między procesorami (węzły). W momencie rozpoczęcia działania metodologii NTG jest pusty i sukcesywnie rozbudowywany metodą zachłanną (szuka się najlepszej w danym momencie ścieżki lub dodatkowego połączenia). Gdy metoda kończy działanie, graf NTG wraz z wyznaczonymi trasami oraz ostateczne uszeregowanie obliczeń i transmisji tworzą kompletne rozwiązanie.

Model ponadto przyjmuje, że każdy moduł PE posiada własną pamięć dla programu i danych. Model programowo-komunikacyjny aplikacji to przekazywanie wiadomości (ang. *message-passing*). Wobec powyższego założono, że transmisje między zadaniami przypisanymi do tego samego PE (tzw. intraprocesorowe) mają zerowy czas wykonywania. Interfejs NI umożliwia jednoczesny odbiór i nadawanie przez PE, ale nie zezwala na przeplot komunikatów. Oznacza to, że w dowolnej chwili nadawać/odbierać można tylko jedną wiadomość.

Do porównania z innymi strategiami tworzenia sieci NoC definiujemy następujące metryki, służące oszacowaniu jakości uzyskanego rozwiązania:

- czas działania systemu – intuicyjnie im dłuższy tym słabsza wydajność systemu; ten parametr ma wpływ również na składową statyczną pobieranej mocy w sieciach NoC,
- liczba użytych przez topologię łącz międzyruterowych – każde łącze międzyruterowe to zarezerwowany kolejny zasób układu jak również pobierana dodatkowa moc [20]; ponadto im prostsza struktura połączeń tym łatwiejsza implementacja układowa,
- średnia ważona skoków między ruterami dla całości komunikacji między procesorami – ten parametr ma wpływ zarówno na wydajność systemu, jak i jego energochłonność; dłuższe trasy oznaczają większe opóźnienia podczas przesyłu komunikatów i wpływają negatywnie na dynamiczną składową mocy w systemach NoC.

Do ostatniego kryterium średnia ważona skoków została zdefiniowana w taki sposób, by większe transmisje przesyłane dłuższymi trasami miały większy wpływ na ostateczny wynik niż mniejsze korzystające z krótszych dróg. Opisuje to poniższa formuła

$$hop_{AVG} = \frac{\sum_{1}^{n}(M_{i,j} \cdot hop(M_{i,j}))}{\sum_{1}^{n}(M_{i,j})} \tag{1}$$

gdzie:

$|M_{i,j}|$ – wielkość transmisji (mierzona najczęściej we flitach),
$hop(M_{i,j})$ – liczba ruterów/łącz na trasie danego komunikatu,
n – liczba wszystkich transmisji międzyprocesorowych w systemie.

Załóżmy że koszt implementacji jednego połączenia w sieci NoC wynosi C_L. Dla tak określonych parametrów koszt systemu możemy zdefiniować następująco

$$C = \sum_{i=1}^{r}\left(Cu_i + \sum_{M(i)} C_i(v_{M(i)}) \right) + c * C_L \tag{2}$$

gdzie:

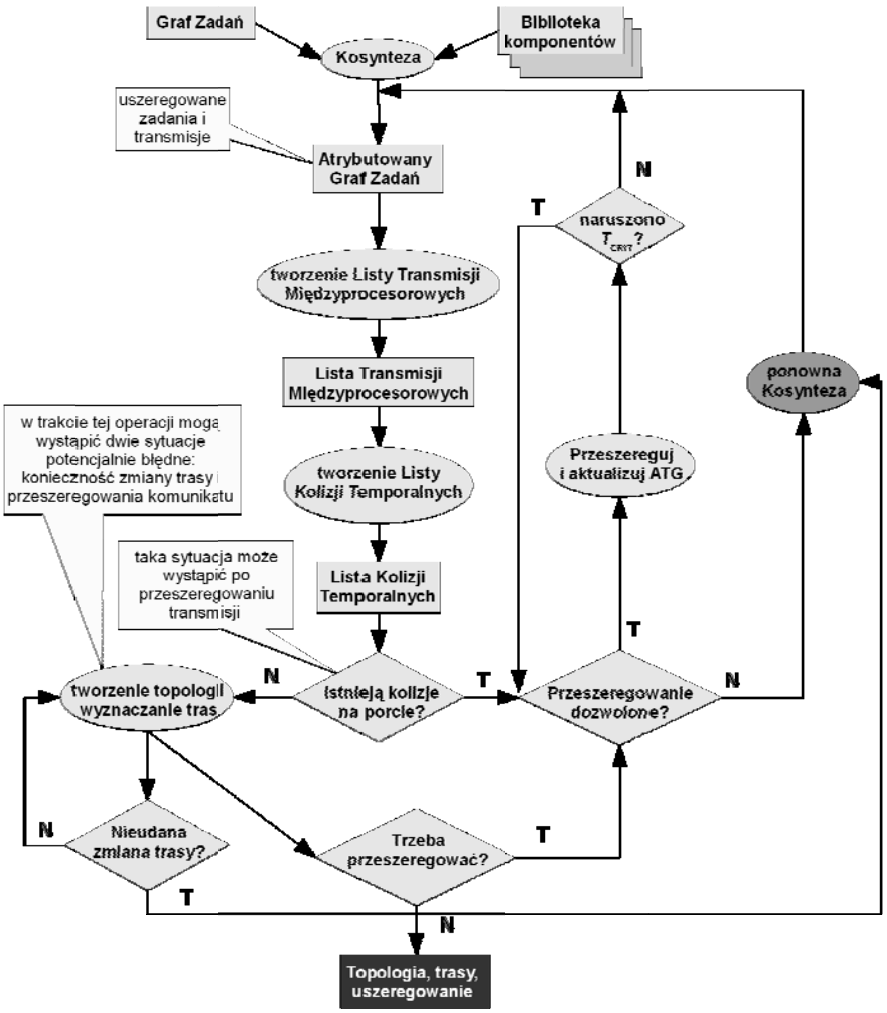
r – liczba zasobów,
$M(i)$ – numery zadań przydzielonych zasobowi R_i,
c – liczba połączeń w sieci NoC.

3. Metodologia

3.1. Schemat metodologii

Na rysunku 2 przedstawiono sieć działań podejmowanych w ramach prezentowanej metodologii kosyntezy systemów wbudowanych o architekturze sieci jednoukładowej.

W pierwszym kroku wykonywana jest kosynteza systemu specyfikowanego w formie grafu TG. Kosynteza tworzy graf ATG zoptymalizowany pod kątem minimalnego kosztu. Następnie graf ATG jest przetwarzany pod kątem wyodrębnienia wszystkich transmisji międzyprocesorowych. Rezultatem jest lista ITL (ang. *Interprocessor Transmission List*). Każda pozycja na liście opisana jest informacjami o procesorach: nadawczym i odbiorczym, czasie rozpoczęcia, czasie trwania oraz trasie (początkowo pusta – NULL). Na podstawie ICL powstaje, stanowiąca sedno modelu kolizji, lista kolizji transmisyjnych TCL (ang. *Transmission Collision List*). Elementami TCL są pary nachodzących na siebie w czasie wiadomości. W każdej parze najpierw umieszczany jest dłuższy ze skonfliktowanych komunikatów, a potem krótszy. Dzięki temu algorytm w dalszych krokach podejmuje korzystniejsze decyzje – wyznacza krótszą trasę lub dedykowane połączenie między modułami PE dla dłuższych transmisji. W kolejnym kroku sprawdza się obecność kolizji na portach wśród pozycji TCL. Mimo, że kosynteza szereguje komunikaty wykluczając tego typu kolizje, to mogą się one pojawić na skutek przeszeregowania podczas kolejnych iteracji algorytmu. Jeśli algorytm stwierdzi powstanie kolizji na porcie – obojętnie: źródłowym



Rys. 2. Sieć działań metodologii generującej bezkolizyjne sieci jednoukładowe

Fig. 2. Design flow of contention-free NoC generation

lub docelowym – to jedynym sposobem jej usunięcia jest przeszeregowanie. Może ono doprowadzić do naruszenia któregoś z zadanych przez projektanta systemu ograniczeń czasowych aplikacji (T_{CRIT}), co skutkuje próbą przeszeregowania drugiej z kolidujących wiadomości. Porażka i w tym przypadku doprowadzi do ponownej kosyntezy z dodatkowym ograniczeniem czasowym wymuszającym wcześniejsze uszeregowanie transmisji powodującej naruszenie T_{CRIT}. Algorytm przeszeregowania opóźnia tą z wiadomości, która w mniejszym stopniu pogarsza czasy zakończenia zadań z ustanowionymi ograniczeniami czasowymi (ale bez ich przekroczenia). W ostatnim kroku budowana jest topologia połączeń dla architektury NoC.

Proces ten opisano dalej, w osobnym podrozdziale.

3.2. Kosynteza

Ideą algorytmu kosyntezy jest iteracyjna rafinacja kolejnych rozwiązań, startując od pewnego rozwiązania początkowego. W każdym kroku algorytmu badane są różne możliwości modyfikacji systemu, a do następnego kroku wybierane jest rozwiązanie dające największy zysk (to znaczy rozwiązanie, które jest najlepsze według przyjętej miary jakości). Podstawowymi elementami algorytmu są:
- rozwiązanie początkowe,
- sposób modyfikacji (rafinacji),
- miara określająca zysk z modyfikacji.

Jako rozwiązanie początkowe tworzony jest system o największej szybkości. Każde zadanie przydzielone jest do innego procesora (najszybszego dla danego zadania). Algorytm nie generuje struktury połączeń, transmisje są szeregowane przy założeniu, że każdy procesor transmituje dane w sposób sekwencyjny z maksymalną szybkością. W przykładzie z rysunku 1 początkowa architektura systemu będzie się składała z 10 procesorów ($8 \times R_3$ oraz $2 \times R_4$). Koszt początkowy systemu będzie więc wynosił 5025, a maksymalna szybkość obliczeń 16.

3.2.1. Miara zysku

Zysk jest parametrem określającym jakość danej modyfikacji i decydującym o wyborze rozwiązań w kolejnych krokach algorytmu. Od sposobu zdefiniowania funkcji zysku zależy zatem zbieżność algorytmu. Zachłanna funkcja zysku (np. spadek kosztu lub wzrost szybkości systemu) zwykle prowadzi do wyboru skrajnych modyfikacji tzn. do alokacji najtańszych lub najszybszych zasobów. Dlatego w praktyce stosuje się bardziej złożone funkcje zysku. W algorytmie [28] zysk został zdefiniowany jako stosunek wzrostu szybkości do wzrostu kosztu. Jednak okazuje się, że tak zdefiniowany zysk nie zapewnia zbieżności rafinacji, ponadto użyte kryterium nie zawsze pozwala spełnić ograniczenia czasowe.

W prezentowanym algorytmie zysk został zdefiniowany w taki sposób, aby oprócz minimalizacji kosztu stwarzał również możliwości efektywnej rafinacji projektu w następnych krokach. Dlatego w funkcji zysku bierze się również pod uwagę wzrost przestrzeni rafinacji. W tym celu obliczane są następujące współczynniki:
- S_i – najwcześniejsza chwila rozpoczęcia wykonywania i-tego zadania,
- L_i – najpóźniejsza chwila rozpoczęcia wykonywania i-tego zadania, taka że wykonanie wszystkich następnych zadań nie naruszy ograniczeń czasowych.

Jeśli dla jakiegoś zadania $L_i < S_i$, to oczywiście rozwiązanie nie spełnia wymagań czasowych. Warunek ten jest sprawdzany w każdym kroku algorytmu. Zwykle im większa wartość $L_i - S_i$, tym są większe możliwości rafinacji związanej z i-tym zadaniem. Załóżmy, że dla grafu z rysunku 1 szukamy najtańszego rozwiązania wykonującego obliczenia w czasie $T_{\max} = 50$. Wtedy wartości L_i *i* S_i dla rozwiązania początkowego przedstawia tabela 2.

Tabela 2

Parametry L_i i S_i

	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
S	0	3	3	8	6	8	6	9	10	13
L	34	37	37	44	44	42	40	45	48	47

Podczas działania algorytmu zmienia się czas wykonywania zadań oraz ich uszeregowanie w związku z tym zmieniają się także współczynniki L_i i S_i.

Zysk ΔE uwzględniający spadek kosztu oraz swobodę modyfikacji systemu jest zdefiniowany w następujący sposób

$$\Delta E = \begin{cases} \dfrac{-\Delta K_s}{-\Delta\Omega}, & \text{dla} \quad \Delta\Omega < 0 \\ -\Delta K_S, & \text{dla} \quad \Delta\Omega = 0 \\ -\Delta K_S \cdot \Delta\Omega, & \text{dla} \quad \Delta\Omega > 0 \end{cases} \tag{3}$$

gdzie:

$$\Omega = \sum_{i=1}^{n} (L_i - S_i)$$

– ΔK_S oznacza wzrost kosztu systemu (nie uwzględniający kosztu połączeń),
– $\Delta\Omega$ określa wzrost obszaru swobody rafinacji projektu w następnych krokach algorytmu.

Tak zdefiniowany zysk określa, że z rozwiązań o takim samym spadku kosztu zostanie wybrane rozwiązanie o większym współczynniku „swobody" w rafinacji zadań (co daje większe możliwości w alokacji zasobów w następnych krokach algorytmu). Ponadto parametr ten „hamuje" przydział zadań do wolnych zasobów (i jednocześnie o niskim koszcie), który daje duży zysk kosztu, ale znacznie zwalnia cały system.

3.2.2. Rafinacja systemu

W każdym kroku algorytmu rozpatrywane są różne modyfikacje systemu, wybierana jest modyfikacja dająca największy zysk. Zbyt duża liczba analizowanych modyfikacji znacznie zwiększyłaby złożoność algorytmu. Dlatego w praktyce rozpatruje się tylko proste modyfikacje np. polegające na przenoszeniu zadań z zasobów wykonujących najmniej zadań do zasobów, które nie są w pełni wykorzystane [28]. W ten sposób można zmniejszyć liczbę zasobów w systemie.

W odróżnieniu od istniejących algorytmów kosyntezy rozpatrujących modyfikacje systemu z punktu widzenia zadań, prezentowany algorytm rozpatruje modyfikacje z punktu widzenia architektury tzn. zamiast przydziału zadań w pierwszej kolejności modyfikowana jest alokacja zasobów. W jednym kroku algorytmu wykonywana jest modyfikacja systemu polegająca na:

1. Dodaniu jednego zasobu do systemu i przyporządkowaniu mu maksymalnej liczby zadań w kolejności od największego zysku (w przypadku gdy każde przesunięcie zadania powoduje wzrost kosztu systemu to nie zostaną mu przyporządkowane żadne zadania). Następnie usuwane są zasoby nie używane, tj. którym zostały odebrane wszystkie zadania.
2. Usunięciu zasobu. Dla każdego zasobu algorytm próbuje rozdzielić przyporządkowane mu zadania pomiędzy pozostałe zasoby. Tutaj też algorytm kieruje się największym zyskiem lub najmniejszą stratą (dopuszcza się również przesunięcia dla których zysk < 0). W ten sposób możliwe jest usunięcie z systemu, droższego zasobu ale o tańszych zadaniach. W tym kroku liczy się tylko ostateczny koszt systemu.

Połączenie dodania i usunięcia zasobu w jeden krok algorytmu daje dodatkowe możliwości modyfikacji systemu. Na przykład możliwe jest, aby w jednym kroku został dodany nowy zasób, zwiększając koszt systemu, a następnie inny zasób został usunięty powodując ostatecznie spadek kosztu. W ten sposób algorytm ma możliwość wydobywania się z lokalnych minimów kosztu.

W każdym kroku rozpatrywane są wszystkie możliwości dodania i usunięcia jednego zasobu, które nie naruszają ograniczeń czasowych. Wybierana jest modyfikacja dająca największy zysk. Algorytm kończy działanie gdy nie ma możliwości modyfikacji systemu dla której zysk jest większy od 0.

3.2.3. Szkic algorytmu kosyntezy

Szkic algorytmu kosyntezy przedstawia się następująco:

Utwórz architekturę początkową A systemu;
Oblicz koszt początkowy systemu K_s*;*
repeat
Zysk = 0;
for *każdy zasób* P_i ***do***
  $A' = A \quad P_i$*;*
  repeat
    Znajdź zadanie v_k *dla którego* E *jest największe po przesunięciu go do zasobu* P_i*;*
    if $E > 0$ ***then*** *Przydziel zadanie* v_k *do zasobu* P_i
  until *nie istnieje zadanie dla którego* $E > 0$*;*
  $A' = A'$ *– zasoby nie używane;*
  if $E > Zysk$ ***then***
    $Zysk = E$; $A^{best} = A'$*;*
  endif*;*
  for *każdy zasób* $P_j \quad A'$ ***do***
    $A'' = A' - P_j$*;*
    for *każde zadanie* $v_k \in P_j$ ***do***
    Znajdź zasób $P_l \in A''$ *dla którego* E *jest największe, po przesunięciu do niego zadania* v_k*;*
    Przydziel zadanie v_i *do zasobu* P_l
    endfor*;*
  if $E > Zysk$ ***then***
    $Zysk = E$; $A^{best} = A''$*;*
  endif*;*
  endfor
  endfor
  if $Zysk > 0$ ***then*** $A = A'$*;*
until $Zysk = 0$*;*

Gdzie δE oznacza zysk po przesunięciu jednego zadania do nowego zasobu (bez uwzględnienia kosztu jednostkowego), natomiast ΔE oznacza zysk po całkowitej modyfi-

kacji systemu. Zewnętrzna pętla ***for*** sprawdza wszystkie możliwości dodania zasobu do systemu. Przesunięcie zadań do nowo dodanego zasobu odbywa się w wewnętrznej pętli ***repeat***. Z tak powstałej architektury A' dokonywana jest próba usunięcia jednego zasobu (wewnętrzne pętle ***for***). Jeśli uda się znaleźć zasób którego usunięcie da większy zysk to zasób ten jest usuwany i powstaje architektura A''. W każdym kroku algorytmu zapamiętywana jest architektura A^{best}, dla której uzyskano największy zysk. Architektura ta jest architekturą początkową w następnym kroku. Kryterium dodatniego zysku zapewnia zbieżność algorytmu.

W przypadku przydziału zadań niezależnych (leżących na równoległych ścieżkach w grafie) do tego samego procesora uniwersalnego, konieczne jest określenie kolejności wykonywania tych zadań. W w/w algorytmie wybiera się taką kolejność dla której jest większy zysk (mniejsza wartość $\Delta\Omega$). W ten sposób podane kryterium określa również sposób szeregowania zadań. Wybór kolejności jest wykonywany poprzez sprawdzenie wszystkich możliwych przypadków, ale nie zmienia się kolejności zadań wcześniej przydzielonych. Podobnie wykonywane jest szeregowanie transmisji, w pierwszej kolejności wykonywane są transmisje dla których wartość Ω jest najmniejsza.

3.3. Generowanie topologii połączeń NoC

Po usunięciu kolizji na portach następuje etap generowania topologii połączeń dla pozostałych kolizji wraz z ewentualnymi przeszeregowaniami. W pierwszym przebiegu metody, zaraz po kosyntezie, ten krok tworzy połączenia tylko dla kolizji na trasie, gdyż podczas alokacji, odwzorowania i szeregowania kolizje na portach usuwa algorytm kosyntezy. Szczegółowy opis poszczególnych kroków generacji topologii NoC jest przedstawiony w [24]. Poniżej zaprezentowano jedynie ogólne zasady.

Dla każdej pozycji z listy TCL:

– jeżeli dla danych transmisji nakładających się w czasie już przypisano trasy (usunięto inne kolizje, w których brały udział rozpatrywane komunikaty), dokonywane jest sprawdzenie czy nie są to kolidujące trasy, jeśli tak – podejmowana jest próba modyfikacji (często – pogorszenia, wydłużenia) trasy dla krótszej z transmisji, a w przypadku niepowodzenia – dla dłuższej; zmiana wyznaczonych tras wymaga sprawdzenia bezkolizyjności uprzednio usuniętych konfliktów, niepowodzenie tej operacji prowadzi do ponownej kosyntezy; jednocześnie przekazywana jest informacja o zadaniu-nadawcy krótszego z komunikatów (wraz z czasem o jaki przekroczono ograniczenie czasowe), dla którego nie udało się zmienić trasy – zadanie takie podczas ponownej kosyntezy będzie miało ustanowione bardziej rygorystyczne ograniczenie czasowe (skrócone o przekazany czas),
– jeżeli dla danej transmisji nie wyznaczono dotąd trasy – to sprawdza się – czy w już utworzonej topologii istnieje najkrótsza droga dla komunikatu; to samo dotyczy drugiej kolidującej transmisji; w przypadku kolizji tras drugi komunikat będzie miał wyznaczoną inną niż najkrótsza drogę,
– niepowodzenie poprzedniego kroku skutkuje próbą wygenerowania kolejnego, dedykowanego połączenia międzyruterowego; rozbudowa topologii podlega ograniczeniom – jeden ruter może mieć nie więcej niż 4 łącza wejściowe i 4 wyjściowe,
– jeżeli poprzednie kroki okazały się niewystarczające dla danej pozycji na liście TCL (nie da się rozwiązać danego konfliktu za pomocą dedykowanej topologii), podejmo-

wana jest próba przeszeregowania komunikatów opisana wcześniej – jej niepowodzenie spowoduje ponowną kosyntezę z ustanowieniem nowego, krótszego ograniczenia czasowego dla zadania-nadawcy mniejszej z transmisji.

Po przetworzeniu wszystkich pozycji z listy TCL sprawdza się, czy lista ITL zawiera węzły nieuwzględnione w TCL, czyli transmisje niekolidujące z żadnymi innymi. Jeśli tak podejmowana jest próba wyznaczenia najkrótszej drogi dla każdej brakującej transmisji (może to również oznaczać konieczność wstawienia dodatkowego połączenia lub węzła do sieci).

4. Wyniki wykonanych eksperymentów

Ze względu na brak analogicznych metod kosyntezy dedykowanych architektur NoC nie jest możliwe porównanie efektywności zaproponowanej metody poprzez porównanie wyników eksperymentów. Skuteczność algorytmu EWA dla architektur wielomagistralowych została wykazana w [23], uzyskane wyniki okazały się znacznie lepsze od wyników uzyskanych przy zastosowaniu innych znanych z literatury metod. Dlatego w pracy tej ograniczono się do eksperymentów oceniających jakość uzyskanych rozwiązań, eksperymenty te przede wszystkim pokazują efektywność metody generacji topologii sieci NoC.

Eksperymenty szacujące jakość otrzymanej architektury NoC przeprowadzono metodami analitycznymi. Podobnie jak w pracach [12, 13, 15] wygenerowano narzędziem podobnym do TGFF [29] sześć rozbudowanych, syntetycznych grafów zadań zróżnicowanych pod kątem natężenia obliczeniowo-komunikacyjnego. Parametry aplikacji przedstawiono w tabeli 3. Przedostatnia kolumna wskazuje na prawdopodobieństwo wystąpienia kolizji w danym systemie (im mniejszy współczynnik tym wyższe prawdopodobieństwo), natomiast ostatnia informuje o poziomie natłoku komunikacyjnego (im wyższy współczynnik tym większy natłok). Natłok rozumiany jest jako ilość jednocześnie transmitowanych wiadomości w mikrosieci.

Tabela 3

Parametry badanych aplikacji

Aplikacja	Liczba zadań	Liczba transmisji	Liczba PE	Czas obliczeń/ czas transmisji	Czas kolizji/ czas transmisji
Graf01	33	42	9	0,56	0,24
Graf02	34	31	12	1,07	0,21
Graf03	37	41	14	0,92	0,88
Graf04	33	26	12	1,5	0,39
Graf05	32	28	12	1,1	0,72
Graf06	35	46	10	1,01	0,84

Następnie porównano czasy wykonania systemów odwzorowanych w różne architektury docelowe: oparte na wspólnej magistrali oraz na mikrosieci. Dodatkowo do porównania różnych podejść w generowaniu docelowej topologii NoC wzięto pod uwagę następujące rozwiązania: topologia nieregularna z dedykowanymi połączeniami eliminującymi kolizje generowanymi dla list TCL posortowanych według wielkości kolizji (WWK) oraz

według kolejności występowania kolizji w czasie (WKK), topologia regularna typu mesh (losowo generowane kwadratowy NxN oraz minimalny prostokątny MxN), topologia nieregularna tworzona według zasady rezerwacji pasma, podobna do rozwiązań [16, 18], ale z dodatkowymi dedykowanymi jednokierunkowymi łączami (bndwdth+łacza) dla usunięcia ewentualnych kolizji lub bez nich (bndwdth+przeszeregowanie). Drugi wariant wymaga przeszeregowania kolidujących transmisji, gdyż przepustowość łącz międzyruterowych musi być jednakowa dla ujednolicenia porównań. Podobne założenie przyjęto w przypadku obu topologii mesh oraz wspólnej magistrali – wszelkie kolizje transmisji usuwano przeszeregowaniem. Topologie „bndwdth" łączą ze sobą bezpośrednio (magistralami dwukierunkowymi) procesory wymieniające największe ilości danych. Wyniki porównania szybkości działania systemów przedstawiono w tabeli 4. Referencyjnym systemem jest uszeregowany przez kosyntezę graf zadań. Zaproponowane rozwiązanie (WWK) dało najlepsze rezultaty dla wszystkich porównywanych systemów – dwóch przypadkach wynik był identyczny z wynikiem kosyntezy (zerowe pogorszenie). Najsłabiej wypadła architektura magistralowa. Co ciekawe, dedykowana dla zysku wydajności topologia typu „bndwdth" dla żadnego z systemów nie stanowiła rozwiązania lepszego od WWK.

Tabela 4

Pogorszenie (procentowo) czasów wykonania różnych architektur względem wyniku kosyntezy

Aplikacja	Wspólna magistrala [%]	Network on Chip [%]				
		WWK, WKK	mesh NxN	mesh MxN	bndwdth+ łącza	bndwdth+ przeszeregowanie
Graf01	20,9	0,43	3,54	8,33	0,43	3,91
Graf02	19,38	0,42	0,42	0,42	0,42	0,42
Graf03	39,53	4,28	15,8	12,58	4,28	9,69
Graf04	30,69	4,33	29,85	27,19	0,05	12,41
Graf05	51,09	0	0	0	0	30,79
Graf06	52,81	0	0	0	0,93	2,8

Tabela 5 zawiera porównanie jakości rozwiązań dla różnych topologii NoC. Kryteria porównania to średnia ważona liczba skoków (1) oraz liczba potrzebnych do budowy danej topologii łącz jednokierunkowych. Systemem referencyjnych jest proponowany w niniejszej pracy NoC typu WWK. System typu WKK stanowi na tym polu rozwiązanie tylko nieznacznie gorsze. Trasy komunikatów są tu średnio o ok. 3% dłuższe (wartość ujemna dla danej pozycji oznacza poprawę względem systemu wzorcowego), zaś topologia wymaga średnio 2% więcej łącz międzyruterowych. Obie grupy systemów typu mesh wypadają znacznie gorzej – pogorszenie jest średnio kilkudziesięcioprocentowe. Dla długości tras sięga nawet 53%, a użytych łącz – 66%. Ostatnie dwie grupy rozwiązań – „bndwdth" – wypadły nieco lepiej, jeśli chodzi o długość wyznaczonych tras (średni zysk w długości tras to 3,4% i 2,5%). Taki wynik nie dziwi, gdyż są to metody mające na celu minimalizację ruchu w mikrosieci. Natomiast zapotrzebowanie na zasoby układu w obu ostatnich podejściach było większe od systemu WWK i wyniosło średnio odpowiednio 21,8% oraz 14,9%. W drugim przypadku spadek jakości rozwiązania jest mniejszy, gdyż do usunięcia kolizji

metoda korzysta z przeszeregowania zamiast z dedykowanych, dodatkowych połączeń między ruterami.

Tabela 5

Pogorszenie parametru hop_{AVG} oraz liczby użytych łącz jednokierunkowych w porównaniu z zaprezentowanym podejściem (NoC WWK)

Aplikacja	WWK hop_{AVG} (łącza)	Pogorszenie arch. NoC: hop_{AVG}/łącza [%]				
		WKK	mesh NxN	mesh MxN	bndwdth+ łącza	bndwdth+ przeszereg.
Graf01	2,75 (14)	–2,23/–7,69	10,42/41,67	30,38/46,15	–11,79/17,65	–9,56/12,5
Graf02	2,44 (17)	5,43/10,53	44,8/64,58	37,11/50	5,43/26,09	5,43/22,73
Graf03	2,69 (23)	7,88/–4,55	39,55/52,08	39,41/42,5	–8,91/17,86	–5,08/11,54
Graf04	2,44 (17)	5,79/–6,25	46,37/64,58	37,76/50	–6,55/32	–0,41/22,73
Graf05	2,11 (22)	1,4/8,33	53,42/54,17	45,62/35,29	2,31/21,43	–1,93/0
Graf06	2,73 (16)	0,36/11,11	18,02/66,67	21,33/38,46	–0,74/15,79	–3,41/20

5. Wnioski

W pracy przedstawiono nowy algorytm kosyntezy systemów heterogenicznych o architekturze NoC, ukierunkowany na minimalizację kosztu przy zadanych ograniczeniach czasowych. Zaproponowany jako alternatywa dla architektur magistralowych nowy sposób budowania topologii połączeń NoC również daje dużo lepsze rezultaty – przewaga szybkości działania nad systemami ze wspólną szyną komunikacyjną to średnio 35%. Porównanie wygenerowanych topologii NoC z innymi – regularnymi i nieregularnymi – również potwierdza wysoką jakość uzyskanych rozwiązań. Jedynie na polu długości tras przydzielonych komunikatom zaprezentowane podejście ustępuje nieco metodom typu „bandwidth--based" średnio o ok. 3%.

Dalszy kierunek prac przewiduje modyfikację metodologii polegającej na wbudowaniu generacji topologii NoC w algorytm kosyntezy, w ten sposób podczas rafinacji uwzględniona będzie architektura wraz z topologią połączeń. Dzięki temu uniknie się potrzeby wielokrotnego powtarzania kosyntezy, chociaż odbędzie się to kosztem większej złożoności etapów związanych z generowaniem kolejnych rozwiązań. Do metodologii można również dodać kryterium minimalizacji poboru mocy. W toku są również prace uzupełniające sieć działań algorytmu z rysunku 2 o etap rozplanowania elementów systemu w układzie scalonym (ang. *floorplanning*). Celem jest minimalizacja długości połączeń w uzyskanej topologii dla układów o elementach homo- i heterogenicznych.

Literatura

[1] Gupta R.J., De Micheli G., *Hardware-Software Co-synthesis for Digital Systems*, IEEE Design & Test, Vol. 10, No. 3, September 1993, 29-41.

[2] Henkel J., Ernst R., *A Hardware/Software Partitioner using a dynamically determined Granularity*, Proc. of Design Automation Conference, 1997, 691-696.

[3] Kalavade A., Lee E.A., *The Extended Partitioning Problem: Hardware/ Software Mapping and Implementation-Bin Selection*, Proc. of 6th International Workshop on Rapid Systems Prototyping, 1995.
[4] Dave B.P., Lakshminarayana G., Jha N.K., *COSYN: Hardware-Software Co-Synthesis of Embedded Systems*, Proc. of Design Automation Conference, 1997, 703-708.
[5] Dick R.P., Jha N.K., *MOGAC: A multiobjective Genetic Algorithm for the Co-Synthesis of Hardware-Software Embedded Systems*, Proc. of International Conference on Computer Aided Design, 1997, 522-529.
[6] Eles P., Peng Z., Kuchcinski K., Doboli A., *System Level Hardware/ Software Partitioning Based on Simulated Annealing and Tabu Search*, Design Automation for Embedded Systems, Vol. 2, No. 1, 1997, 5-32.
[7] Eles P., Kuchcinski K., Peng Z., Doboli A., Pop P., *Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems*, Proc. of Conference on Design Automation and Test in Europe (DATE'98), 1998, 132-138.
[8] Lee H.G., Chang N., Ogras U.Y., Marculescu R., *On-Chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches*, ACM Transactions on Design Automation of Electronic Systems, 12(3), Article 23, 2007.
[9] Bjerregaard T., Mahadevan S., *A survey of research and practices of network-on-chip*, ACM Computing Surveys, 38(1), 2006, 71-121.
[10] Marculescu R., Ogras U.Y., Peh L.S., Jerger N.E., Hoskote Y., *Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives*, IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 28(1), 2009, 3-19.
[11] Hu J., Marculescu R., *Communication and task scheduling of application--specific Networks-on-Chip*, IEEE Proceedings of Computers and Digital Techniques, 2005, 643-651.
[12] Lei T., Kumar S., *A two-step genetic algorithm for mapping task graphs to a network on chip architecture*, Proceedings of Euromicro Symposium on Digital Systems Design, 2003, 180-187.
[13] Chen Y.-J., Yang C.-L., Chang Y.-S., *An architectural co-synthesis algorithm for energy-aware Network-on-Chip design*, Journal of Systems Architecture, 55, 2009, 299-309.
[14] Shin D., Kim J., *Communication Power Optimization for Network-on-Chip Architectures*, Journal of Low Power Electronics, 2(2), 2006, 1-12.
[15] Jang W., Pan D.Z., *A3MAP: Architecture-Aware Analytic Mapping for Networks--on-Chip*, Asian and South Pacific Design Automation Conference (ASPDAC), 2010, 523-528.
[16] Murali S., et al., *Designing application-specific networks on chips with floorplan information*, International Conference on Computer Aided Design, 2006, 355-362.
[17] Ho W.H., Pinkston T.M., *A design methodology for efficient application--specific on-chip interconnects*, IEEE Transactions on Parallel and Distributed Systems, 17(2), 2006, 174-190.

[18] Neeb Ch., Wehn N., *Designing efficient irregular networks for heterogeneous systems-on-chip*, Journal of Systems Architecture: the EUROMICRO Journal, 54(3-4), 2008, 384-396.
[19] Ogras U.Y., Marculescu R., *Application-specific network-on-chip architecture customization via long-range link insertion*, International Conference on Computer Aided Design, 2005, 246-253.
[20] Wang D., Matsutani H., Amano H., Koibuchi M., *A link removal methodology for Networks-on-Chip on reconfigurable systems*, International Conference on Field Programmable Logic and Application, 2008, 269-274.
[21] Chou C.L., Marculescu R., *Contention-aware application mapping for Network-on-Chip communication architectures*, International Conference on Computer Design, 2008, 164-169.
[22] Leary G., Chatha K.S., *Automated technique for design of NoC with minimal communication latency*, CODES+ISSS, 2009, 471-480.
[23] Deniziak S, *Cost-efficient synthesis of multiprocessor heterogeneous systems,* Control and Cybernetics, 33(2), 2004, 341-355.
[24] Deniziak S., Tomaszewski R., *Contention-avoiding custom topology generation for network-on-chip*, International Symposium on Design and Diagnostics of Electronic Circuits & Systems, 2009, 234-237.
[25] Wolf W., *High-Performance Embededed Computing*, Morgan Kaufman, 2006.
[26] Suzuki K., Sangiovanni-Vincentelli A., *Efficient Software Performance Estimation Methods for Hardware/Software Codesign*, Proc. of the ACM/IEEE Design Automation Coference, 1996, 605-610.
[27] Sethuraman B., Bhattacharya P., Khan J., Vemuri R., *LiPaR: A light-weight parallel router for FPGA-based Networks-on-Chip*, In 15th Great Lakes Symposium on VLSI, 2005, 452-457.
[28] Yen T.Y., Wolf W., *Sensitivity-Driven Co-Synthesis of Distributed Embedded Systems*, Proc. of International Symposium on System Synthesis, 1995, 4-9.
[29] Dick R.P., Rhodes D.L., Wolf W., *TGFF: Task graphs for free*, International Workshop on Hardware/Software Codesign, 1998, 97-101.