II Konferencja Naukowa KNWS'05
"Informatyka- sztuka czy rzemiosło"
15-18 czerwca 2005, Złotniki Lubańskie

METODA TRANSFORMACJI SPECYFIKACJI BEHAWIORALNEJ UKŁADÓW CYFROWYCH NA SIECI PETRIEGO W SYNTEZIE SYSTEMOWEJ

Zbigniew Skowroński

Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski
65-246 Zielona Góra, ul. Podgórna 50

e-mail: Z.Skowronski@iie.uz.zgora.pl

STRESZCZENIE

W pracy przedstawiono metodę transformacji specyfikacji behawioralnych układów cyfrowych w językach HDL w równoważną im semantycznie sieć Petriego dla potrzeb syntezy systemowej. Opracowana metoda zachowuje wszystkie implikacje, będące konsekwencją specyficznego cyklu symulacji tych języków. Dzięki czemu została zachowana odpowiedniość między wynikami symulacji specyfikacji w językach HDL i modelu formalnego (interpretowane sieci Petriego) bez względu na liczbę procesów i sygnałów występujących w specyfikacji. Dodatkowo, metoda została tak zrealizowana, aby istniała możliwość jej wykorzystania w transformacji specyfikacji w językach programowania (Pascal, C) na równoważne im pod względem semantycznym sieci Petriego.

1. WPROWADZENIE

Pierwszym krokiem w syntezie systemowej jest przekształcenie wejściowej specyfikacji behawioralnej w wybraną reprezentację pośrednią, stanowiącą formalny model tej specyfikacji. Tradycyjnie modelem formalnym jest graf przepływu danych i sterowania (ang. Control and Data Flow Graph, CDFG) [3], a wejściowa specyfikacja behawioralna danego układu cyfrowego najczęściej zadana jest w językach HDL (VHDL [5], Verilog [6]).

Specyfikacja behawioralna niejednokrotnie tworzona jest przez projektantów bezpośrednio na podstawie opisu słownego. Pomimo tego, że języki HDL posiadają mechanizmy tworzenia opisu z równoległością, to specyfikacja behwioralna może posiadać pewną wadę wynikającą z faktu, że umysł ludzki pracuje w sposób sekwencyjny, czego naturalną konsekwencją jest trudność w opracowywaniu równoważnego opisu wielu współbieżnie przebiegających zdarzeń. W efekcie, prowadzi to otrzymania opisu, który zdecydowanie odbiega od kryterium minimalnego czasu realizacji i charakteryzuje się słabym stopniem zrównoleglenia procesów w układzie. Dodatkowo sekwencyjny sposób opisu zdarzeń współbieżnych zrywa naturalną więź przyczynowo-skutkową.

Z tego względu, bardzo ważnym zagadnieniem, na które należy zwrócić uwagę w trakcie realizacji procesu transformacji jest sposób przekształcenia wejściowej specyfikacji

funkcjonalnej na model formalny. Metoda transformacji powinna zapewniać duży stopień zrównoleglenia procesów w układzie i zachowywać semantykę specyfikacji behawioralnej. Reprezentacja opisów wieloprocesowych pracujących równolegle natrafia na poważne trudności w przypadku stosowania modelu CDFG, który nie jest wystarczająco dostosowany do opisywania współbieżności. W pracy zastosowano interpretowane sieci Petriego [1,10, 12, 14], dzięki którym możliwe jest wprowadzenie w sposób jawny współbieżności operacji i uwzględnienie konsekwencji cyklu symulacji [9, 12, 13), wynikającego z dyskretnego taktowania zdarzeń zarówno w przypadku języka VHDL, jak i Verilog.

Podstawowym zadaniem jest więc transformacja wieloprocesowych specyfikacji funkcjonalnych w językach HDL w równoważną im semantycznie sieć Petriego, zachowującą wszystkie implikacje wynikające z pojęcia cyklu symulacji tych języków [12, 13]. W ten sposób zostaje zachowana odpowiedniość wyników symulacji specyfikacji wejściowej (VHDL, Verilog) i modelu formalnego (interpretowana sieć Petriego), bez względu na liczbę procesów i sygnałów występujących w specyfikacji.

2. ZAŁOŻENIA PROPONOWANEJ METODY TRANSFORMACJI

Wejściowa specyfikacja behawioralna układu cyfrowego zadana w językach HDL zawiera złożone instrukcje współbieżne (np. `process` - w języku VHDL, `always` - w języku Verilog), które wewnątrz realizowane są w sposób sekwencyjny. Z tego względu, proponowana metoda transformacji w pierwszej fazie procesu przekształcania musi wyraźnie definiować sposób przetworzenia opisu sekwencyjnego, występującego wewnątrz tych instrukcji, w odpowiadające mu elementy modelu formalnego. Jednocześnie wprowadzana jest równoległość tych operacji, które są wzajemnie niezależne. W następnej fazie transformacji zostaje zrealizowany mechanizm synchronizacji procesów za pomocą operacji oczekiwania oraz rozbudowa modelu formalnego o moduły odpowiedzialne za uaktualnienie wartości dla sygnałów [9, 12].

Podstawowymi założeniami, jakie przyjęto w metodzie, są [12, 13]:

- zamiana podczas translacji specyficznych instrukcji przypisania, charakterystycznych dla specyfikacji w językach HDL (VHDL lub Verilog), na współbieżnie procesy, zawierające instrukcje sekwencyjne;
- przypisanie każdej operacji wewnątrz procesu do miejsca sieci w taki sposób, że znakowanie rozpatrywanego miejsca odpowiada wykonywaniu danej operacji. Jeśli wykonanie operacji jest warunkowe, wówczas warunek ten przypisany jest do tranzycji wejściowej miejsca reprezentującego daną operację.

W ogólnym przypadku translator, program komputerowy realizujący zaproponowaną metodę transormacji, powinien zostać wzbogacony o kilka dodatkowych elementów [12, 13]:

analizator leksykalny, odczytujący kod źródłowy i przetwarzający go na zbiór elementów składniowych wykorzystywanych w kolejnym etapie transformacji; *analizator składni*, którego zadaniem jest sprawdzenie, czy zbiór elementów składniowych przygotowanych wcześniej przez analizator leksykalny, zawiera elementy zgodne z regułami składni oraz *generator specyfikacji tekstowej modelu formalnego*, który na podstawie otrzymanej formy pośredniej tworzy tekstowy opis sieci Petriego w formacie PNSF (ang. Petri Net Specification Format [8].

Podstawą formatu PNSF jest specyfikacja regułowa, oparta na logice sekwentów Gentzena, z elementami logiki temporalnej [2].

W proponowanym podejściu do analizy leksykalnej i składni stosowane są ogólnie dostępne kompilatory i symulatory języków HDL. Kod źródłowy przeanalizowany przez analizator leksykalny i składni zostaje przekształcony na opis w danym języku HDL, zawierający współbieżnie pracujące procesy (zbudowane z sekwencyjnie wykonywanych operacji), które w kolejnej fazie transformacji zostają poddane analizie przepływu danych oraz sterowania (moduły zaprojektowanych translatorów). W konsekwencji otrzymuje się zbiór operacji oraz zależności, który zostanie wykorzystany do przetworzenia wejściowej specyfikacji funkcjonalnej w odpowiadające im elementy sieci Petriego. Tak otrzymany zbiór elementów składowych sieci Petriego, wzbogacony o sieci realizujące synchronizację operacji oczekiwania (sieć *synchronizująca*) i uaktualniania wartości dla sygnałów (sieć *aktualizująca*) [12], zostaje przetworzony na format tekstowy sieci Petriego.

Z tego względu zasadniczą i zarazem podstawową fazą transformacji jest przekształcenie operacji sekwencyjnych zapisanych w procesie (procesach) w sieć Petriego. Jednocześnie z przekształceniem ich w sieć przeprowadzana jest analiza możliwości równoległego wykonywania poszczególnych operacji, w celu przyspieszenia realizacji całej specyfikacji.

3. OPERACJE SEKWENCYJNE A RÓWNOLEGŁE

Do rozpatrywanego zagadnienia podchodzi się w dwojaki sposób, wynikający z definicji maksymalnej równoległości operacji. Pierwszą definicję formułuje się następująco:

> *System z maksymalną równoległością operacji, to taki system, w którym każde dwie operacje wzajemnie niezależne realizowane są współbieżnie.* (*Def. 1*)

Jako operacje wzajemnie niezależne rozumiane są takie pary lub zbiory operacji, które nie są powiązane ze sobą żadną relacją przyczynowo-skutkową. Stosując takie podejście do metody przetworzenia opisu sekwencyjnego w równoległy, konieczne jest wyszukanie relacji niezależności pomiędzy wszystkimi operacjami, co w realizacji praktycznej wymaga porównania typu *każda z każdą*. Rozwiązanie to komplikuje obliczenia. Można ten problem uprościć przez zastosowanie podziału analizowanego opisu na mniejsze bloki lecz ten z kolei wprowadza nowe operacje i obliczenia.

Alternatywą do tego podejścia jest zaproponowana w pracy metoda wykorzystująca odwrócenie podanej wcześniej definicji:

System z maksymalną równoległością operacji, to taki system, w którym sekwencyjnie wykonywane są tylko operacje związane zależnością przyczynowo-skutkową. (*Def. 2*)

W takim podejściu złożoność obliczeniowa maleje w znaczący sposób. Wynika to z faktu, że w typowych przypadkach zbiór operacji, od których dana operacja jest zależna, jest mniejszy od zbioru operacji, z którymi może ona być realizowana współbieżnie. Jeśli dodatkowo zostaną określone warunki w jakich dwie operacje są zależne, to w ten sposób zostanie ograniczony zbiór operacji jakie należy przeanalizować. Podstawą definicji takich warunków jest przepływ danych. Na jego podstawie zostanie określony warunek konieczny i wystarczający wzajemnej niezależności operacji:

Dwie operacje są wzajemnie niezależne wtedy, gdy żadna z nich nie przygotowuje danych dla drugiej oraz nie określają one tej samej zmiennej wyjściowej. (*Def. 3*)

Formalnie warunek ten zapisuje się następująco:

Niech N_1 i N_2 oznaczają dwie operacje, a In_i i Out_i dla i=1,2 odpowiednio zbiory zmiennych wejściowych i wyjściowych i-tej operacji. Operacje N_1 i N_2 nazywamy niezależnymi, gdy $In_1 \cap Out_2 = \varnothing \wedge In_2 \cap Out_1 = \varnothing \wedge Out_2 \cap Out_1 = \varnothing$. (*Def. 4*)

Warto również zauważyć, że nie jest konieczne spełnienie warunku $In_1 \cap In_2 = \varnothing$.

Spośród operacji, które nie spełniają definicji 4 w stosunku do operacji N_k wyróżnia się dwa zbiory operacji: potencjalnie poprzedzających operację N_k (potencjalne poprzedniki - PPr_k) i potencjalnie występujących po niej (potencjalne następniki - PSu_k).

Formalnie definiuje się te zbiory następująco:

$$PPr_k = \{N_i \mid Out_i \cap In_k \neq \varnothing\} \quad (Def.\ 5)$$

$$PSu_k = \{N_j \mid Out_k \cap In_j \neq \varnothing\} \quad (Def.\ 6)$$

Z kolei zbiory te ogranicza się do zbioru poprzedników i następników danej operacji, oznaczonych odpowiednio Pr_k i Su_k, według zasady:

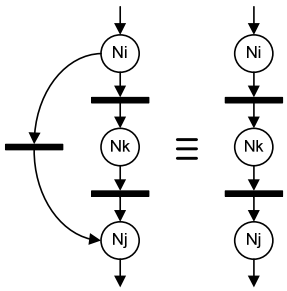
Operacją poprzedzającą operację N_k może być operacja N_i, jeśli i<k i $N_i \in PPr_k$. Analogicznie definiuje się zbiór następników:

$$Pr_k = \{N_i \mid i<k \wedge N_i \in PPr_k\} \quad (Def.\ 7)$$

$$Su_k = \{N_j \mid j>k \wedge N_j \in PSu_j\} \quad (Def.\ 8)$$

Oprócz zbiorów poprzedników i następników, dla każdej operacji należy określić dodatkowo zbiory operacji bezpośrednio i globalnie poprzedzających. Konieczność ich wprowadzenia wynika z dążenia do zminimalizowania liczby ścieżek przepływu sterowania w tworzonej sieci Petriego. Jeżeli na przykład, jednemu rodzajowi wierzchołków (np. miejscom) przypiszemy operacje, to ścieżki łączące takie wierzchołki odpowiadają relacji poprzedzania

i następstwa. Oznacza to, że jeśli dwa wierzchołki połączone są ścieżką, to jeden z nich poprzedza drugi. Nie zawsze jednak ścieżka taka jest wymagana. Jeśli w sytuacji takiej, że z jednego wierzchołka do drugiego prowadzą co najmniej dwie ścieżki, to jedną z nich można usunąć, ponieważ relacja następstwa realizowana jest pośrednio przez drugą ścieżkę. Przykład takiej sytuacji przedstawia rysunek 1.



Rys. 1. Usuwanie zbędnych ścieżek między operacjami

Lewa ścieżka zostaje usunięta, gdyż przejście z operacji N_i do N_k wystąpi poprzez ścieżkę prawą. Usunięcie zbędnych ścieżek powoduje zmniejszenie liczby tranzycji w tworzonej sieci Petriego.

Zasadę usuwania ścieżek określa się następująco:

> *Ścieżkę łączącą operacje N_i i N_j można usunąć, gdy operacja N_i należy do zbioru operacji bezpośrednio poprzedzających N_j i jednocześnie należy do zbioru operacji poprzedzających ją globalnie:*
>
> $$N_i \in Pr\text{-}L_j \wedge N_i \in (Pr\text{-}G_j - Pr\text{-}L_j) \qquad (Def.\ 9)$$

gdzie: $Pr\text{-}L_j$ jest zbiorem bezpośrednich poprzedników operacji N_j a $Pr\text{-}G_j$ jest zbiorem globalnych poprzedników operacji N_j.

Zbiór operacji bezpośrednio poprzedzających (poprzedniki lokalne) definiuje się następująco:

$$Pr\text{-}L_j = \{N_i \mid \forall x \in In_j : x \in Out_i \wedge j\text{-}i=min\} \qquad (Def.\ 10)$$

Drugi warunek $j\text{-}i=min$ oznacza, że N_i jest ostatnią operacją, dla której x był parametrem wejściowym.

Zbiór operacji globalnie poprzedzających (poprzedników globalnych) operacji N_i definiuje się rekurencyjnie jako zbiór wszystkich operacji poprzedzających operacje należące do zbioru $Pr\text{-}L_j$. Formalnie określa się to następująco:

$$Pr\text{-}G_j = \{N_n \mid \forall N_i \in Pr\text{-}L_j : N_n \in Pr\text{-}G_i\} \qquad (Def.\ 11)$$

Proces przekształcania rozpoczyna się od ponumerowania wszystkich operacji. Kolejnym krokiem jest utworzenie zbiorów zmiennych wejściowych i wyjściowych każdej operacji. Jako zmienne wejściowe należy również uwzględnić zmienne wchodzące w skład warunku

rozpoczęcia wykonania operacji.

Na podstawie zbiorów zmiennych wejściowych i wyjściowych dla każdej operacji tworzony jest zbiór operacji bezpośrednio ją poprzedzających. Powstaje on w następujący sposób:

Dla każdej zmiennej wejściowej danej operacji wyszukiwana jest ostatnia operacja, w której dana zmienna była zmienną wyjściową. Jeżeli była to operacja bezwarunkowa, to jej numer wpisuje się do zbioru wprost, jeżeli zaś była ona warunkowa, to należy wpisać ją razem z poprzednią operacją, spełniającą podany warunek zależności zmiennej, grupując je razem. Takie zagłębianie należy prowadzić aż do momentu, kiedy operacja poprzedzająca będzie bezwarunkowa lub kiedy warunek operacji poprzedzającej będzie taki sam jak danej operacji.

Po utworzeniu zbiorów operacji bezpośrednio poprzedzających, tworzy się zbiory operacji globalnie poprzedzających, a na podstawie tych zbiorów usuwa się zbędne relacje poprzedzania, zgodnie z zasadą podaną wcześniej. Następnym krokiem jest już tworzenie sieci Petriego.

Na podstawie wyżej przeprowadzonych rozważań można zdefiniować metodę tramsformacji specyfikacji funkcjonalnej układów cyfrowych zadanej w językach HDL na interpretowaną sieć Petriego.

4. METODA PRZEKSZTAŁCANIA OPERACJI SEKWENCYJNYCH W SIEĆ PETRIEGO

Schemat blokowy metody transformacji specyfikacji behawioralnej układów cyfrowych na sieci Petriego, upraszczający się do przekształcenia operacji sekwencyjnych w interpretowaną sieć Petriego, w przypadku nie uwzględniania konsekwencji cyklu symulacji, przedstawia rysunek 2 [12] oraz dalsza część pracy.

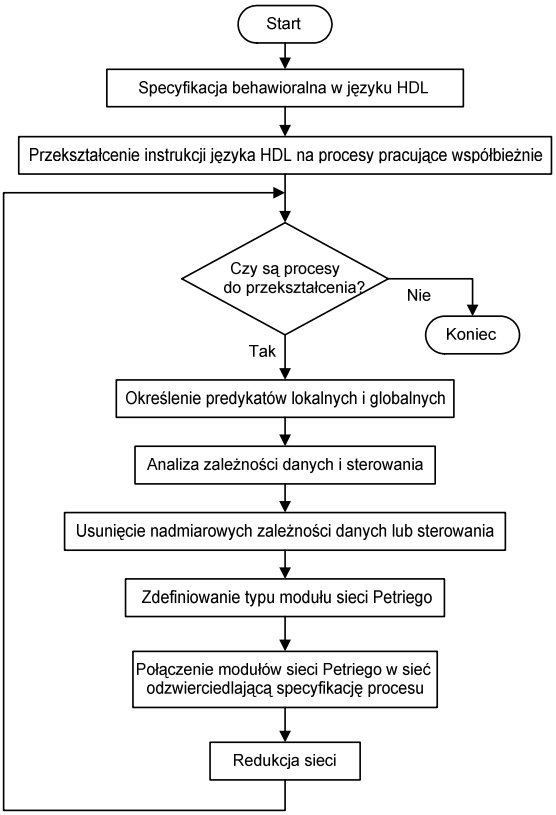
Niech N_i oznacza *i*-tą operację wejściowej specyfikacji zapisanej w języku HDL (VHDL, Verilog), przy czym uporządkowanie operacji jest dowolne. Każdą operację N_i opisuje kilka parametrów, które określa się poprzez analizę specyfikacji. Formalnie operację N_i definiuje się jako uporządkowaną szóstkę:

$$N_i = (In_i, Out_i, Op_i, Pr_i, Su_i, Con_i) \qquad (Def.\ 12)$$

gdzie: In_i jest zbiorem zmiennych wejściowych N_i; Out_i jest zbiorem zmiennych wyjściowych N_i; Op_i jest zbiorem operandów N_i; Pr_i jest zbiorem poprzedników N_i; Su_i jest zbiorem następników N_i; Con_i jest predykatem operacji N_i.

Predykaty poszczególnych operacji definiuje się na podstawie analizy zależności (ang. Dependency Analysis).

Dowolną operację N_j nazywa się zależną od innej operacji N_k jeśli N_k przygotowuje dane dla N_j (zależność danych) lub realizacja operacji N_k powinna się zakończyć przed rozpoczęciem realizacji operacji N_j (zależność kontrolna). (*Def. 13*)



Rys. 2. Schemat blokowy przekształcenia specyfikacji behawioralnej układów cyfrowych zadanej w językach HDL na sieć Petriego (bez uwzględnienia konsekwencji cyklu symulacji)

Typowym przykładem zależności kontrolnej jest sytuacja, w której N_k jest operacją warunkową, a realizacja N_j zależy od wyniku N_k, choć nie ma między nimi zależności danych (*if* N_k *then* N_j).

Następnikami operacji N_i *są operacje, dla których* N_i *jest poprzednikiem.* (*Def. 14*)

Predykaty są logicznymi warunkami wykonania operacji. W pracy wykorzystuje się je do określania zależności kontrolnych, które znajdują odzwierciedlenie w tworzonej sieci Petriego.

Predykat operacji jest logicznym iloczynem warunków, jakie powinny być spełnione by operacja mogła zostać zrealizowana. (*Def. 15*)

Zwykle odnosi się to do wyników działania operacji warunkowych, które mogą być zagnieżdżone. Jeśli operacja nie jest operacją warunkową uważa się, że jej predykat ma stałą wartość *TRUE* i nie zaznacza się go w sieci.

W przedstawianej metodzie stosowane są dwa rodzaje predykatów: predykaty globalne i predykaty lokalne. Podana wcześniej definicja 15 odnosi się do predykatów globalnych

Con-G. Predykaty lokalne *Con-L* definiuje się w odniesieniu do relacji zależności operacji.

Dla dwóch operacji N_k i N_m takich, że $N_k \in Pr_m$ i Con-$G_m \in$ Con-G_k predykat lokalny Con-L_m operacji N_m definiuje się jako Con-G_k − Con-G_m. (*Def. 16*)

Przykładowo, niech operacja N_g będzie poprzednikiem operacji N_h, a ich globalne poprzedniki to odpowiednio *Con-G_g=C1* i *Con-G_h=C1*C2*. W takim przypadku predykatem lokalnym N_h będzie *Con-L_h=C2*.

Składnia sieci Petriego wymaga, aby miejsca były łączone z tranzycjami i na odwrót. Z tego względu sekwencja operacji reprezentowanych przez miejsca sieci Petriego musi być rozdzielana tranzycjami. Dla ułatwienia konstruowania sieci tworzona jest ona z bloków (modułów). *Moduł i* odpowiadający *operacji i* jest podsiecią definiowaną jako:

$$BM_i = (P_i, T_i, F_i) \qquad (Def.\ 4\text{-}17)$$

gdzie: $P_i = \{p_{i1}, p_{i2}, ..., p_{im}\}$ jest skończonym niepustym zbiorem miejsc, z których jedno reprezentuje operację N_i; $T_i = \{t_{i1}, t_{i2}, ..., t_{in}\}$ jest skończonym niepustym zbiorem tranzycji; $P_i \cap T_i = \varnothing$ i $P_i \cup T_i \neq \varnothing$; $F_i \subseteq (P_i \times T_i) \cup (T_i \times P_i) \cup (P_i \times \#)$ jest skończonym niepustym zbiorem łuków, w którym łuki $(P_i \times \#)$ nazywane są łukami końcowymi i służą do łączenia modułu z innymi modułami.

Ponadto jeśli operacja N_i ma niepusty predykat (*Con-$L_i \neq TRUE$*), wówczas predykat ten jest przypisywany do tranzycji wejściowej miejsca reprezentującego operację.

Moduły mają różną strukturę, co wynika z różnego rodzaju zależności między operacjami odzwierciedlonej różnymi predykatami. W ogólnym przypadku moduły odpowiadające operacjom dzieli na dziewięć typów, oznaczanych dwoma literami. Pierwsza litera określa zależności *wejściowe*, druga zaś - *wyjściowe*. Stosowane są następujące kody:

- dla *wejścia* modułu: *A* - z jednego poprzednika, *B* - z kilku poprzedników o niesprzecznych predykatach, *C* - z kilku poprzedników o sprzecznych predykatach;
- dla *wyjścia* modułu: *A* - do jednego następnika, *B* - do kilku następników o niesprzecznych predykatach, *C* - do kilku następników o sprzecznych predykatach.

Przykładowo, moduł typu *AB* ma tylko jedno wejście do tranzycji zezwalającej (wejściowej) i kilka następników o niesprzecznych predykatach. Predykaty są sprzeczne jeśli wzajemnie się wykluczają, jak na przykład *X* i */X*.

Pełna sieć Petriego opisująca wnętrze procesu (tzw. sieć *procesowa*) tworzona jest poprzez łączenie ze sobą modułów. Zasadę łączenia modułów można sformułować następująco:

Dla każdej pary operacji N_k i N_m takich, że $N_k \in Pr_m$ jeden z łuków końcowych modułu BM_k jest łączony do (jednej z) tranzycji wejściowej (wejściowych) modułu BM_m. (*Def. 18*)

Wykorzystując przedstawione zasady przekształcania operacji sekwencyjnych w sieć Petriego, zadanie transformacji specyfikacji behawioralnej układu cyfrowego zadanej

w języku HDL na interpretowaną sieć Petriego można podzielić na siedem następujących po sobie kroków:

1. Określenie predykatów lokalnych i globalnych;
2. Uzyskanie informacji o następnikach i poprzednikach danej operacji wewnątrz procesu za pomocą analizy zależności danych i sterowania;
3. Usunięcie nadmiarowych zależności danych i sterowania;
4. Zdefiniowanie typu modułu sieci Petriego, odpowiadającego danej operacji;
5. Połączenie modułów sieci Petriego w celu otrzymania sieci odpowiadającej specyfikacji procesu (sieć *procesowa*);
6. Ewentualna redukcja uzyskanej sieci *procesowej*;
7. Rozbudowa sieci *procesowej*, otrzymanej w krokach 1-6, o sieć *synchronizującą* i *aktualizującą* (szeroko opisane w pracy [12]).

Sekwencja kroków 1-7 jest przeprowadzona dla każdego procesu, jaki został określony w wejściowej specyfikacji behawioralnej.

Tabela 1. Przykład transformacji wejściowej specyfikacji behawioralnej w celu uzyskania predykatów

Wejściowa specyfikacja w języku HDL	Specyfikacja w pseudokodzie

`if (Var3 != 0)`
`  Var4 = In2 + 4;`
`  else Var4 = In3 - In5;`
	`Con1 := Var3 <> 0`
	`if Con1`
	`  then Var4 := In2 + 4;`
	`if NOT Con1`
	`  then Var4 := In3 - In5;`

W celu określenia predykatów należy dodatkowo przekształcić wejściową specyfikację w ten sposób, że operacje warunkowe zamienia się na operacje ze zmienną logiczną, będącą wynikiem realizacji danej operacji. Sposób tej transformacji obrazuje tabela 1.

Po dokonaniu przekształcenia wejściowej specyfikacji w specyfikację w pseudokodzie dokonuje się realizacji następnych kroków zdefiniowanych w metodzie.

5. PODSUMOWANIE

Praca została ukierunkowana pod kątem wykorzystania w zintegrowanym projektowaniu systemów sprzętowo-programowych (ang. Hardware/Software Co-Design). Tego rodzaju projektowanie [4, 7] jest stosunkowo nową dziedziną syntezy systemowej, rozwijaną praktycznie od kilku lat [1, 11, 15]. Zasadniczą ideą takiego podejścia jest całościowe (systemowe) spojrzenie na projektowane urządzenie, bez przedwczesnego podziału jego specyfikacji na część programową i sprzętową, jak to ma miejsce w metodach tradycyjnych. Konsekwencją tego faktu jest konieczność opracowania nowych algorytmów projektowania.

Badania polegające na sformalizowaniu specyfikacji oraz zastosowaniu metod formalnych w procesie projektowania pozwolą na jego skrócenie oraz potencjalnie mogą przynieść lepsze wyniki, z punktu widzenia takich kryteriów jak: szybkość działania, koszt, zużycie energii, waga urządzenia, czy innych.

Praca naukowa finansowana ze środków Komitetu Badań Naukowych w latach 2003-2006 jako projekt badawczy nr 4 T11C 006 24.

LITERATURA

[1] M. Adamski, Z. Skowroński: *Interpretowane sieci Petriego - model formalny w zintegrowanym projektowaniu mikroprocesorowych systemów sprzętowo-programowych*, Pomiary Automatyka Kontrola, 2003, nr 2-3, s. 17-20

[2] M. Adamski: *Projektowanie układów cyfrowych systematyczną metodą strukturalną*, Monografie, Wydawnictwo WSI w Zielonej Górze, Zielona Góra, 1990

[3] D. Gajski, L. Ramachandran: *Introduction to High-Level Synthesis*, IEEE Design & Test of Computers, vol. 11, No 4, Winter 1994, pp. 44-54

[4] J. van den Hurk, J. Jess: *System Level Hardware/Software Co-Design, An Industrial Approach*, Kluwer Academic Publishers, 1997, ISBN 0-7923-8084-3

[5] *IEEE Standard VHDL Language Reference Manual*, IEEE, New York, 1988

[6] *IEEE Standard HDL Based on the Verilog® HDL*, IEEE, New York, 1996

[7] Ed. by A. A. Jerraya, J. Mermet: *System-Level Synthesis*, Kluwer Academic Publishers, 1999, ISBN 0-7923-5749-3

[8] T. Kozłowski, E. L. Dagless, J. M. Saul, M. Adamski, J. Szajna: *Parallel controller synthesis using Petri nets*, IEEE Proceedings, Computers and Digital Techniques, vol. 142, no 4, pp. 263-271, July 1995

[9] J. Mirkowski, Z. Skowroński: *Translation of C and VHDL specifications into interpreted Petri nets for Hardware/Software codesign*, Mixed design of integrated circuits and systems, Boston, Kluwer Academic Publishers, 1998, pp. 163-168, ISBN 0-7923-8116-5

[10] T. Murata: *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, vol. 77, No 4, April 1989, pp. 548-580

[11] Z. Skowroński, J. Mirkowski: *Zintegrowane projektowanie systemów mikroprocesorowych ze specjalizowanymi układami cyfrowymi*, II Konferencja Informatyka na wyższych uczelniach dla gospodarki narodowej, Gdańsk, 22-24 listopada 1996 r, tom I, s. 123-126

[12] Z. Skowroński: *Translacja specyfikacji funkcjonalnej układów cyfrowych na sieć Petriego dla potrzeb syntezy systemowej*, Rozprawa doktorska, Politechnika Szczecińska, Wydział Informatyki, 2000, s. 156

[13] Z. Skowroński: *Problem translacji specyfikacji funkcjonalnej układów cyfrowych w języku VHDL na interpretowaną sieć Petriego w zintegrowanej syntezie systemów sprzętowo-*

programowych,: Reprogramowalne Układy Cyfrowe - RUC 2001, Materiały IV Krajowej Konferencji Naukowej, Politechnika Szczecińska, Szczecin, Polska, s. 103-113

[14] Z. Skowroński: *Translacja specyfikacji funkcjonalnej układów cyfrowych na sieć Petriego dla potrzeb syntezy wysokiego poziomu*, Materiały I Ogólnopolskich Warsztatów Doktoranckich OWD'99, Istebna-Zaolzie, 17-21 października 1999, s. 185-196, ISBN 83-907217-5-9

[15] A. Stasiak, Z. Skowroński: *The intermediate model for hardware/software microsystems based on Petri nets*, Discrete-Event System Design - DESDes' 04, Proceedings of the International Workshop. Dychów, Polska, 2004, Zielona Góra, University of Zielona Góra Press, pp. 51-56, ISBN 83-89712-15