STUDIA INFORMATICA 2012
Volume 33 Number 1 (104)

Iwona GROBELNA
Uniwersytet Zielonogórski, Wydział Elektrotechniki, Informatyki i Telekomunikacji

REGUŁOWY MODEL LOGICZNY REKONFIGUROWALNEGO STEROWNIKA LOGICZNEGO DO WERYFIKACJI I SYNTEZY

Streszczenie. Artykuł przedstawia regułowy model logiczny rekonfigurowalnego sterownika logicznego opisanego za pomocą interpretowanej sieci Petriego, która jest formalną specyfikacją zachowania systemów dyskretnych. Model logiczny, jako abstrakcyjny opis, nadaje się zarówno do formalnej weryfikacji, jak i syntezy logicznej. W pracy są rozpatrywane różne warianty opisu reguł.

Słowa kluczowe: rekonfigurowalny sterownik logiczny, model logiczny, interpretowane sieci Petriego, formalna weryfikacja, synteza

RULE-BASED LOGICAL MODEL OF RECONFIGURABLE LOGIC CONTROLLER FOR VERIFICATION AND SYNTHESIS

Summary. The article presents rule-based logical model of reconfigurable logic controller, by means of Control Interpreted Petri Nets, which are formal specification of discrete systems behavior. Logical model, as an abstract description, is easy to formally verify and to synthesize. In the paper, various rules notations are discussed.

Keywords: reconfigurable logic controller, logical model, Control Interpreted Petri Nets, formal verification, synthesis

1. Wprowadzenie

Sterowniki logiczne [1] są obecnie najczęściej projektowane pod konkretnego klienta, który ma ściśle określone wymagania. Rekonfigurowalne układy FPGA (ang. *Field Programmable Gate Array*) [1-6] są coraz częściej wykorzystywane w urządzeniach codziennego użytku, a także w wyspecjalizowanych systemach. Podejmowane są udane próby połączenia układów FPGA z istniejącymi systemami, przykładowo z systemem Microsoft Windows [7],

w celu zapewnienia bezpiecznych, krytycznych czasowo operacji. Implementacja w układach FPGA jest ostateczna, zwłaszcza w przypadku urządzeń małoseryjnych. W przypadku urządzeń wieloseryjnych implementacja w układach FPGA jest prototypowa, później na masową skalę, są stosowane układy ASIC [6]. Do zalet układów FPGA [1] należy ich uniwersalność i elastyczność, krótki czas opracowywania prototypu, możliwość rekonfiguracji, a także niski koszt opracowania i weryfikacji projektu.

Zachowanie rekonfigurowalnych sterowników logicznych jest specyfikowane przy użyciu różnych formalizmów [8]: interpretowanych, sterujących sieci Petriego (zwanych także interpretowanymi sieciami Petriego sterowania, ang. *Control Interpreted Petri Nets*), sieci SFC (ang. *Sequential Function Charts*) czy też diagramów języka UML (maszyn stanów).

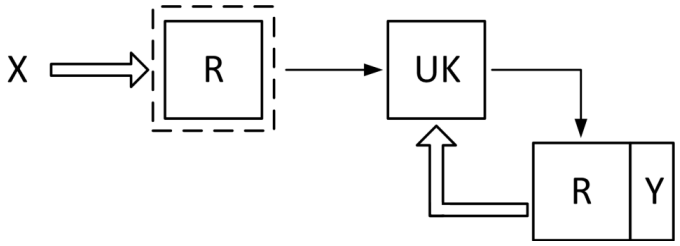
Specyfikacja sterownika logicznego jest pierwszym etapem w procesie jego projektowania oraz tworzenia. Niezmiernie ważne jest zatem, aby spełniała wymagania stawiane jej przez przyszłego użytkownika [9]. Projekt powinien uwzględniać nie tylko podstawowy cel działania sterownika, ale także nieprzewidziany wpływ otoczenia, czy też błędną obsługę ze strony użytkowników. Formalna weryfikacja przygotowanej specyfikacji (jej walidacja) pozwala na wczesne wykrycie błędów wynikających m.in. z nieprawidłowej interpretacji specyfikacji.

W artykule zaprezentowano abstrakcyjny regułowy model logiczny, który opisuje działanie sterownika w zwarty i ściśle określony sposób. Model ten jest wykorzystywany do celów syntezy oraz weryfikacji modelowej i stanowi format pośredni opisujący zachowanie projektowanego sterownika logicznego [10-13].

Rozdział drugi rozpoczyna się ogólnymi informacjami, dotyczącymi proponowanego modelu logicznego, następnie przedstawia definicję zmiennych i ich początkowych wartości. Kolejno zostały omówione różne warianty opisu reguł. Uwzględniono także zmiany wartości sygnałów wejściowych oraz wyjściowych. Rozdział trzeci prezentuje zagadnienia związane z formalną weryfikacją przygotowanego modelu logicznego, zaś rozdział czwarty koncentruje się na jego syntezie. Artykuł kończy się podsumowaniem.

2. Regułowy model logiczny

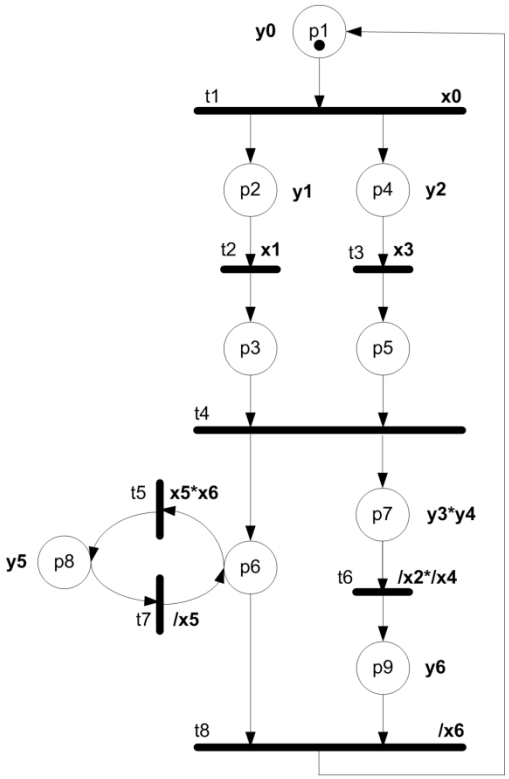
Model logiczny otrzymany na podstawie interpretowanej sieci Petriego jest przedstawiony na poziomie RTL (poziom przesłań międzyrejestrowych, ang. *Register Transfer Level*) w taki sposób, że jest łatwo syntezowalny jako rekonfigurowalny sterownik logiczny. Sam model odzwierciedla zachowanie automatu cyfrowego Moore'a [14] z rejestrem wejść (opcjonalnie) oraz rejestrem wyjść (rys. 1). Układ kombinacyjny (*UK*) steruje zachowaniem systemu i operuje na stanach wewnętrznych układu.



Rys. 1. Automat cyfrowy Moore'a z rejestrem wejść i wyjść
Fig. 1. Moore automaton with input and output registers

Informacje w modelu logicznym są pogrupowane zgodnie z przynależnością do poszczególnych kategorii, omówionych w dalszej części rozdziału, które rozpoczynają się słowami kluczowymi (odpowiednio):

- *VARIABLES* – zmienne wykorzystywane w modelu logicznym,
- *INITIALLY* – początkowe wartości zmiennych,
- *TRANSITIONS* – *o*pis reguł (tranzycji),
- *OUTPUTS* – *a*ktywność sygnałów wyjściowych,
- *INPUTS* – aktywność sygnałów wejściowych.



Rys. 2. Przykładowa interpretowana sieć Petriego dla procesu sterowania
Fig. 2. Sample Control Interpreted Petri Net

Interpretowana sieć Petriego uwzględnia właściwości kontrolowanych obiektów. Stany lokalne zmieniają się po realizacji tranzycji, co z kolei ma miejsce przy wystąpieniu oczekiwanego zdarzenia. Warunki tranzycji są skojarzone z sygnałami wejściowymi sterownika

logicznego, zaś miejsca – z jego sygnałami wyjściowymi (wyjścia typy Moore'a). Występujące jednocześnie stany lokalne określają stan globalny.

Zasady funkcjonowania interpretowanej sieci Petriego [8, 15, 16] (przykład na rys. 2) są zapisywane formalnie przy wykorzystaniu logiki temporalnej. Taka reprezentacja logiczna dobrze opisuje zarówno strukturę, jak i zachowanie sieci.

2.1. Definicja zmiennych i ich początkowych wartości

Model logiczny zawiera definicje zmiennych (słowo kluczowe *VARIABLES*) oraz ich początkowe wartości (słowo kluczowe *INITIALLY*). Elementy interpretowanej sieci Petriego (miejsca, sygnały wejściowe i wyjściowe) są bezpośrednio odwzorowane w modelu logicznym jako zmienne, przy czym każdy element jest traktowany jako osobna zmienna binarna typu logicznego.

Zmienne kodujące miejsca są aktywne (przyjmują wartość logiczną prawdy), jeżeli dane miejsce zawiera żeton, w przeciwnym razie są nieaktywne (przyjmują wartość logiczną fałszu). W danej chwili wiele miejsc lokalnych oznakowanych w tym samym stanie globalnym może jednocześnie zawierać żeton (procesy współbieżne).

Zmienne kodujące sygnały wejściowe oraz wyjściowe przyjmują wartość logiczną prawdy, jeżeli dany sygnał jest aktywny, w przeciwnym razie przyjmują wartość logiczną fałszu. W danej chwili wiele sygnałów wejściowych/wyjściowych może być aktywnych jednocześnie, stąd wiele zmiennych wejściowych/wyjściowych może mieć przypisaną wartość logiczną prawdy.

Model logiczny utworzony na podstawie sieci z rys. 2 zawiera kolejno definicję miejsc, sygnałów wejściowych oraz wyjściowych:

```
VARIABLES
  places:  p1, p2, p3, p4, p5, p6, p7, p8, p9
  inputs:  x0, x1, x2, x3, x4, x5, x6
  outputs: y0, y1, y2, y3, y4, y5, y6
```

Zdefiniowane zmienne przyjmują pewne określone wartości początkowe:

```
INITIALLY
   p1; !p2; !p3; !p4; !p5; !p6; !p7; !p8; !p9;
  !x0; !x1; !x2; !x3; !x4; !x5; !x6;
   y0; !y1; !y2; !y3; !y4; !y5; !y6;
```

Negacja zmiennej wyrażona wykrzyknikiem poprzedzającym nazwę zmiennej oznacza, że zmienna przyjmuje logiczną wartość początkową fałszu, brak negacji oznacza logiczną wartość prawdy.

Alternatywnym podejściem jest wykorzystanie logiki wielowartościowej do reprezentacji zmiennych i wartości, jakie mogą one przyjmować. Zmienna reprezentująca miejsca sieci Petriego mogłaby wtedy przyjmować jedną z wartości ze zbioru stanów globalnych. Zmienne

reprezentujące sygnały wejściowe oraz wyjściowe należałoby jednak nadal traktować jako osobne zmienne, przyjmujące wartości prawdy/fałszu, mając na uwadze fakt, że wiele sygnałów wejściowych lub wyjściowych może być aktywnych jednocześnie. W przypadku małej liczby takich sygnałów można by się pokusić o użycie logiki wielowartościowej, tak jak w przypadku definicji miejsc. Zmienna reprezentująca sygnały przyjmowałaby wtedy jedną z wartości ze zbioru sygnałów (sygnały występujące pojedynczo lub w grupach). Logika binarna jednak lepiej nadaje się jednocześnie do logicznej syntezy oraz weryfikacji modelowej.

2.2. Opis reguł

Model logiczny poza definicją zmiennych zawiera także zbiór reguł (słowo kluczowe *TRANSITIONS*), które opisują zachowanie projektowanego systemu i określają zmiany wartości zmiennych z upływem czasu. Tranzycje są interpretowane jako wspomniane zbiory reguł.

2.2.1. Orientacja na tranzycje

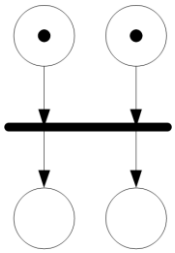
Każda reguła (tranzycja) jest przedstawiona w osobnym wierszu i zaczyna się nazwą tranzycji wraz z dwukropkiem. Nazwa tranzycji stanowi etykietę danej reguły. Reguła jest zbudowana z dwóch części.

Część pierwsza zawiera warunki realizacji tranzycji, a zatem nazwy aktywnych miejsc oraz nazwy sygnałów wejściowych (jeżeli takie są wymagane), koniecznych do realizacji tranzycji. Jeżeli warunek zawiera więcej niż jedną zmienną, zmienne są połączone logicznym operatorem *i* (zapisanym jako &). Możliwe jest także łączenie zmiennych logicznym operatorem *lub* (zapisanym jako |), co może wystąpić przy aktywacji tranzycji jednym z wielu sygnałów wejściowych. Podobnie jak w przypadku początkowych wartości zmiennych, zmienna może przyjmować wartość logiczną prawdy (miejsce aktywne/sygnał wejściowy aktywny) lub wartość logiczną fałszu (sygnał wejściowy nieaktywny).

Druga część reguły przedstawia zmianę znakowania miejsc sieci Petriego. Zastosowany operator logiki temporalnej *X* wskazuje tutaj, że zmiana znakowania sieci nastąpi w następnym stanie systemu. Analogicznie do wcześniejszych możliwych wartości zmiennych, po realizacji tranzycji pewne miejsca mogą stać się aktywne (miejsca wyjściowe tranzycji) lub nieaktywne (miejsca wejściowe tranzycji, nazwy tych zmiennych są poprzedzone wykrzyknikiem). Formalny zapis tranzycji w modelu logicznym jest przedstawiony poniżej:

nazwa_tranzycji: warunki_realizacji_tranzycji -> X zmiana_znakowania_miejsc;

Proponowane rozwiązanie skupia się na tranzycjach (rys. 3). Uwzględniane są miejsca wejściowe tranzycji, warunki realizacji tranzycji (odwołujące się do odpowiednich kombinacji sygnałów wejściowych) oraz miejsca wyjściowe tranzycji.



Rys. 3. Orientacja na tranzycje
Fig. 3. Transition-oriented model

Tranzycje sieci z rys. 2 są przedstawione jako osobne reguły. Realizacja każdej tranzycji powoduje zmianę znakowania jej miejsc wejściowych oraz wyjściowych. Przykładowo, realizacja tranzycji *t*1 spowoduje usunięcie żetonu z miejsca *p*1 (wyrażone przez !*p*1) oraz dodanie żetonu do dwóch miejsc rozpoczynających dwa procesy współbieżne: *p*2 oraz *p*4 (wyrażone przez *p*2 *&* *p*4). Zbiór reguł w modelu logicznym (dla sieci z rys. 2) przyjmuje postać:

```
TRANSITIONS
t1: p1 & x0 -> X (!p1 & p2 & p4);
t2: p2 & x1 -> X (!p2 & p3);
t3: p4 & x3 -> X (!p4 & p5);
t4: p3 & p5 -> X (!p3 & !p5 & p6 & p7);
t5: p6 & x5 & x6 -> X (!p6 & p8);
t6: p7 & !x2 & !x4 -> X (!p7 & p9);
t7: p8 & !x5 -> X (p6 & !p8);
t8: p6 & p9 & !x6 -> X (!p6 & !p9 & p1);
```

Miejsca, które nie są wymienione w danej regule, nie zmieniają swojego znakowania po realizacji opisywanej tranzycji. Oznacza to, że dana reguła nie ma wpływu na zmianę znakowań miejsc, które nie są w niej wymienione. Reguły dotyczą zatem realizacji tranzycji sieci Petriego, a tym samym zmiany znakowania miejsc. Sytuacje, w których tranzycja nie może zostać zrealizowana, nie są uwzględniane, przyjmując domniemanie, że aktywne miejsca utrzymują wtedy swoje znakowanie.

Proponowane podejście jest inercyjnym opisem zorientowanym na tranzycje (podejście wzorowane na pracach [15-17]). W pracy [18], z uwagi na obecność wyjść typu Mealy'ego, dodatkowo oprócz miejsc i sygnałów wejściowych występują także sygnały wyjściowe związane z realizacją danej tranzycji.

Podobne rozwiązanie zostało zaproponowane także w [19], gdzie opis tranzycji jest przedstawiony za pomocą stanów aktualnych (*p*) oraz stanów następnych (*p'*). Oprócz wymienienia miejsc, których znakowanie się zmienia, zostają wymienione także miejsca, które utrzymują swoje aktywne bądź nieaktywne znakowanie. Całość jest zapisana wtedy

w postaci równania, które, dla przykładowej tranzycji *t*1 z interpretowanej sieci Petriego przedstawionej na rys. 2, wyglądałoby następująco (po uwzględnieniu sygnałów wejściowych):

```
ft1 = (p1 & x0) & !p1' & p2' & p4' & [(p3 <=> p3') & (p5 <=> p5')
      & (p6 <=> p6') & (p7 <=> p7') & (p8 <=> p8') & (p9 <=> p9')]
```

Do dalszych badań został właśnie wybrany opis zorientowany na tranzycje. Rozwiązanie to wydaje się być proste w zastosowaniu przez przyszłych użytkowników, gdyż zapisanie takich reguł nie zajmuje dużo czasu i jest zadaniem intuicyjnym. Ponadto, transformacja tak zapisanych reguł do opisu modelu w formacie wejściowym do narzędzia weryfikującego NuSMV oraz do syntezowalnego kodu w języku opisu sprzętu VHDL może zostać łatwo zautomatyzowana, gdyż odbywa się według ściśle określonych zasad.

Jednakże, w dalszej części rozdziału pokrótce zostały także omówione inne możliwości opisu reguł – opis zorientowany na tranzycje i miejsca, na same miejsca oraz na warunki początkowe i końcowe.

2.2.2. Orientacja na tranzycje i miejsca

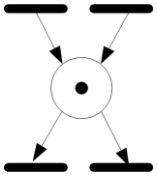
Alternatywnym rozwiązaniem mógłby być opis reguł zorientowany na tranzycje i miejsca [15]. Oprócz warunków zmiany znakowania sieci przedstawionych w opisie zorientowanym na tranzycje, wymienione byłyby także warunki utrzymania znakowania aktywnych miejsc. Obie części opisu byłyby komplementarne, to znaczy dla każdej reguły (tranzycji) zostałaby zapisana jedna linia, opisująca zachowanie układu w przypadku braku możliwości realizacji danej tranzycji. Do opisu modelu doszłyby także warunki utrzymania znakowania miejsc. Mogą one występować po warunkach związanych z realizacją tranzycji (najpierw warunki realizacji tranzycji, a następnie warunki utrzymania znakowania miejsc) lub mogą się z nimi przeplatać (będą rozpatrywane kolejne tranzycje, dla każdej z nich najpierw warunek realizacji, a następnie utrzymania znakowania). Warunki utrzymania znakowania miejsc dla interpretowanej sieci Petriego z rys. 2, uzupełniające reguły związane z realizacją tranzycji (model zorientowany na tranzycje), przedstawiono poniżej:

```
p1 & !x0 -> X p1;
p2 & !x1 -> X p2;
p4 & !x3 -> X p4;
p3 & !p5 -> X p3;
p5 & !p3 -> X p5;
p6 & !(x5 & x6 | p9 & !x6) -> X p6;
p7 & (x2 | x4) -> X p7;
p8 & x5 -> X p8;
p9 & x6 | p9 & !x6 & !p6 -> X p9;
```

2.2.3. Orientacja na miejsca

Innym podejściem byłby opis zorientowany na miejsca [15,16]. Reguły dotyczyłyby wtedy nie tranzycji, ale poszczególnych miejsc (rys. 4). Każda reguła zawierałaby zmianę znakowania miejsc po realizacji tranzycji oraz utrzymanie miejsc w sytuacji przeciwnej. Formalny zapis reguły przedstawiono poniżej:

nazwa_miejsca: warunki_realizacji_tranzycji ->
X zmiana_znakowania_miejsc | X utrzymanie_znakowania_miejsc;



Rys. 4. Orientacja na miejsca
Fig. 4. Place-oriented model

Jeżeli dane miejsce posiada więcej niż jedną tranzycję wyjściową, liczba reguł wzrasta. I tak, w sieci na rys. 2 miejsce *p*6 posiada dwie tranzycje wyjściowe (*t*5 i *t*8), stąd też dla tego miejsca zostają zapisane dwie oddzielne reguły. Przykład opisu modelu zorientowanego na miejsca (dla sieci z rys. 2):

```
p1: p1 & x0 -> X (p2 & p4) | X p1;
p2: p2 & x1 -> X p3 | X p2;
p3: p3 & p5 -> X (p6 & p7) | X p3;
p4: p4 & x3 -> X p5 | X p4;
p5: p3 & p5 -> X (p6 & p7) | X p5;
p6: p6 & x5 & x6 -> X p8 | X p6;
p6: p6 & p9 & !x6 -> X p1 | X p6;
p7: p7 & !x2 & !x4 -> X p9 | X p7;
p8: p8 & !x5 -> X p6 | X p8;
p9: p6 & p9 & !x6 -> X p1 | X p9;
```

2.2.4. Orientacja na warunki początkowe i końcowe

Ostatnią rozpatrywaną możliwością była orientacja na warunki początkowe i końcowe (ang. *preconditions/postconditions*) [15], gdzie kolejno są podane warunki realizacji tranzycji, zmiana znakowania miejsc wskutek realizacji tranzycji oraz utrzymanie aktualnego znakowania miejsc, w przypadku gdy tranzycja nie może zostać odpalona. Formalny zapis takiej reguły, podzielonej na trzy części, przedstawiono poniżej:

warunki początkowe (ang. *preconditions*):
nazwa_tranzycji: warunki_realizacji_tranzycji -> X aktywowana_tranzycja;
warunki końcowe (ang. *postconditions*):
nazwa_tranzycji: X znakowane_miejsce;

utrzymanie znakowania:

miejsce & nierealizowana_tranzycja -> X miejsce;

Przykład opisu modelu zorientowanego na warunki początkowe oraz końcowe (dla interpretowanej sieci Petriego z rys. 2):

```
PRECONDITIONS
t1: p1 & x0 -> X t1;
t2: p2 & x1 -> X t2;
t3: p4 & x3 -> X t3;
t4: p3 & p5 -> X t4;
t5: p6 & x5 & x6 -> X t5;
t6: p7 & !x2 & !x4 -> X t6;
t7: p8 & !x5 -> X t7;
t8: p6 & p9 & !x6 -> X t8;

POSTCONDITIONS
t1: X (p2 & p4);
t2: X p3;
t3: X p5;
t4: X (p6 & p7);
t5: X p8;
t6: X p9;
t7: X p6;
t8: X p1;

HOLDING
p1 & !t1 -> X p1;
p2 & !t2 -> X p2;
p3 & !t4 -> X p3;
p4 & !t3 -> X p4;
p5 & !t4 -> X p5;
p6 & !t5 & !t8 -> X p6;
p7 & !t6 -> X p7;
p8 & !t7 -> X p8;
p9 & !t8 -> X p9;
```

Należy tutaj zauważyć, że w pierwszej części, dotyczącej warunków początkowych (słowo kluczowe *PRECONDITIONS*), występują odwołania do tranzycji, miejsc oraz sygnałów wejściowych. W części drugiej poświęconej warunkom końcowym (słowo kluczowe *POSTCONDITIONS*) są jedynie rozpatrywane tranzycje oraz miejsca. Podobnie ma to miejsce w części trzeciej (słowo kluczowe *HOLDING*), gdzie dodatkowo występuje negacja tranzycji oznaczająca niezrealizowaną tranzycję. I tak, warunkiem początkowym tranzycji $t1$ jest aktywne miejsce *p1* oraz aktywny sygnał wejściowy $x0$. Warunek końcowy tranzycji $t1$ to aktywne miejsca $p2$ i $p4$. Brak realizacji tranzycji $t1$ połączony z aktywnym miejscem $p1$ powoduje utrzymanie znakowania tego miejsca.

2.3. Zmiana wartości sygnałów wyjściowych

Kolejnym rozpatrywanym elementem sterownika logicznego odwzorowanym w modelu logicznym są sygnały wyjściowe. Interpretowane są one jako przypisania sygnałów wyjściowych do odpowiednich miejsc, zgodnie z zasadą:

dla każdego miejsca przypisz (jeżeli zdefiniowano) aktywne sygnały wyjściowe

Sygnały wyjściowe są definiowane po użyciu słowa kluczowego *OUTPUTS* i rozpatrywane dla kolejnych miejsc sieci Petriego. Z lewej strony strzałki jest podane miejsce, którego aktywne znakowanie pociąga za sobą aktywność wskazanych z prawej strony sygnałów wyjściowych. Jeżeli aktywność danego sygnału wyjściowego jest związana z więcej niż jednym miejscem, sygnał ten występuje wielokrotnie z prawej strony strzałki przy różnych miejscach. Proponowany zapis dotyczy wyjść typu Moore'a [14], gdzie stan wyjścia zależy tylko od wewnętrznego stanu układu. Formalny zapis aktywnych sygnałów wyjściowych w modelu logicznym przedstawiono poniżej:

aktywne_miejsce -> aktywne_sygnały_wyjściowe;

gdzie

aktywne_sygnały_wyjściowe = aktywny_sygnał_1 & aktywny_sygnał_2 & ... ;

Sygnały wyjściowe są zatem przypisane do miejsc, w których są one aktywne. Przykładowo (dla sieci z rys. 2), sygnał wyjściowy $y0$ jest aktywny jedynie przy aktywnym znakowaniu miejsca $p1$, zaś aktywne znakowanie miejsca $p7$ pociąga za sobą aktywność sygnałów wyjściowych $y3$ i $y4$. Wszystkie pozostałe sygnały wyjściowe, które nie występują z prawej strony danej reguły (dla poszczególnych miejsc), pozostają domyślnie nieaktywne. Możliwe byłoby także jawne wskazanie aktywności/nieaktywności sygnału wyjściowego jak w [15], jednakże proponowany zapis wydaje się być intuicyjny i nie wymaga dodatkowych informacji, które mogłyby wpłynąć negatywnie na jego czytelność. Sygnały wyjściowe w modelu logicznym (dla przykładowej sieci z rys. 2) zapisane są w postaci:

```
OUTPUTS
p1 -> y0;
p2 -> y1;
p4 -> y2;
p7 -> y3 & y4;
p8 -> y5;
p9 -> y6;
```

Sygnały wyjściowe sterownika kontrolują zachowanie systemu i sterują procesami. Dlatego ich poprawne działanie jest ważnym elementem funkcjonowania systemu.

2.4. Zmiana wartości sygnałów wejściowych

Zmiany sygnałów wejściowych (słowo kluczowe *INPUTS*) także są definiowane w modelu logicznym. Są one jednak wykorzystywane tylko przy weryfikacji modelowej (tworzeniu opisu modelu). Model w języku opisu sprzętu nie zmienia bowiem bezpośrednio wartości sygnałów wejściowych. Sygnały wejściowe pochodzące od różnych obiektów, systemu nadrzędnego lub operatora systemu są rozpatrywane podobnie jak sygnały wyjściowe dla kolejnych miejsc sieci Petriego. Z lewej strony strzałki podane jest miejsce, a z prawej strony

istotne sygnały wejściowe w zależności od aktywności znakowania miejsca. Zawsze są więc podane dwie możliwe wartości sygnału wejściowego (*sygnał aktywny/sygnał nieaktywny*), co jest zapisane w postaci:

nazwa_sygnału_wejściowego | *!nazwa_sygnału_wejściowego*

Jeżeli przy aktywnym znakowaniu danego miejsca są spodziewane zmiany wielu sygnałów wejściowych, to z prawej strony jest zamieszczony ich zbiór:

aktywne_miejsce -> sygnały_wejściowe;
gdzie
sygnały_wejściowe = sygnał_1 & sygnał_2 & ... ;
sygnał_i przyjmuje jedną z wartości (!sygnał_i | sygnał_i)

Sygnały wejściowe są przypisane do miejsc, w których są istotne i mogą stać się aktywne. W każdym innym stanie, domyślnie sygnał pozostaje w stanie nieaktywnym. Przykładowo, startowy sygnał wejściowy $x0$ może być uaktywniony, gdy znakowanie sieci Petriego obejmuje miejsce początkowe $p1$. W pozostałych przypadkach (domyślnie) sygnał wejściowy *x0* jest nieistotny i nieaktywny. Takie zachowanie sygnałów wejściowych nawiązuje także do sytuacji początkowej, w której wszystkie sygnały są nieaktywne. Zmiany wartości sygnałów wejściowych w modelu logicznym przedstawiono poniżej:

```
INPUTS
p1 -> !x0 | x0;
p2 -> !x1 | x1;
p4 -> !x3 | x3;
p6 -> (!x5 | x5) & (!x6 | x6);
p7 -> (!x2 | x2) & (!x4 | x4);
p8 -> !x5 | x5;
p6 & p9 -> !x6 | x6;
```

Jeżeli jednak pewien sygnał wejściowy początkowo jest aktywny (przyjmuje początkową logiczną wartość prawdy), to staje się ona domyślną wartością tego sygnału. W miejscach określonych w sekcji sygnałów wejściowych modelu logicznego sygnał ten może stać się nieaktywny, w pozostałych miejscach pozostaje aktywny (zachowuje logiczną wartość domyślną prawdy).

3. Weryfikacja modelowa proponowanego modelu logicznego

Technika weryfikacji modelowej [20-22] jest jedną z metod formalnej weryfikacji specyfikacji systemów, oprócz innych jak na przykład automatyczne dowodzenie twierdzeń (ang. *theorem proving*). Weryfikacja jest przeprowadzana automatycznie przez narzędzia wnioskowania komputerowego (narzędzia typu *model checker*, przykładowo NuSMV [23]).

Dane wejściowe to opis modelu (przedstawiony w języku opisu danego narzędzia) oraz lista wymagań stawianych projektowanemu systemowi (zawierająca zdefiniowane za pomocą logiki temporalnej właściwości zapisane w charakterystycznym dla danego narzędzia języku specyfikacji). Należy mieć na uwadze fakt, że tylko wyspecyfikowane właściwości zostaną sprawdzone. Narzędzie weryfikujące przeprowadza weryfikację systemu poprzez sprawdzenie, czy dany model spełnia stawiane mu wymagania. Danymi wyjściowymi jest odpowiedź, czy model i jego specyfikacja są spójne, a w przypadku, gdy tak nie jest – dodatkowo zostaje wygenerowany kontrprzykład.

Model logiczny, omówiony w poprzednim rozdziale, jest przekształcany do formatu wejściowego narzędzia weryfikującego NuSMV [23, 24] (wersja 2.5.2) według ściśle określonych reguł, przedstawionych w pracach [10-13]. Opis weryfikowalnego modelu przedstawia za pomocą reguł zmiany wartości zdefiniowanych zmiennych – zmianę znakowania miejsc interpretowanej sieci Petriego, zmianę wartości sygnałów wejściowych i wyjściowych. Sam model może zostać poddany wstępnej symulacji. Po dostarczeniu przez użytkownika listy wymagań jest możliwe przeprowadzenie formalnej weryfikacji.

Właściwości, jakie projektowany system ma spełniać, są definiowane przy wykorzystaniu logiki temporalnej [25]: logiki liniowej LTL lub logiki rozgałęzionej CTL. Trafny dobór właściwości podlegających weryfikacji jest zadaniem trudnym i wymaga dużej uwagi ze strony projektanta lub inżyniera. Nie jest możliwe sprawdzenie wszystkich potencjalnie możliwych właściwości systemu, są tylko sprawdzane te właściwości, które zostaną zdefiniowane. Osoba weryfikująca powinna więc dobrać takie właściwości, które są istotne z punktu widzenia działania systemu [22]. W przypadku złożonych systemów nigdy jednak nie będziemy mieć pewności, czy wyspecyfikowano wszystkie ważne wymagania, czy też pewne z nich pominięto. Minimalny zbiór podlegający weryfikacji zawiera właściwości krytyczne, które decydują o zdrowiu czy nawet życiu osób, których los zależy od zaprojektowanego systemu (przemysł lotniczy, samochodowy, energetyczny, medyczny itp.).

4. Synteza proponowanego modelu logicznego

Połączenie układów FPGA [2, 3, 5] jako docelowej platformy sprzętowej z językiem opisu sprzętu VHDL zapewnia dużą niezawodność, wysoką szybkość działania oraz bezpieczeństwo. Dodatkowo, istnieje możliwość późniejszej modyfikacji działającego już układu, co ma duże znaczenie praktyczne, jak też zablokowania możliwości odczytu zawartości układu FPGA, co podnosi jego bezpieczeństwo.

Metodologia projektowania na poziomie RTL pozwala na przekształcenie algorytmu w realizację sprzętową oraz pozwala na zastosowanie koncepcji zmiennych i sekwencyjnego

wykonywania operacji [1]. Specyfikacją akceptowaną przez narzędzia syntezy na poziomie przesłań międzyrejestrowych jest opis projektu w języku opisu sprzętu VHDL [26]. Stąd też model logiczny jest przekształcany do syntezowalnego kodu właśnie w języku VHDL.

Bezpośrednia implementacja współbieżnych sterowników w układach FPGA jest podobna do regułowej realizacji sterowników logicznych na podstawie klasycznych diagramów sekwencji [2]. Realizacja tranzycji jest zsynchronizowana z aktywnym zboczem zegara.

Celowo zostało zachowane wysokie podobieństwo pomiędzy modelem weryfikowalnym a modelem syntezowalnym. Ułatwia to znacznie proces automatycznej transformacji modelu logicznego do docelowych modeli, a także zapewnia ich spójność.

Interpretowana, sterująca sieć Petriego, która stanowi punkt wyjściowy do modelu logicznego, jest siecią bezpieczną, co oznacza, że każde miejsce może zawierać co najwyżej jeden żeton. Miejsca mogą być wtedy zaimplementowane przy użyciu prostych przerzutników, ponieważ znakowanie miejsc jest wyrażone zmienną binarną (1⁄0 – aktywne/nieaktywne znakowanie miejsca) [27]. Ilość przerzutników potrzebnych do obsługi miejsc, z zastosowaniem kodowania one-hot, równa się liczbie tych miejsc (a więc liczbie stanów lokalnych) [5, 28].

Przekształcenie modelu logicznego do języka VHDL odbywa się według ściśle określonych reguł, przedstawionych w pracach [11, 12]. Zastosowane zostało kodowanie *one-hot* (zwane również kodowaniem izomorficznym miejsc) [5, 27, 28], które jest najwierniejszym (i najprostszym) odwzorowaniem modelu logicznego, jednakże może powodować większe zużycie zasobów sprzętowych. Dla każdego miejsca jest wtedy tworzony jeden przerzutnik, którego nazwa pokrywa się z identyfikatorem odpowiedniego miejsca. Przerzutnik ustawia wartość 1, gdy dane miejsce zawiera żeton, w przeciwnym wypadku utrzymuje wartość 0. Ponadto, kodowanie *one-hot* jest zalecane przy implementacji w układach FPGA, a nawet postrzegane jako najbardziej efektywna metoda kodowania stanów [26] (między innymi w układach FPGA firmy *Xilinx*), zwłaszcza dla małych automatów.

Utworzony model w języku VHDL można poddać wstępnej symulacji, przykładowo w środowisku *Active HDL*. W proponowanym syntezowalnym modelu jest możliwa jednoczesna (przy narastającym takcie zegara) realizacja tranzycji niepozostających ze sobą w konflikcie.

Następnie jest możliwe przeprowadzenie procesu syntezy przygotowanego układu, przykładowo w środowisku *XILINX PlanAhead*. Model przygotowany w języku VHDL nie zawiera niesyntezowalnych konstrukcji, którymi są między innymi konstrukcje z parametrami czasowymi lub struktury danych o niezdefiniowanym rozmiarze [26].

Synteza jest przeprowadzana w formie szybkiego prototypowania (ang. *Rapid Prototyping*) [3,6], co we współczesnej metodologii projektowania układów cyfrowych pozwala na

częstą weryfikację (symulację, analizę) tworzonego systemu. Głównym jej celem jest sprawdzenie, czy zaprojektowany system w ogóle działa, przy czym układ niekoniecznie jest zoptymalizowany. Zagadnienia związane z optymalizacją układu w niektórych sytuacjach mogą mieć jednak duże znaczenie [5, 27, 29].

5. Podsumowanie

Artykuł proponuje zastosowanie regułowego modelu logicznego jako formatu pośredniego pomiędzy pierwotną specyfikacją rekonfigurowalnego sterownika logicznego a modelem weryfikowalnym i syntezowalnym. Początkowa interpretowana sieć Petriego może zostać poddana animacji czy też formalnej analizie. Następnie sieć jest zapisywana w postaci regułowej specyfikacji, która jest wykorzystywana zarówno do formalnej weryfikacji, jak i syntezy logicznej. Formalna weryfikacja modelowa jest przeprowadzana z użyciem narzędzia komputerowego wnioskowania NuSMV. Utworzony na podstawie modelu logicznego program w języku opisu sprzętu VHDL może zostać poddany syntezie, a także symulacji behawioralnej czy też funkcjonalnej (w środowiskach CAD).

Zgodność modelu i specyfikacji wymagań może zostać sprawdzona przed implementacją rzeczywistego systemu, co w wielu przypadkach może zaoszczędzić czas i ograniczyć koszty powstania końcowego produktu.

W celu szerszego przetestowania proponowanych rozwiązań została przygotowana aplikacja służąca do automatycznej konwersji regułowej formy specyfikacji rekonfigurowalnego sterownika logicznego do modelu podlegającego weryfikacji oraz do modelu podlegającego syntezie logicznej. Badania oparto na układach firmy *Xilinx*, która jest pionierem w dziedzinie układów FPGA.

DODATKOWE INFORMACJE

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI
Lubuskie
UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Autor jest stypendystą w ramach Poddziałania 8.2.2 „Regionalne Strategie Innowacji”, Działania 8.2 ”Transfer Wiedzy”, Priorytetu VIII „Regionalne Kadry Gospodarki” Programu Operacyjnego Kapitał Ludzki współfinansowanego ze środków Europejskiego Funduszu Społecznego Unii Europejskiej i z budżetu państwa.

BIBLIOGRAFIA

1. Łuba T.: Programowalne układy przetwarzania sygnałów i informacji. WKŁ, Warszawa 2008.
2. Adamski M., Węgrzyn M., Węgrzyn A.: Safe reconfigurable logic controllers design. Measurements Models Systems and Design, [ed.] Korbicz J., Warszawa 2007, s. 343÷370.
3. Ahrends S.: Neue Ansätze für effizientes Rapid Prototyping von Embedded Systemen. National Instruments, Embedded Computing Conference, Winterthur 2008.
4. Barkalov A., Titarenko L.: Evolution of Programmable Logic. Logic Synthesis for FSM-Based Control Units, Lecture Notes in Electrical Engineering, 2009, Vol. 53, s. 53÷75.
5. Nemec J.: Stoke the fires of FPGA design. Electronic design, Oct. 25, 1994, s. 97÷105.
6. Pawłowski M., Skorupski A.: Projektowanie złożonych układów cyfrowych. WKŁ, Warszawa 2010.
7. Kołek K., Turnau A.: FPGA as a part of MS Windows control environment. Proceedings of the International Multiconference on Computer Science and Information Technology, 2006, s. 401÷406.
8. Adamski M., Chodań M.: Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC. Wydawnictwo Politechniki Zielonogórskiej, 2000.
9. Kropf T.: Introduction to Formal Hardware Verification. ISBN 3-540-65445-3, Springer-Verlag, Berlin, Heidelberg, New York 1999.
10. Grobelna I.: Weryfikacja modelowa interpretowanych sieci Petriego sterowania. Pomiary Automatyka Kontrola, 2011, Vol. 57, nr 6, s. 666÷670.
11. Grobelna I.: Regułowa reprezentacja interpretowanych sieci Petriego sterowania dla potrzeb syntezy i weryfikacji. Pomiary Automatyka Kontrola, 2011, Vol. 57, nr 8, s. 942÷944.
12. Grobelna I.: Formal verification of embedded logic controller specification with computer deduction in temporal logic. Przegląd Elektrotechniczny (przyjęte do druku).
13. Grobelna I., Adamski M.: Model Checking of Control Interpreted Petri Nets,.2011, Mixed Design of Integrated Circuits and Systems – MIXDES 2011: proceedings of the 18th international conference, Polska, 2011, s. 621÷626 (dostępne w IEEE Xplore).
14. Zieliński C.: Podstawy projektowania układów cyfrowych. PWN, Warszawa 2003.
15. Adamski M., Monteiro J. L.: From Interpreted Petri net specification to Reprogrammable Logic Controller Design. Proceedings of the IEEE International Symposium on Industrial Electronics – ISIE 2000, Meksyk, 2000, Vol. 1, s. 13÷19.

16. Adamski M.: Specification and synthesis of Petri net based reprogrammable logic controller. W: Programmable devices and systems 2001 (PDS 2001): a proceedings volume from the 5th IFAC Workshop, London: Pergamon, 2002, s. 95÷100.
17. Adamski M.: Projektowanie układów cyfrowych systematyczną metodą strukturalną. Wydawnictwo Wyższej Szkoły Inżynierskiej w Zielonej Górze, Zielona Góra 1990.
18. Fernandes J. M., Adamski M., Proenca A. J.: VHDL generation from hierarchical Petri net specifications of parallel controllers. IEEE Proceedings – Computers and Digital Techniques, 1997, Vol. 144, no. 2, s. 127÷137.
19. Girault C., Valk R.: Petri Nets for Systems Engineering. A Guide to Modeling, Verification, and Applications. Springer 2003.
20. Clarke E.M., Grumberg O., Peled D.A.: Model checking. The MIT Press, 1999.
21. Emerson E.A.: The Beginning of Model Checking: A Personal Perspective. Lecture Notes in Computer Science. 25 Years of Model Checking: History, Achievements, Perspectives, 2008, s. 27÷45.
22. Huth M.: Some current topics in model checking. J. Software Tools for Technology Transfer 9:1, 2007, s. 25÷36.
23. Cavada R. et al.: NuSMV 2.5 User Manual. pobrane ze strony http://nusmv.fbk.eu/.
24. Grobelna I.: Weryfikacja modelowa z NuSMV. Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, 2011.
25. Ben-Ari M.: Logika matematyczna w informatyce. Klasyka informatyki. WNT, Warszawa 2005.
26. Zwoliński M.: Projektowanie układów cyfrowych z wykorzystaniem języka VHDL. WKŁ, Warszawa 2007.
27. Pardey J., Bolton M.: Logic synthesis of synchronous parallel controllers. Computer Design: VLSI in Computers and Processors, 1991, ICCD'91. Proceedings, s. 454÷457.
28. Adamski M.: A rigorous design methodology for reprogrammable logic controllers. The International Workshop on Discrete-Event System Design, DESDes'01, 2001.
29. De Micheli G.: Synteza i optymalizacja układów cyfrowych. WNT, Warszawa 1998.

Wpłynęło do Redakcji 8 listopada 2011 r.

Abstract

The article presents rule-based logical model of reconfigurable logic controller, by means of Control Interpreted Petri Nets, which are formal specification of discrete systems behavior.

Logical model, as an abstract description, is easy to formally verify and to synthesize. It corresponds to the functionality of Moore automaton (Fig. 1) with input (optionally) and output registers. Logical model, derived from Control Interpreted Petri Net (Fig. 2) contains variables with initial values, rules describing system behavior (transition firings) and changes of input and output signals. In the paper, various rules notations are discussed – model oriented on transitions (Fig. 3), transitions and places, places (Fig. 4) and finally model oriented on preconditions and postconditions.

Model checking technique is used as formal verification method. Logical model is transformed into NuSMV (*model checker*) input format according to some strictly defined rules. It is possible to simulate prepared model description. After delivery of requirements list, which are supposed to be satisfied, model checker tool can formally verify system description.

Logical model can be also transformed into synthesizable code in VHDL, according to some strictly defined rules. For places representation *one-hot* encoding was used, which is recommended for implementation in FPGAs. VHDL model can be firstly simulated, i.e. in *Active HDL* environment. It can be also synthesized in form of rapid prototyping, i.e. in *XILINX PlanAhead* environment.

As a support for testing, an application was developed, which allows automatic conversion of rule-based specification of reconfigurable logic controller into model description for formal verification and into synthesizable code. Researches were based on Xilinx FPGA's.

The result of applying proposed methods is the assurance that verified behavioral specification in temporal logic will be an abstract program of matrix reconfigurable logic controller. So, logic controller program (its implementation) will be valid according to its primary specification. This may shorten the duration time of RLCs development process and, consequently, save money.

Adres

Iwona GROBELNA: Uniwersytet Zielonogórski, Wydział Elektrotechniki, Informatyki i Telekomunikacji, ul. Podgórna 50, 65-246 Zielona Góra, Polska, i.grobelna@iie.uz.zgora.pl.