Marek WĘGRZYN
UNIWERSYTET ZIELONOGÓRSKI

Modelowanie sieci Petriego w języku VHDL

Dr inż. Marek WĘGRZYN

Adiunkt w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Pełni funkcję kierownika Zakładu Inżynierii Komputerowej. Zainteresowania badawcze obejmują zagadnienia projektowania systemów cyfrowych, ze szczególnym uwzględnieniem logiki programowalnej i języków opisu sprzętu (VHDL i Verilog). Członek PTI, POLSPAR, IEEE oraz IFAC (przewodniczący komitetu TC 3.1).

e-mail: M.Wegrzyn@iie.uz.zgora.pl

Streszczenie

Sieć Petriego dobrze nadaje się do modelowania współbieżnych układów cyfrowych, w szczególności do układów sterowania. W celu szybkiego prototypowania takich układów przygotowywane są odpowiadające im modele w językach opisu sprzętu. Opracowywane modele wykorzystywane są zarówno do celów symulacji, jak i syntezy. Implementacja odbywa się z wykorzystaniem programowalnych matryc bramkowych FPGA. Do aktualnie stosowanych języków HDL zalicza się VHDL i Verilog. W referacie przedstawiono sposób modelowania sieci Petriego w języku VHDL. Referat ma charakter przeglądowy.

Słowa kluczowe: Sterownik logiczny, sieci Petriego, modelowanie, synteza, VHDL, FPGA.

Petri net modeling in VHDL

Abstract

Petri nets are used to specification of concurrent Logic Controllers. For rapid prototyping of such systems HDL models are prepared. Models are used for both, simulation and synthesis. As implementation technology, programmable logic, e.g. FPGA devices, is applied. VHDL and Verilog are used in modern CAD systems. In the paper a short overview of VHDL modeling method is presented.

Keywords: Logic Controller, Petri net, modeling, synthesis, VHDL, FPGA.

1. Wstęp

Układy scalone projektowane pod kątem specyficznego zastosowania (ang. *Application Specific Integrated Circuit*, *ASIC*) zrobiły w ciągu ostatnich kilku lat wielką karierę. Wykorzystywane są one wszędzie tam, gdzie zależy na funkcjonalności, elastyczności, małych wymiarach, małym poborze mocy i niezawodności projektowanych urządzeń. Na szczególną uwagę zasługują układy programowane przez użytkownika, które są dostarczane i wykonywane jako uniwersalne cyfrowe układy scalone. Wykorzystanie tego typu układów, w tym szczególnie układów FPGA (ang. *Field Programmable Gate Array*), pozwala na przygotowanie projektów o dużej złożoności, przy minimalizacji ryzyka i kosztów projektowania. Jednakże ich projektowanie wymusza zastosowanie systemów CAD (ang. *Computer Aided Design*). Duże rozmiary realizowanych projektów systemów cyfrowych przekraczają możliwości łatwego, w sposób ręczny, przygotowania projektu. Z tego względu wykorzystanie zaawansowanych pakietów programowych wspomagających projektowanie jest już koniecznością w praktyce inżynierskiej. Zastosowanie systemów CAD znacznie przyspiesza pracę, a czasami wręcz umożliwia ich realizację. Stosowanie specjalnych języków wykorzystywanych do specyfikacji projektów na coraz to wyższym poziomie abstrakcji staje się standardowym elementem nowoczesnych systemów CAD. Aktualnie najczęściej stosowanymi językami tego typu są VHDL i Verilog. Opisane w tych językach projekty są następnie poddawane automatycznej syntezie, czyli generowaniu struktury projektu w postaci listy połączeń, np. format EDIF.

Teoria sieci jest teorią organizacji systemów, którą przeszło 30 lat temu zapoczątkowała rozprawa Carla Adama Petriego. Od czasu ukazania się tej pracy, sieci Petriego znalazły zastosowanie w różnych dziedzinach, wprowadzono nowe klasy sieci wraz z algorytmami ich analizy [10,14]. W porównaniu z innymi modelami systemów, sieci Petriego cechuje przede wszystkim możliwość przedstawienia w sposób jawny zależności przyczynowych w pewnym zbiorze zdarzeń oraz ich relacji współbieżności, tzn. realizacji odpowiednich procesów przez równoległe bloki. Relacja ta ma zasadnicze znaczenie dla całej bazy pojęciowej sieci Petriego [2,5,10]. Aparat teorii sieci Petriego stosuje się w metodach rozwiązywania zadań między innymi z zakresu: projektowania układów cyfrowych, programowania sterowników mikroprocesorowych, projektowania baz danych, syntezy oprogramowania systemowego, monitorowania pracy urządzeń, wykrywania i diagnostyki uszkodzeń itd.

2. Sieć Petriego a współbieżne układy sterowania

Sieć Petriego dobrze nadaje się do modelowania współbieżnych układów cyfrowych. W większości projektów inżynierskich istnieje konieczność skoordynowania częściowo niezależnych procesów o charakterze sekwencyjnym. W większości nietrywialnych przypadków, próba zastąpienia modelu układu przedstawionego sieci Petriego równoważnym modelem w postaci pojedynczego, klasycznego sekwencyjnego automatu cyfrowego [3,4] z góry skazana na niepowodzenie. Nawet w przypadku prostych przykładów o charakterze ilustracyjnym (akademickim) jest to wyjątkowo trudne i niezwykle pracochłonne. Liczba koniecznych do realizacji elementów logicznych wzrasta w gwałtowny sposób wraz ze stopniem równoległości układu (liczbą modelowanych procesów współbieżnych). Poprawny projekt równoważnego automatu sekwencyjnego nie gwarantuje możliwości jego fizycznej implementacji w miejsce zrealizowanego już kontrolera współbieżnego.

W systemach uniwersyteckich do specyfikacji układów opisanych sieciami Petriego wykorzystuje się tekstowy opis regułowy wywodzący się z zapisu struktury sieci w logice sekwentów [1]. Wprowadzony przez Kozłowskiego PNSF (ang. *Petri Net Specification Format*) [9] nie zawiera elementów hierarchii. Zmodyfikowana wersja PNSF stosowana w systemie CONPAR [7] wprowadza elementy hierarchii w sposób niezadawalający projektanta złożonych układów cyfrowych. W równolegle opracowanym formacie PNSF2 [15] uwzględniono również podejścia hierarchicznego oraz wprowadzono pewne, dodatkowe rozszerzenia, jak na przykład specyfikację wyjść typu Moore'a i Mealy'ego. Należy dodatkowo wskazać, że rozpatrywane sieci Petriego przedstawiają sieci synchroniczne, tzn. realizacja dowolnej tranzycji możliwa jest jedynie przy aktywnym zboczu zegara synchronizującego.

W celu dalszej analizy możliwości modelowania sieci Petriego w języku VHDL zostaną przedstawione wybrane zagadnienia z bogatej teorii sieci przedstawiające szczególnie ich strukturę.

Zbiory tranzycji wejściowych i wyjściowych miejsca p definiowane są następująco:

$$^\bullet p = \{t \in \boldsymbol{T} : (t, p) \in \boldsymbol{F}\},$$
$$p^\bullet = \{t \in \boldsymbol{T} : (p, t) \in \boldsymbol{F}\},$$

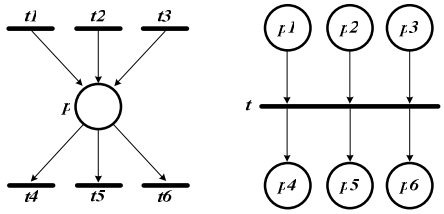
gdzie $\boldsymbol{T}$ – zbiór tranzycji sieci; $\boldsymbol{F}$ – jest relacja przepływu w danej sieci, elementy zbioru $\boldsymbol{F}$ noszą nazwę *łuków*; t – rozpatrywana tranzycja; p – rozpatrywane miejsce.

W podobny sposób określa się zbiory miejsc wejściowych i wyjściowych danej tranzycji t:

$$^\bullet t = \{p \in \boldsymbol{P} : (p, t) \in \boldsymbol{F}\},$$
$$t^\bullet = \{p \in \boldsymbol{P} : (t, p) \in \boldsymbol{F}\},$$

gdzie P – zbiór miejsc sieci.

Rys. 1 przedstawia fragmenty sieci ilustrujące zarówno zbiory wejściowe, jak i wyjściowe obu typów wierzchołków, tzn. miejsc i tranzycji. Oprócz wersji graficznej podano również zapis w postaci odpowiednich zbiorów.



$\bullet p = \{t1, t2, t3\}$ $\quad \bullet t = \{p1, p2, p3\}$

$p\bullet = \{t4, t5, t6\}$ $\quad t\bullet = \{p4, p5, p6\}$

a) b)

Rys. 1. a) Tranzycje wejściowe i wyjściowe miejsca *p*; b) miejsca wejściowe i wyjściowe tranzycji *t*
Fig. 1. a) In/out transitions for place *p*; b) in/out places for transition *t*

Równanie (1) przedstawia ogólną zależność opisującą znacznik w danym miejscu *p*, czego rozwinięciem jest (2).

$$p = ustaw(p) + utrzymaj(p) \tag{1}$$

$$p = \sum_{t_i \in \bullet p} (t_i) + p \,\&\, not\Big(\sum_{t_j \in p\bullet} (t_j)\Big) \tag{2}$$

Stosując prawa de Morgana równanie (2) przyjmuje postać:

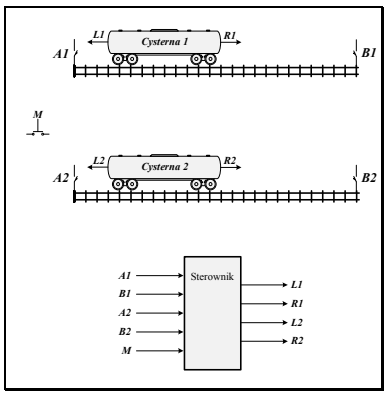
$$p = \sum_{t_i \in \bullet p} (t_i) + p \,\&\, \Big(\prod_{t_j \in p\bullet} not(t_j)\Big) \tag{3}$$

Taka postać jest łatwiejsza do zapisu w wybranym języku opisu sprzętu i będzie stanowić podstawę przy omawianiu różnych metod modelowania sieci Petriego.

3. Modelowanie w języku VHDL

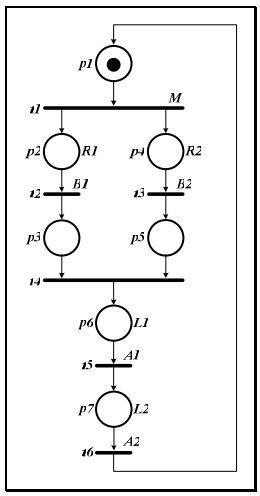
Zaawansowane systemy CAD, np. Active-HDL firmy Aldec, pozwalają na zintegrowanie rezultatów pracy z nowoczesną techniką projektowania systemów cyfrowych w środowisku różnych języków opisu sprzętu (HDL). Przedmiotem wielu prac było modelowanie układów sterowania w takich językach, jak: VHDL [6,7,12,13,15,19,20] i Verilog [8,11,15,16,17,18]. W dalszej części referatu modelowanie zostanie ograniczone do języka VHDL.

W celu lepszego zilustrowania sposobów modelowania sieci Petriego zostanie wykorzystany przykład układu sterowania dwoma wózkami (Rys.2) [15]. Sieć Petriego pokazano na rys. 3, a odpowiadającą jej specyfikację w formacie PNSF2 na rys. 4. Tabela 1 zawiera odpowiednie reguły opisujące działanie sterownika.



Rys. 2. Schemat procesu wraz z symbolem sterownika
Fig. 2. Schema of the process, and the controller symbol

Podstawowy model prezentowany w tej pracy wykorzystuje konstrukcję *process* do opisania kolejno wszystkich miejsc danej sieci Petriego. Uwzględniając fakt, że rozpatrywane są synchroniczne sieci Petreigo, dlatego procesy są czułe właśnie na sygnał synchronizujący, tzn. *Clk*. Ponadto, w celu zwiększenia możliwości testowania realizowanego układu wprowadzony został synchroniczny sygnał zerujący (*Reset*). Główną część procesu stanowi instrukcja warunkowa uwzględniająca zależność (3). W analizowanym przykładzie wyjścia typu Moore'a opisano w postaci grupy odpowiednich przypisań współbieżnych. Model pokazano na rys. 5. Ze względu na fakt, że specyfikacja bazuje na opisie poszczególnych miejsc nazwano ją jako *zorientowaną na miejsca z wieloma procesami*.



Rys. 3. Sieć Petriego
Fig. 3. Petri net

```
.clock CLK
.inputs M A1 B1 A2 B2
.comb_outputs L1 R1 L2 R2

.part Sterownik
.places p1 p2 p3 p4 p5 p6 p7
.transitions t1 t2 t3 t4 t5 t6

.net
 t1: p1 * M  |- p2;
 t2: p2 * B1 |- p3;
 t3: p4 * B2 |- p5;
 t4: p3 * p5 |- p6;
 t5: p6 * A1 |- p7;
 t6: p7 * A2 |- p1;

.MooreOutputs
 p2 |- R1;
 p4 |- R2;
 p6 |- L1;
 p7 |- L2;

.marking p1
.end
```

Rys. 4. Specyfikacja układu sterowania w PNSF2
Fig. 4. PNSF2 Specification of Logic Controller

Tab. 1. Tabela decyzyjna dla sterownika
Tab. 1. Decision table for control unit description

Tranzycja	*Aktualny stan lokalny*	*Warunek*	*Następny stan lokalny*
t1	p1	M	p2, p4
t2	p2	B(1)	p3
t3	p4	B(2)	p5
t4	p3, p5	-	p6
t5	p6	A(1)	p7
t6	p7	A(2)	p1

Uwaga: Symbol '-' oznacza przejście bezwarunkowe (logiczne '1').

W kolejnym modelu (Rys. 6), również *zorientowanym na miejsca*, uwzględniono fakt, że wszystkie procesy synchronizowane są jednym zegarem. Uproszcza to model, ponieważ realizacja wszystkich tranzycji specyfikowana jest w jednym procesie *MI*. Ponadto wyodrębniono także warunki realizacji poszczególnych tranzycji i zapisano je w dodatkowym procesie *TR*. Jednak prowadzi to do zatarcia prezentacji równoległości poszczególnych miejsc i tranzycji sieci. Nie ma to wpływu ani na symulację, ani syntezę modelu.

```
-- Metoda zorientowana na miejsca (oddzielne procesy dla każdego miejsca)
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity Sterownik is
    port (Clk, Reset: in std_logic;
          A1, A2, B1, B2, M: in std_logic;
          L1, L2, R1, R2: out std_logic);
end Sterownik;

architecture zorientowana_na_miejsca_proc of Sterownik is
signal p1, p2, p3, p4, p5, p6, p7 : std_logic;

begin
proc_p1: process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p1<= '1';
          elsif (p7='1' and A2='1') or (p1='1' and M= '0') then p1 <= '1';
                                                           else p1 <= '0';
          end if;
       end if;
end process;

proc_p2 : process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p2<= '0';
          elsif (p1='1' and M= '1') or (p2='1' and B1='0') then p2 <= '1';
                                                           else p2 <= '0';
          end if;
       end if;
end process;

proc_p3 : process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p3<= '0';
          elsif (p2='1' and B1='1') or (p3='1' and p5='0') then p3 <= '1';
                                                           else p3 <= '0';
          end if;
       end if;
end process;

proc_p4 : process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p4<= '0';
          elsif (p1='1' and M= '1') or (p4='1' and B2='0') then p4 <= '1';
                                                           else p4 <= '0';
          end if;
       end if;
end process;

proc_p5 : process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p5<= '0';
          elsif (p4='1' and B2='1') or (p5='1' and p3='0') then p5 <= '1';
                                                           else p5 <= '0';
          end if;
       end if;
end process;

proc_p6 : process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p6<= '0';
          elsif (p3='1' and p5='1') or (p6='1' and A1='0') then p6 <= '1';
                                                           else p6 <= '0';
          end if;
       end if;
end process;

proc_p7 : process (Clk)
begin  if rising_edge(Clk) then
          if Reset='1' then p7<= '0';
          elsif (p6='1' and A1='1') or (p7='1' and A2='0') then p7 <= '1';
                                                           else p7 <= '0';
          end if;
       end if;
end process;

--Wyjścia
L1 <= p6;
L2 <= p7;
R1 <= p2;
R2 <= p4;

end zorientowana_na_miejsca_proc;
```

Rys. 5. Model w VHDL – metoda zorientowana na miejsca z wieloma procesami
Fig. 5. VHDL model - place-oriented method with processes for all places

```
-- Metoda zorientowana na miejsca

architecture zorientowana_na_miejsca of Sterownik is
signal p1, p2, p3, p4, p5, p6, p7 : std_logic;
signal t1, t2, t3, t4, t5, t6 : std_logic;

begin
TR:process (p1, p2, p3, p4, p5, p6, p7, A1, A2, B1, B2, M)
begin
    if p1='1' and M= '1' then t1 <= '1'; else t1 <= '0'; end if;
    if p2='1' and B1='1' then t2 <= '1'; else t2 <= '0'; end if;
    if p4='1' and B2='1' then t3 <= '1'; else t3 <= '0'; end if;
    if p3='1' and p5='1' then t4 <= '1'; else t4 <= '0'; end if;
    if p6='1' and A1='1' then t5 <= '1'; else t5 <= '0'; end if;
    if p7='1' and A2='1' then t6 <= '1'; else t6 <= '0'; end if;
end process;

MI:process (Clk)
begin
    if rising_edge(Clk) then
       if Reset='1' then p1 <= '1'; p2 <= '0'; p3 <= '0'; p4 <= '0';
                         p5 <= '0'; p6 <= '0'; p7 <= '0' ;
                    else if t6='1' or (p1='1' and t1='0')
                            then p1<='1'; else p1<='0';  end if;
                         if t1='1' or (p2='1' and t2='0')
                            then p2<='1'; else p2<='0';  end if;
                         if t2='1' or (p3='1' and t4='0')
                            then p3<='1'; else p3<='0';  end if;
                         if t1='1' or (p4='1' and t3='0')
                            then p4<='1'; else p4<='0';  end if;
                         if t3='1' or (p5='1' and t4='0')
                            then p5<='1'; else p5<='0';  end if;
                         if t4='1' or (p6='1' and t5='0')
                            then p6<='1'; else p6<='0';  end if;
                         if t5='1' or (p7='1' and t6='0')
                            then p7<='1'; else p7<='0';  end if;
       end if;
    end if;
end process;

--Wyjścia jak w poprzednim modelu

end zorientowana_na_miejsca;
```

Rys. 6. Model w VHDL – metoda zorientowana na miejsca, z procesem dla miejsc
Fig. 6. VHDL model - place-oriented method with process for places

Inne podejście (Rys. 7) jest prezentowane, gdy opis będzie *zorientowany na tranzycje*. W pierwszym kroku stan wszystkich miejsc ustawiany jest na 0, a następnie wychodząc kolejno od wszystkich miejsc analizowane są ich tranzycie wyjściowe i na tej podstawie ustawiane jest nowe znakowanie całej sieci.

Kolejny model (Rys. 8) łączy w jednym procesie ustawienie i utrzymanie stanu poszczególnych miejsc.

```
-- Metoda zorientowana na tranzycje
architecture zorientowana_na_tranzycje of Sterownik is
signal p1, p2, p3, p4, p5, p6, p7 : std_logic;

begin
OT:process (Clk)
begin
    if rising_edge(Clk) then
       if Reset='1' then p1 <= '1'; p2 <= '0'; p3 <= '0';
                         p4 <= '0'; p5 <= '0'; p6 <= '0'; p7 <= '0';
                    else p1 <= '0'; p2 <= '0'; p3 <= '0';
                         p4 <= '0'; p5 <= '0'; p6 <= '0'; p7 <= '0';
                         if p7='1' then if A2='1' then p1 <= '1';
                                                  else p7 <= '1'; end if;
                         end if;
                         if p1='1' then if M= '1' then p2 <= '1'; p4 <= '1';
                                                  else p1 <= '1'; end if;
                         end if;
                         if p2='1' then if B1='1' then p3 <= '1';
                                                  else p2 <= '1'; end if;
                         end if;
                         if p4='1' then if B2='1' then p5 <= '1';
                                                  else p4 <= '1'; end if;
                         end if;
                         if p3='1' and p5='1' then p6 <= '1';
                                              else if p3='1' then p3 <= '1';
                                                   end if;
                                                   if p5='1' then p5 <= '1';
                                                   end if;
                         end if;
                         if p6='1' then if A1='1' then p7 <= '1';
                                                  else p6 <= '1'; end if;
                         end if;
       end if;
    end if;
end process;
--Wyjścia jak w poprzednim modelu
end zorientowana_na_tranzycje;
```

Rys. 7. Model w VHDL – metoda zorientowana na tranzycje
Fig. 7. VHDL model - transition-oriented method

```
-- Metoda zorientowana na miejsca i tranzycje
architecture zorientowana_na_miejsca_i_tranzycje of Sterownik is
signal p1, p2, p3, p4, p5, p6, p7 : std_logic;

begin
process (Clk)
begin
    if rising_edge(Clk) then
       if Reset='1' then p1 <= '1'; p2 <= '0'; p3 <= '0'; p4 <= '0';
                         p5 <= '0'; p6 <= '0'; p7 <= '0';
                    else p1 <= '0'; p2 <= '0'; p3 <= '0'; p4 <= '0';
                         p5 <= '0'; p6 <= '0'; p7 <= '0';
                         --realizacja tranzycji, czyli ustaw miejsca
                         if p1='1' and M= '1' then  p2<='1'; p4<='1'; end if;
                         if p2='1' and B1='1' then  p3<='1'; end if;
                         if p4='1' and B2='1' then  p5<='1'; end if;
                         if p3='1' and p5='1' then  p6<='1'; end if;
                         if p6='1' and A1='1' then  p7<='1'; end if;
                         if p7='1' and A2='1' then  p1<='1'; end if;
                         --utrzymanie miejsca
                         if p1='1'  and M= '0' then p1 <='1'; end if;
                         if p2='1'  and B1='0' then p2 <='1'; end if;
                         if p4='1'  and B2='0' then p4 <='1'; end if;
                         if p3='1'  and p5='0' then p3 <='1'; end if;
                         if p5='1'  and p3='0' then p5 <='1'; end if;
                         if p6='1'  and A1='0' then p6 <='1'; end if;
                         if p7='1'  and A2='0' then p7 <='1'; end if;
       end if;
    end if;
end process;
--Wyjścia jak w poprzednim modelu
end zorientowana_na_miejsca_i_tranzycje;
```

Rys. 8. Model w VHDL – metoda zorientowana na miejsca i tranzycje
Fig. 8. VHDL model – transition-and-place-oriented method

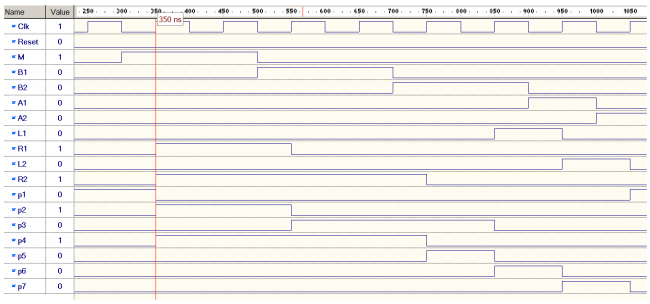
```
-- Metoda zorientowana na miejsca, wersja z 'alias'
entity Sterownik is
    generic (Ile_Miejsc: integer := 7;
             Ile_Tranzycji: integer := 6;
             Ile_Wejsc : integer := 5;
             Ile_Wyjsc : integer := 4);
    port (Clk, Reset: in std_logic;
          Wejscia: in std_logic_vector(Ile_Wejsc downto 1);
          Wyjscia: out std_logic_vector(Ile_Wyjsc downto 1));
end Sterownik;

architecture zorientowana_na_miejsca_alias of Sterownik is
signal p : std_logic_vector(Ile_Miejsc downto 1);
signal t : std_logic_vector(Ile_tranzycji downto 1);

alias p1: std_logic is p(1);
alias p2: std_logic is p(2);
...

alias t1: std_logic is t(1);
alias t2: std_logic is t(2);
...
alias M:  std_logic is Wejscia(1);
alias A1: std_logic is Wejscia(2);
...
alias L1: std_logic is Wyjscia(1);
alias L2: std_logic is Wyjscia(2);
...
begin
TR:process (p, Wejscia)
... -- dalsza czesc, jak w modelu zorientowanym na miejsca (Rys. 6)

end zorientowana_na_miejsca_i_tranzycje;
```

Rys. 9. Model w VHDL – metoda zorientowana na miejsca, wersja z 'alias'
Fig. 9. VHDL model –place-oriented method, version with 'alias'

Ostatni prezentowany model (Rys. 9) jest modyfikacja modelu z rys. 6. Zmiany mają na celu ujednolicenie interfejsu (*entity*). Wszystkie *wejścia* i *wyjścia* sterownika, jak również tranzycie *t* oraz miejsca *p* zgrupowano w odpowiednie wektory. Ponadto wykorzystano konstrukcję *alias* i dzięki temu właściwa część modelu jest zgodna z wcześniej opisaną.

4. Symulacja i synteza

Przedstawione w rozdz. 3 modele mogą być wykorzystane zarówno do symulacji, jak i syntezy projektowanego układu sterowania. Wyniki symulacji wszystkich zaprezentowanych modeli są sobie równoważne. Przykładowy wykres prezentujący wyniki symulacji przedstawiono na rys. 10.



Rys. 10. Wyniki symulacja dla modelu (Rys.5)
Fig. 10. Simulation results for the model from Fig. 5

Większe różnice w wynikach mają miejsce przy syntezie. Tutaj wpływ sposobu modelowania ma już znaczenie. Ponadto, wyniki syntezy mierzone w postaci wykorzystanych zasobów układu FPGA są uzależnione od stosowanych narzędzi syntezy. Obrazuje to dodatkowo, jakie metody warto stosować, jako bardziej uniwersalne, czyli niezależne od konkretnego systemu CAD.

Tabela 1. Wyniki syntezy (Xilinx FPGA – XC3S250epq208-5)
Table 1. Synthesis summary (Xilinx FPGA – XC3S250epq208-5)

Model	*Liczba bloków (slice)*	*Liczba tablic LUT4*	*Liczba przerzutników (FF)*
VHDL (Rys. 5)	*4 (14)*	*7 (13)*	*7*
VHDL (Rys. 6)	*4 (14)*	*7 (13)*	*7*
VHDL (Rys. 7)	*4*	*7*	*7*
VHDL (Rys. 8)	*4 (11)*	*7 (8)*	*7*
VHDL (Rys. 9)	*4 (14)*	*7 (13)*	*7*

Uwaga: wyniki podano dla sytemu MG Precision 2008a, a w nawiasach dla Xilinx XST (ISE 10.1.3)

5. Wnioski

Do modelowania współbieżnych układów sterowania wykorzystywane są sieci Petriego. Jednak takie modele nie są popularne wśród inżynierów, którzy preferują języki opisu sprzętu. Modele w językach HDL wykorzystywane są zarówno do symulacji, jak i syntezy. Logika programowalna, np. układy FPGA, stanowią elastyczną i efektywną platformę do implementacji układowej projektowanych systemów. W referacie przedstawiono przegląd wybranych metod modelowania układów opisanych sieciami Petriego w języku VHDL. Wybór odpowiedniego sposobu ma wpływ zarówno na przejrzystość modelu, jak również na efektywność syntezy. W zaprezentowanym przeglądzie metod ograniczono się do takich, w których sposób kodowania miejsc wybierany jest jako opcja w programie do syntezy i najczęściej automatycznie sprowadza się do metody typu *one-hot*. Inne metody kodowania wykraczają poza przyjęty zakres. Z tego też względu nie uwzględniono również podejścia hierarchicznego.

Automatyczne generowanie modeli zgodnie z zaprezentowanymi metodami stanowi moduł akademickiego systemu wspomagającego projektowanie współbieżnych sterowników logicznych PeNLogic rozwijanego w Uniwersytecie Zielonogórskim.

6. Literatura

[1] M.Adamski: Parallel Controller Implementation using Standard PLD Software. w: W.R.Moore, W.Luk (Eds), FPGAs, The Oxford 1991 International Workshop on Field Programmable Logic and Applications, Abingdon EE&CS, Abingdon (UK), 1991, pp.296-304.
[2] Z.Banaszak, J.Kuś, M.Adamski: Sieci Petriego. Modelowanie, Sterowanie i Synteza Systemów Dyskretnych. Wydawnictwo Wyższej Szkoły Inżynierskiej, Zielona Góra, 1993.
[3] S.Baranov: Logic Synthesis for Control Automata. Kluwer Academic Publishers, Boston, 1994.
[4] H.Belhadj, L.Gerbaux, M.-C.Bertrand, G.Saucier: Specification and Synthesis of Communicating Finite State Machines. w: G.Saucier, J.Trilhe, (Red.), Synthesis for Control Dominated Circuits, Elsevier Science Publishers B.V. , IFIP, North-Holland, 1993, ss.91-102.
[5] R.David, H.Alla: Petri Nets & Grafcet. Tools for modelling discrete event systems. Prentice Hall, New York, 1992.
[6] L.Gomes, A.Costa, J.P.Barros, P.Lima: From Petri net models to VHDL implementation of digital controllers. The 33rd Annual Conference of the IEEE Industrial Electronics Society, IECON 2007, 2007, Taipei, Taiwan.
[7] J.M.Fernandes, M.Adamski, A.J.Proença: VHDL generation from hierarchical Petri net specifications of parallel controllers. IEE Proc., Part E - Computers and Digital Techniques, Vol.144, No.2, March 1997, ss.127-137.
[8] Z.Hajduk: Sprzętowa implementacja rozmytych sieci Petriego jako układów sterowania. Rozprawa doktorska. Uniwersytet Zielonogórski, Wydział Elektrotechniki, Informatyki i Telekomunikacji, Zielona Góra, 2006.
[9] T.Kozłowski, E.L.Dagless, J.M.Saul, M.Adamski, J.Szajna: Parallel controller synthesis using Petri nets. IEE Proceedings, Part E: Computers and Digital Techniques, 1995, Vol.142, No.4, ss.263-271.
[10] T.Murata: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol.77, No.4, April 1989, ss.541-580.
[11] P.Minns, I.Elliott: FSM based Digital Design using Verilog HDL. John Wiley & Sons, Ltd., Chichester, England, 2008.
[12] J.Pardey, M.Bolton: Parallel controller synthesis for concurrent data paths. Proc. of the IFIP Workshop Control Dominated Synthesis From a Register Transfer Level Description, 1992, ss.16-19.
[13] E.Soto, M,Pereira: Implementing a Petri net specification in a FPGA using VHDL. Proceedings of the International Workshop on Discrete-Event System Design, DESDes 2001.
[14] A.Węgrzyn: Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci Petriego. Oficyna Wydaw. Uniwersytetu Zielonogórskiego, Zielona Góra, 2003.
[15] M.Węgrzyn: Hierarchiczna implementacja współbieżnych kontrolerów cyfrowych z wykorzystaniem FPGA. Rozprawa doktorska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, Warszawa, 1998.
[16] M.Węgrzyn: Modelling and synthesis of safety-critical systems by means of Petri nets and FPGAs. Radioelektronika i Informatyka, (ISSN 1563-0064), Nr 4, 2001, ss.115-118.
[17] M.Węgrzyn: Implementation of Safety Critical Logic Controller by Means of FPGA, Annual Reviews in Control, Vol.27, 2003, ss.55-61.
[18] M.Wegrzyn: Petri Net Decomposition Approach for Partial Reconfiguration of Logic Controllers. The 3rd IFAC Workshop on Discrete-Event System Design, DESDes'06, Rydzyna (Polska), 26-28.09.2006, ss. 323-328.
[19] P.Wolański, M.Węgrzyn, M.Adamski: VHDL Modelling of industrial Logic Control System. Proceedings of the 42nd International Scientific Colloquium, IWK'97, Ilmenau, Niemcy, 22-25.09.1997, Band I, ss.528-533.
[20] P.Wolański: Modelowanie układów cyfrowych na poziomie RTL z wykorzystaniem sieci Petriego i podzbioru języka VHDL. Rozprawa doktorska. Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych, Warszawa, 1998.

Artykuł recenzowany