CZASOPISMO TECHNICZNE
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ

z. 1-I/2008
ISSN 0011-4561
ISSN 1897-631X

GRZEGORZ BAZYDŁO, MARIAN ADAMSKI*

PROJEKTOWANIE STEROWNIKÓW LOGICZNYCH OPISANYCH DIAGRAMAMI MASZYNY STANOWEJ UML

LOGIC CONTROLLERS DESIGN FROM UML STATE MACHINE DIAGRAMS

Streszczenie

W artykule przedstawiono nową metodę projektowania sterowników logicznych realizowanych w sposób układowy w strukturach FPGA z wykorzystaniem języka Verilog i programów profesjonalnych do symulacji i syntezy logicznej. Modelem behawioralnym programu sterownika jest diagram maszyny stanowej UML 2.1.2. Formalnym modelem strukturalnym jest hierarchiczna sieć współpracujących ze sobą automatów cyfrowych.

Słowa kluczowe: *diagram maszyny stanowej*, *UML*, *sterownik logiczny*, *Verilog*, *FPGA*

Abstract

The paper presents a new design method for logic controllers, which are implemented as digital circuit in Field Programmable Gate Arrays (FPGA) by means of hardware description language Verilog and professional tools for simulation and logic synthesis. The UML 2.1.2 state machine diagram is used as an initial behavioral model. The formal structured design model is based on hierarchical network of collaborated Finite State Machines.

Keywords: *state machine diagram*, *UML*, *logic controller*, *Verilog*, *FPGA*

* Mgr inż. Grzegorz Bazydło, prof. dr hab. inż. Marian Adamski, Instytut Informatyki i Elektroniki, Wydział Elektrotechniki, Informatyki i Telekomunikacji, Uniwersytet Zielonogórski.

1. Wstęp

Metody projektowania układów cyfrowych są szeroko rozwijane od wielu lat. Postęp technologiczny wpłynął zasadniczo na zmianę podejścia do modelowania układów sterowania cyfrowego. Jeszcze do końca lat 80. stosowane były głównie elementy dużej skali integracji w połączeniu z elementami małej i średniej skali integracji. Obecnie dominują specjalizowane układy scalone (cyfrowe oraz analogowe) bardzo dużej skali integracji (ang. *Very Large Scale Integration* – VLSI) projektowane dla konkretnych zastosowań. Pojawienie się nowej technologii układów specjalizowanych pociągnęło za sobą rozwój nowych metod projektowych i systemów CAD (ang. *Computer-Aided Design*). Z drugiej strony, powstawanie nowych i tańszych technologii stale powiększa obszar zastosowań układów elektronicznych. Powstaje coraz więcej systemów, zwłaszcza osadzonych (ang. *embedded system*), które towarzyszą człowiekowi w jego codziennym życiu [23].

Opracowano wiele metod projektowych, jednak w wielu przypadkach, dotyczących głównie złożonego zachowania, tradycyjne metody projektowania okazują się niewystarczające. Brak pojęć abstrakcyjnych w modelowaniu zmusza projektanta do operowania dużą liczbą szczegółów, co sprawia, że specyfikacja jest mniej czytelna, a samo projektowanie trudniejsze. Ponadto niektóre z metod nie odpowiadają już dzisiejszym potrzebom w zakresie wykorzystania dostępnych zasobów sprzętowych [22]. Dostawcy technologii oferują bowiem projektantom układy programowalne zawierające nawet 10 mln bramek. Wobec tego coraz większego znaczenia nabiera modelowanie bardziej abstrakcyjne (na poziomie systemu). Wadą takiego podejścia może być większe zużycie zasobów podstawowych elementów elektronicznych, co jednak w sytuacji stałego i równomiernego wzrostu gęstości upakowania tranzystorów w układzie scalonym wydaje się być do zaakceptowania, zwłaszcza że opisywanemu wzrostowi towarzyszy zmniejszenie kosztów elementów jednostkowych. Zaletą modelowania bardziej abstrakcyjnego jest zwiększenie możliwości modelowania funkcjonowania (modelowanie behawioralne), a projektowane urządzenia mogą charakteryzować się większymi i szerszymi możliwościami zastosowań. Pamiętać jednak należy, że takie podejście, jako pierwszy etap projektowania, ze względu na swój abstrakcyjny charakter ma pewne ograniczenia, które rzutują na wszystkie pozostałe etapy i na ostateczną funkcjonalność projektowanego układu [21].

Istnieje wiele metod pozwalających w sposób graficzny modelować zachowanie sterownika (np. diagramy blokowe, sieci SFC [4], FSM, sieci Petriego [1], diagramy stanów [15]), inne oferują tekstowy opis specyfikacji systemu (np. języki opisu sprzętu). Graficzne metody mają tę przewagę, że są bardziej intuicyjne i łatwiejsze do zrozumienia niż metody specyfikacji tekstowej. Te drugie z kolei lepiej nadają się do dalszego przetwarzania. Format tekstowy jest też często jedyną akceptowalną specyfikacją dla zaawansowanych systemów do symulacji, syntezy i implementacji sterowników logicznych. Niestety, brakuje uniwersalnej metody, która – z jednej strony – pozwalałaby na opis złożonych hierarchicznych systemów współbieżnych i bazowałaby na graficznych elementach, z drugiej zaś – dobrze nadawałaby się do dalszego przetwarzania specyfikacji.

Zagadnienia syntezy diagramów UML (ang. *Unified Modelling Language*) w postaci struktur cyfrowych są w literaturze rzadko podejmowane. Z ustaleń autorów wynika, że najistotniejszymi publikacjami dotyczącymi sprzętowej implementacji diagramów są prace [8, 10–12, 20, 21, 28] oraz materiały firmowe firmy I-Logix związane z pakietem Statemate Magnum [18]. W niniejszym artykule uwaga koncentrować się będzie na modelowaniu

zachowania z wykorzystaniem języka UML [26]. Język ten stanowi obecnie jeden z najważniejszych standardów inżynierii oprogramowania i może on także z powodzeniem służyć do graficznej specyfikacji programów dla sterowników logicznych [5, 6]. Szczególnie przydatne do tego celu są diagramy maszyny stanowej, ponieważ silnie odwołują się do wizualizacji oraz jawnego wykorzystywania hierarchii behawioralnej. W artykule zaprezentowano koncepcję syntezy behawioralnej [30] modeli opisanych diagramami maszyny stanowej UML. Wynikiem jest modularny opis modelowanego systemu w języku opisu sprzętu Verilog. Taka specyfikacja tekstowa może być następnie poddana symulacji i syntezie w zewnętrznych środowiskach, np. w systemach *Active HDL* firmy Aldec czy *Xilinx ISE* firmy Xilinx. W celu praktycznej weryfikacji metody zaprojektowano i zrealizowano system CAD (*UML-XML2Verilog*), za pomocą którego możliwa jest automatyczna translacja modeli zapisanych w języku UML (specyfikacja tekstowa XML – ang. *Extensible Markup Language*) do języka opisu sprzętu Verilog.

2. Sterownik logiczny

Sterownik logiczny można potraktować jako przykład systemu reaktywnego [16]. W systemach tych przyjmuje się, że dane wejściowe mogą pojawiać się w dowolnym momencie, co więcej, oczekuje się natychmiastowej reakcji systemu na te zdarzenia. Ponadto systemy reaktywne mogą być sterowane zdarzeniami, prowadzą stałą interakcję z otoczeniem (z użyciem sygnałów i przerwań), zmieniają swój stan w zależności od bieżącego i przeszłego zachowania oraz są to często systemy współbieżne [21]. Zaproponowana w niniejszym artykule metoda projektowania jest szczególnie przydatna dla implementacji układowych w rekonfigurowalnych sterownikach logicznych (RLC – ang. *Reconfigurable Logic Controller*) z wbudowaną matrycą makrokomórek CLB (ang. *Configurable Logic Block*), np. układach FPGA (ang. *Field Programmable Gate Array*). Niemniej jednak istnieje możliwość zastosowania proponowanej koncepcji także do symulacji i weryfikacji programów dla sterowników typu PLC (ang. *Programmable Logic Controller*). W tym celu pośredni model zostanie potraktowany jako modularna struktura współpracujących ze sobą bloków funkcyjnych FBD (ang. *Function Block Diagram*) i przedstawiony zgodnie ze standardem IEC 1131-3 [3].

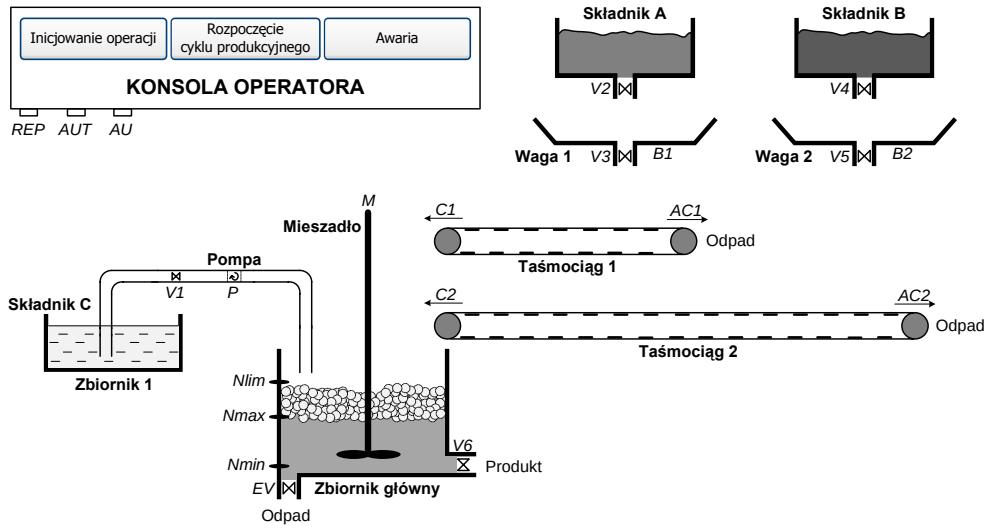
2.1. Przykład specyfikacji behawioralnej

Praktyczne zastosowanie metody zaprezentowano na przykładzie Mieszalnika przemysłowego wzorowanym na opisie reaktora zamieszczonym w książce [21]. Schemat procesu technologicznego Mieszalnika przedstawiono na rys. 1.

Działanie systemu Mieszalnika można podzielić na kilka etapów:

Etap I – inicjowanie pracy

Po wystąpieniu sygnału *REP* (naciśnięcie przycisku na konsoli operatora) następuje opróżnianie zbiornika głównego (usunięcie odpadu) aż do poziomu *Nmin* poprzez otwarcie zaworu *EV*. W tym czasie taśmociągi 1 i 2 wprawiane są w ruch do tyłu (*AC*1, *AC*2), aby usunąć pozostałości z poprzedniego cyklu technologicznego. Taśmociągi 1 i 2 pracują przez czas *t*1, którego upłynięcie sygnalizowane jest wystąpieniem sygnału *FT*1.



Rys. 1. Schemat przykładowego procesu technologicznego

Fig. 1. Case study of technological discrete process

Etap IIa – normalny cykl pracy (Napełnianie)

Jeżeli poziom cieczy w zbiorniku głównym jest mniejszy od poziomu minimalnego *Nmin*, to przy aktywnym sygnale *AUT* (naciśnięcie przycisku na konsoli operatora) otwierają się zawory *V*1, *V*2 i *V*4 oraz wprawiana jest w ruch pompa *P*. Jeśli w czasie napełniania zbiornika głównego poziom piany podniesie się powyżej poziomu *Nlim*, zawór *V*1 jest zamykany oraz pompa *P* jest zatrzymywana. Po opadnięciu piany poniżej poziomu *Nlim* następuje ponowne otwarcie zaworu *V*1 i uruchomienie pompy *P* aż do wypełnienia zbiornika cieczą do poziomu *Nmax*. W tym samym czasie na wagi 1 i 2 nasypywane są składniki A i B (otwarcie zaworów *V*2 i *V*4). Wystąpienie sygnałów *B*1 oraz *B*2 wskazuje, że na wagach znajduje się odpowiednia ilość składników A i B, co oznacza zamknięcie zaworów *V*2 i *V*4.

Etap IIb – normalny cykl pracy (Proces)

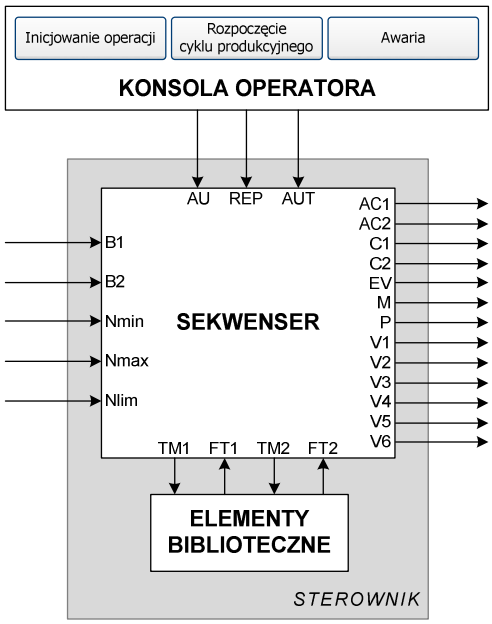
Jeżeli w zbiorniku głównym znajduje się odpowiednia ilość cieczy (poziom *Nmax*) oraz na wagach znajduje się wymierzona ilość składników A i B (czujniki *B*1 i *B*2), następuje otwarcie zaworów *V*3 i *V*5 oraz uruchomienie taśmociągów 1 i 2 (sygnały *C*1 i *C*2). Składniki dodawane są do cieczy w zbiorniku głównym i uruchamiane jest mieszadło (sygnał *M*). Po upływie czasu *t*1 (sygnalizowane wystąpieniem sygnału *FT*1) zamykane są zawory *V*3 i *V*5 oraz zatrzymywane są taśmociągi 1 i 2. Następuje otwarcie zaworu zbiornika głównego *V*6 i odbiór produktu. Po upływie czasu *t*2 (sygnał *FT*2) zatrzymywane jest mieszadło. Jeśli poziom cieczy (produktu) w zbiorniku obniży się poniżej poziomu *Nmin*, następuje zamknięcie zaworu *V*6. Jeśli aktywny jest sygnał *AUT* (konsola operatora), następuje ponowne wykonanie normalnego cyklu pracy systemu (etap II).

Etap III – awaryjne zatrzymanie systemu

Ze względów bezpieczeństwa przewiduje się możliwość nagłego zatrzymania pracy całego systemu w przypadku wystąpienia awarii – sygnał *AU* (konsola operatora). Jeżeli sygnał *AU* pojawi się w czasie etapu IIa (Proces), produkt reakcji jest niezdatny do wykorzystania. Następuje wtedy zatrzymanie pracy całego układu, a następnie po usunięciu awarii zbiornik jest opróżniany i usuwane są składniki z taśmociągów (etap I – Inicjowanie pracy). Jeżeli natomiast sygnał awarii *AU* wystąpi podczas etapu IIb (Napełnianie), to następuje zatrzymanie pracy całego układu, a po usunięciu awarii (brak sygnału *AU* oraz aktywny sygnał *REP*) system wznawia pracę – etap IIb (Napełnianie).

2.2. Struktura blokowa sterownika

Schemat blokowy sterownika logicznego na przykładzie Mieszalnika przedstawiono na rys. 2. Konsola operatora zawiera trzy przyciski, których naciśnięcie bądź przełączenie powoduje wygenerowanie sygnałów *AU*, *REP* lub *AUT*. Za sterowanie całym systemem odpowiedzialny jest sekwenser, który na podstawie sygnałów wejściowych z czujników i bieżącego stanu układu generuje odpowiednie sygnały wyjściowe, sterujące poszczególnymi urządzeniami w systemie (otwieranie lub zamykanie zaworów, wprawianie w ruch taśmociągu, uruchamianie pompy i mieszadła). Blok sekwensera korzysta także z dodatkowych elementów bibliotecznych, jak np. zegar (timer), które generują sygnały oznaczające upływ zadanego czasu. Z punktu widzenia teorii automatów cyfrowych sekwenser pełni rolę automatu sterującego, natomiast elementy biblioteczne (rdzenie projektowe, IPcores) są potraktowane jako składowe automatu operacyjnego.



Rys. 2. Część sterująca i operacyjna sterownika (Mieszalnik)

Fig. 2. Control and data part of logic controller (Industrial Mixer)

3. Modelowanie sterownika logicznego w języku UML

Pierwszym krokiem proponowanej metody jest zamodelowanie w języku UML zachowania projektowanego sterownika logicznego. Język UML to język modelowania wizualnego służący do obrazowania, specyfikowania, tworzenia i dokumentowania szeroko pojętych systemów informatycznych [7, 13]. Autorami tego języka są trzej znani metodolodzy: Grady Booch, Ivar Jacobson, James Rumbaugh. Pierwsza wersja języka UML została opublikowana w 1995 r. (wersja 0.8), obecnie UML ma numer 2.1.2 [25, 26]. Podstawowym celem UML jest modelowanie różnego rodzaju systemów z wykorzystaniem pojęć obiektowych. Język UML [21]:

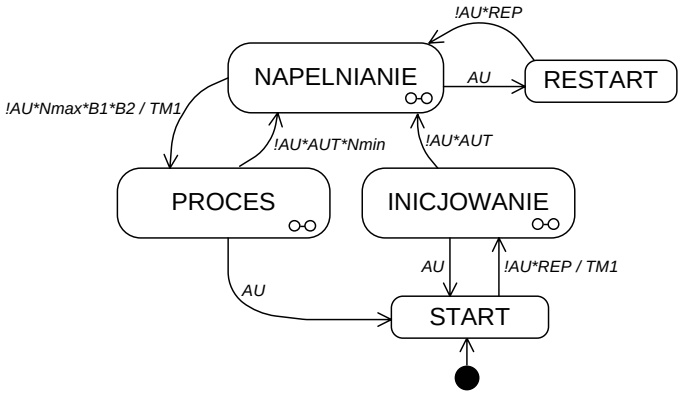
– obejmuje specyfikację, konstrukcję, wizualizację i dokumentację projektowanego systemu,
– łączy w sobie różne istniejące metodyki w jedną zunifikowaną,
– może stanowić wsparcie dla systemów współbieżnych, hierarchicznych i rozproszonych,
– pozwala spojrzeć na system z punktu widzenia przyszłego użytkownika (funkcje systemu).

UML służy nie tylko do projektowania systemów informatycznych. Język ten jest na tyle wyrazisty i uniwersalny, że można za jego pomocą modelować systemy niezwiązane bezpośrednio z oprogramowaniem [5, 19]. Podstawowym środkiem oferowanym przez język UML są diagramy [26], które można traktować jako swego rodzaju rzut systemu. W obecnej wersji języka wyróżnia się trzynaście rodzajów diagramów, które można podzielić na dwie grupy: struktury oraz dynamiki. Do grupy diagramów o nazwie struktury należą diagramy: klas, obiektów, pakietów, struktur połączonych, komponentów oraz rozlokowania. Pozostałych siedem diagramów to diagramy dynamiki: diagramy przypadków użycia, czynności, maszyny stanowej, sekwencji, komunikacji, harmonogramowania i sterowania interakcją. Najciekawsze z punktu widzenia wykorzystania do specyfikacji behawioralnej sterowników logicznych wydają się być diagramy maszyny stanowej, ponieważ wprost odwołują się do pojęcia automatu skończonego [15].

Diagramy maszyny stanowej to graficzne odzwierciedlenie dyskretnego, skokowego zachowania skończonych systemów stan–przejście [29]. Stany na diagramie reprezentowane są w postaci prostokątów z zaokrąglonymi rogami (tzw. krągłokątów [17]). Przejścia pomiędzy stanami oznaczane są strzałkami. Na rysunku 3 przedstawiono diagram maszyny stanowej dla przykładu Mieszalnika. Na prezentowanym diagramie z każdym przejściem związane jest zdarzenie uruchamiające to przejście, oznaczające wystąpienie danego sygnału (np. *AU*) lub kombinacji sygnałów (np. *!AU*Nmax*B*1**B*2). Wykrzyknik przed nazwą sygnału oznacza jego negację, a symbol gwiazdki (*) oznacza iloczyn logiczny sygnałów, czyli jednoczesne wystąpienie danej kombinacji sygnałów. Z przejściem może być także skojarzona pewna akcja, np. */AU*REP*/*TM*1, gdzie *TM*1 oznacza uaktywnienie sygnału *TM*1 uruchamiającego zegar (timer).

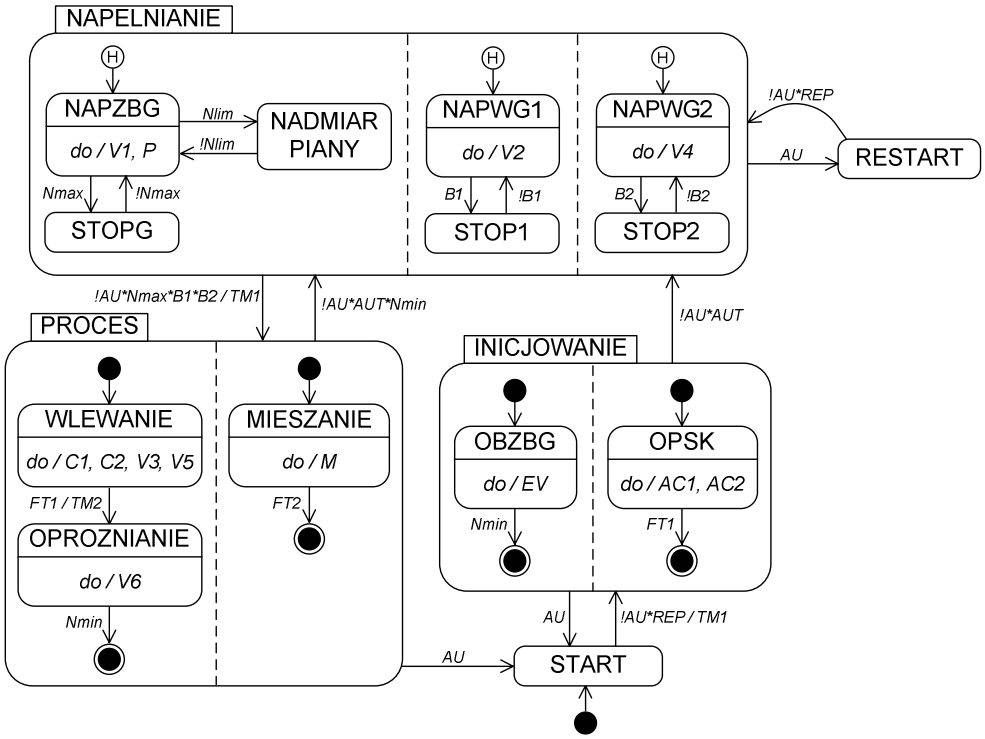
Warto zwrócić uwagę na wspieranie przez diagramy maszyny stanowej takich cech układu, jak hierarchiczność i współbieżność. Pozwala to w sposób intuicyjny i czytelny specyfikować zachowanie złożonych systemów współbieżnych na wybranym poziomie uszczegółowienia [6]. Jeśli nie ma potrzeby prezentacji wszystkich szczegółów modelowanego systemu, można przyjąć wyższy poziom hierarchii i ukryć zbędne (na danym etapie projektowania) informacje. W prezentowanym przykładzie stany *NAPELNIANIE*, *PROCES* i *INICJOWANIE* to w rzeczywistości stany złożone (oznaczone symbolem dwóch małych,

połączonych okręgów w prawym dolnym rogu stanu). Stany złożone mogą zawierać podmaszyny stanowe oraz obszary współbieżne.



Rys. 3. Diagram maszyny stanowej (Mieszalnik) – najwyższy poziom hierarchii

Fig. 3. State machine diagram (Industrial Mixer) – the highest level of hierarchy



Rys. 4. Diagram maszyny stanowej (Mieszalnik) – najniższy poziom hierarchii

Fig. 4. State machine diagram (Industrial Mixer) – the lowest level of hierarchy

Na rysunku 4 zaprezentowano diagram maszyny stanowej dla opisywanego przykładu na najniższym poziomie hierarchii. Występujący na rysunku symbol pseudostanu historii płytkiej (litera *H* w okręgu) oznacza, że po uaktywnieniu stanu *NAPELNIANIE* pierwszym aktywnym stanem będzie ten, który był aktywny ostatnio w momencie przekazania sterowania. Jeżeli stan *NAPELNIANIE* staje się aktywny po raz pierwszy, sterowanie zostanie przekazane do tego stanu, na który wskazuje pseudostan historii.

Na rynku dostępnych jest wiele narzędzi do modelowania w języku UML. Są wśród nich profesjonalne, komercyjne środowiska (np. *Enterprise Architect* firmy Sparx Systems), jak i darmowe programy (np. *Jude Community* firmy ChangeVision czy *ArgoUML* – projekt typu *open source*). Zaletą projektowanej metody jest możliwość wykorzystania praktycznie dowolnego oprogramowania UML, także darmowego, co pozwala zmniejszyć koszty projektowania. Ważne tylko, aby wybrane narzędzie dysponowało możliwością generowania specyfikacji w języku XML (zgodnie z formatem XMI [27]), gdyż właśnie ten format wybrano jako wejściowy do zrealizowanej aplikacji *UML-XML2Verilog*.

4. Synteza modeli opisanych diagramami UML

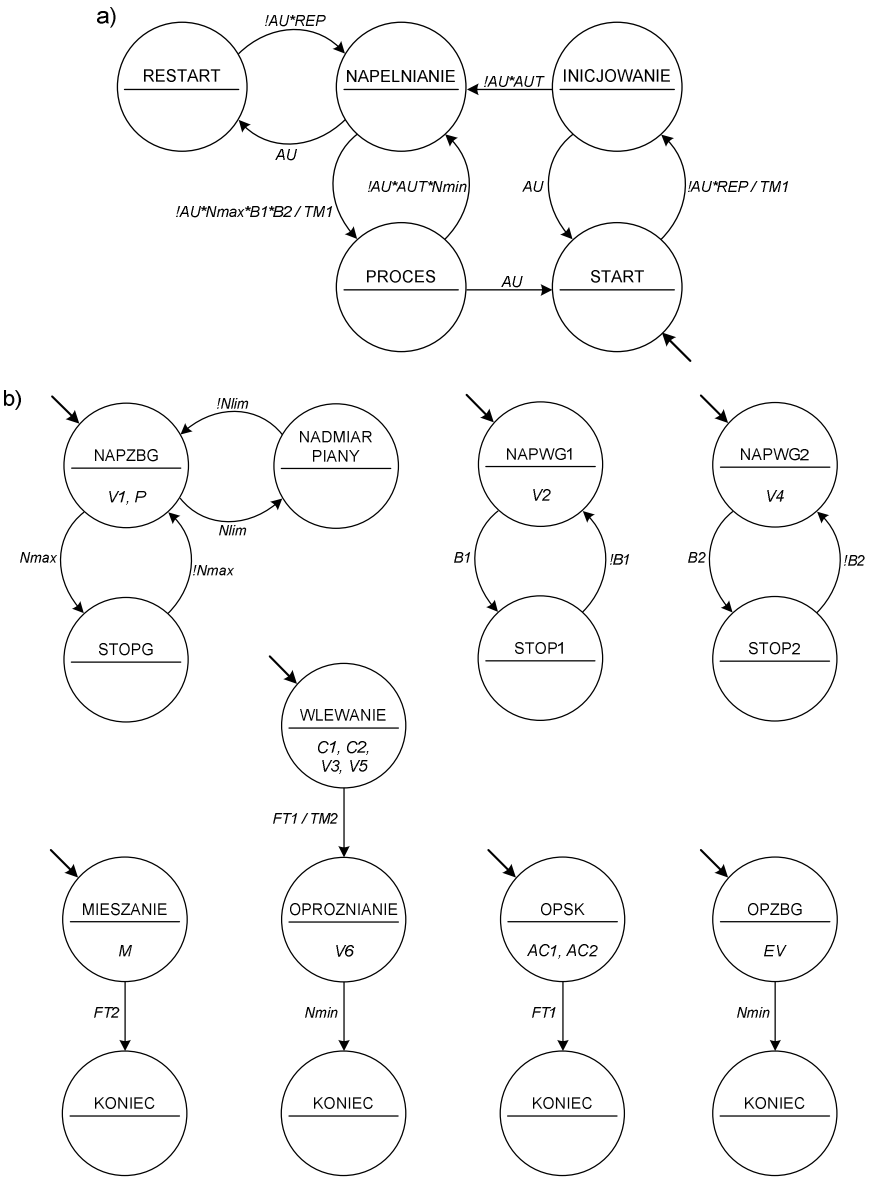
Kiedy sterownik logiczny zostanie zamodelowany za pomocą diagramów UML, kolejnym krokiem jest translacja jego graficznej reprezentacji na specyfikację akceptowalną przez współczesne systemy przemysłowe, jaką jest np. język opisu sprzętu [24]. Jako specyfikację docelową wybrano język Verilog. Wynikowa specyfikacja oparta jest na syntezowalnym podzbiorze tego języka, tzn. takim, który pozwala na przeprowadzenie procesu syntezy i implementacji modelowanego układu. Opracowana metoda syntezy bazuje na metodach opisanych w pracach [1] oraz [10], których najważniejszym elementem jest takie przekształcenie modelu, aby otrzymać hierarchiczną strukturę połączonych modułów, z których każdy można rozpatrywać jako oddzielny automat FSM (ang. *Finite State Machine*). W porównaniu z [10] główną modyfikacją jest zawężenie modelowania do diagramów modularnych, tzn. takich, w których brak jest przejść pomiędzy stanami na różnym poziomie hierarchii. Przekształcenia diagramów niemodularnych są przedmiotem dalszych prac autorów.

Podstawowym założeniem dynamiki układu jest realizacja sterownika jako układu synchronicznego (synchronizacja sygnałem zegarowym). Układ funkcjonuje w pewnym otoczeniu, które wytwarza zdarzenia pobudzające ten układ. Zakłada się, że wszystkie zdarzenia związane z układem (wejściowe, wyjściowe, lokalne) są powiązane z dyskretną dziedziną czasu. Reakcją na dostępne zdarzenia jest realizacja poszczególnych przejść, przy czym przyjmuje się, że układ nie będzie czuły na kolejne zdarzenie, dopóki nie wypracuje odpowiedzi na zdarzenie bieżące. Ważnym założeniem jest także to, że wygenerowane zdarzenia są dostępne dla układu dopiero w następnym momencie dyskretnego czasu (następnym cyklu zegara).

Pierwszym krokiem etapu translacji jest podział modelowanej maszyny stanowej (diagram UML) na automaty FSM. Przykład podziału maszyny stanowej na automaty FSM przedstawiono na rys. 5.

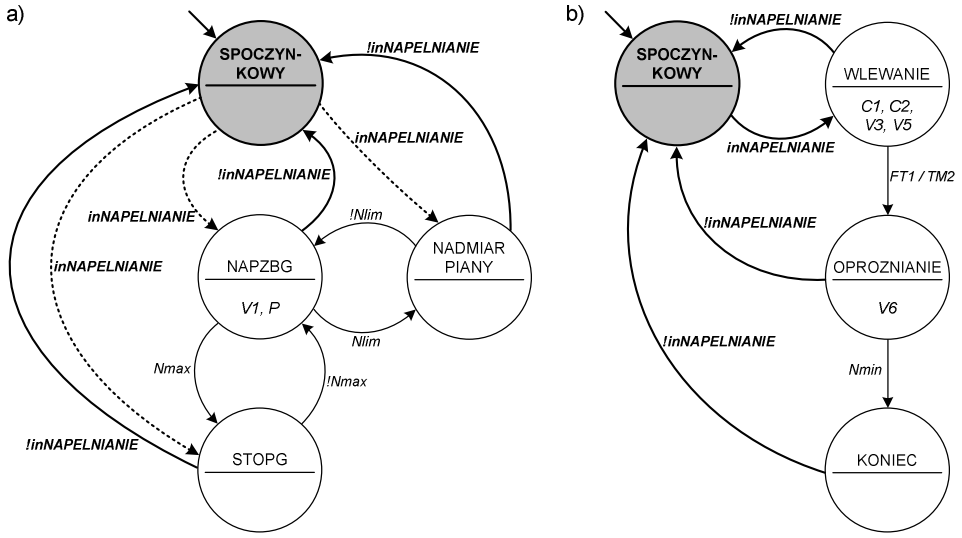
Kolejnym etapem translacji jest dodanie do każdego podrzędnego automatu FSM stanu spoczynkowego (neutralnego) (ang. *idle state*), do którego jest przekazane sterowanie

w czasie, kiedy połączony automat nadrzędny jest nieaktywny. Oprócz stanu spoczynkowego automat FSM należy uzupełnić także o specjalne przejścia do tego stanu. Zmodyfikowane dwa przykładowe automaty FSM przedstawiono na rys. 6.



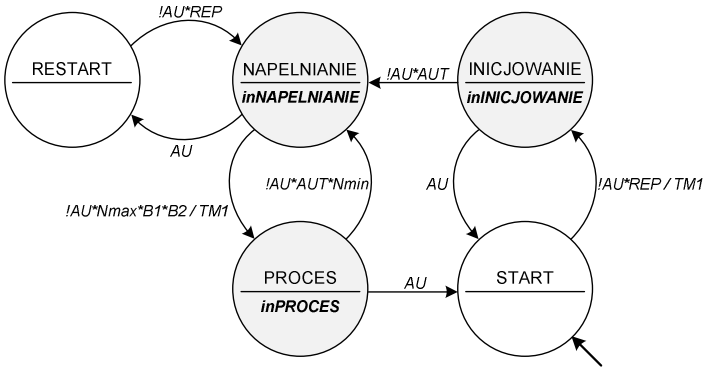
Rys. 5. Podział diagramu maszyny stanowej na automaty FSM (Mieszalnik):
a) automat nadrzędny, b) automaty podrzędne

Fig. 5. Modular decomposition of UML state machine diagram (Industrial Mixer):
a) top FSM, b) lower level FSMs



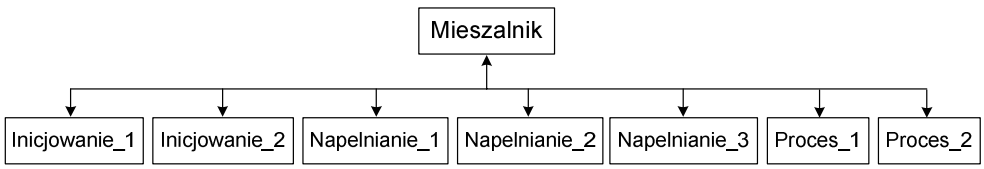
Rys. 6. Wybrane automaty FSM z uzupełnionymi przejściami i stanami bezczynności (Mieszalnik)

Fig. 6. Some of FSMs with additional special transitions and idle states (Industrial Mixer)



Rys. 7. Nadrzędny automat FSM z uzupełnionymi sygnałami (Mieszalnik)

Fig. 7. Top FSM with additional special signals (Industrial Mixer)



Rys. 8. Hierarchiczna struktura połączonych automatów FSM (Mieszalnik)

Fig. 8. Hierarchical structure of linked FSMs (Industrial Mixer)

```
module NazwaModułu (CLK,Reset,SygnałyWejściowe,SygnałyWyjściowe,
                    inSTANK, ..., inSTANL);
  input CLK, Reset, SygnałyWejściowe;     // wejścia
  output inSTANK, ..., inSTANL;           // sygnały związane z komunikacją
                                          // między automatem nadrzędnym i
                                          // automatami podrzędnymi
  wire inSTANK, ..., inSTANL;             // umieszczane w stanach złożonych
  output SygnałyWyjściowe;                // wyjścia
  reg SygnałyWyjściowe;                   // wyjścia
  reg [N:0] stany;
    parameter INITIAL = N'b00000;
    parameter STAN1   = N'b00001;
    parameter STAN2   = N'b00010;
    [...]
    parameter STANM   = N'b11111;
  always @(posedge CLK or Reset) begin
       if (Reset) begin
              stany <= INITIAL;
              // działania związane z realizacją maszyny stanowej
       end
       else case (stany)
              INITIAL : begin
                     // działania związane z realizacją maszyny stanowej
              end
              STAN1 : begin
                     // działania związane z realizacją maszyny stanowej
              end
              [...]
            endcase
  end
  assign inSTANK =(stany==STANK)?1:0;     // sygnały związane z komunikacją
  [...]                                   // między automatem nadrzędnym
  assign inSTANL =(stany==STANL)?1:0;     // i automatami podrzędnymi
endmodule
```

Rys. 9. Szablon nadrzędnego automatu FSM w języku Verilog

Fig. 9. Template for higher level FSM description in Verilog

Umieszczenie na diagramie (rys. 4) pseudostanu historii [21, 9] wskazuje, że działanie tak oznaczonego automatu sekwencyjnego determinowane jest jego poprzednią aktywnością. Występujące na rys. 6a) przejścia związane z realizacją atrybutu historii w automacie FSM oznaczono linią przerywaną. Oczywiście, w danym momencie dostępne będzie tylko jedno aktywne przejście ze stanu spoczynkowego (aktywny sygnał *inNapelnianie*) do tego stanu, który był ostatnim aktywnym stanem przed wystąpieniem sygnału *!inNapelnianie*. Sposób realizacji historii w automacie FSM w języku Verilog przedstawiono na rys. 10.

Kolejnym krokiem jest uzupełnienie nadrzędnego automatu FSM o dodatkowe sygnały związane z aktywnością poszczególnych automatów podrzędnych (rys. 7). Sygnały te należy dodać tylko w stanach złożonych – w przykładzie Mieszalnika są to stany: *INICJOWANIE*, *NAPELNIANIE* i *PROCES*. Komunikacja między automatem nadrzędnym i automatami podrzędnymi w układzie odbywa się właśnie za pomocą tych sygnałów. Jeżeli automat nadrzędny znajduje się w stanie złożonym (np. *NAPELNIANIE*), generowany jest sygnał *inNAPELNIANIE*, co z kolei powoduje aktywność wszystkich automatów podrzędnych, wrażliwych na zmianę sygnału *inNAPELNIANIE*: *NAPELNIANIE_1*, *NAPELNIANIE_2*, *NAPELNIANIE_3*.

```
module NazwaModułu (CLK,Reset,SygnałyWejściowe,SygnałyWyjściowe,inSTANK);
  input CLK, Reset, SygnałyWejściowe;          // wejścia
  input inSTANK;                    // sygnały związane z komunikacją między
                                    // automatem nadrzędnym i tym automatem
  output SygnałyWyjściowe;                     // wyjścia
  reg SygnałyWyjściowe;                        // wyjścia
  reg [N:0] stany;
  reg [N:0] stany_historia;                    // jeśli występuje historia
    parameter SPOCZYNKOWY = N'b00000;          // stan spoczynkowy
    parameter STAN1       = N'b00001;
    parameter STAN2       = N'b00010;
    [...]
    parameter STANM       = N'b11111;
  always @(posedge CLK or Reset) begin
    if (Reset) begin
          stany <= SPOCZYNKOWY;
          stany_historia <= SPOCZYNKOWY;
          // działania związane z realizacją maszyny stanowej
    end else if (inSTANK) begin
          stany <= stany_historia;             // poprzedni aktywny stan
          case (stany)
             SPOCZYNKOWY : begin
                // działania związane z realizacją maszyny stanowej
             end
             STAN1 : begin
                // działania związane z realizacją maszyny stanowej
             end
          [...]
          endcase
       end else begin
          // działania związane z przejściem do stanu spoczynkowego
          stany_historia <= stany;
       end
  end
endmodule
```

Rys. 10. Szablon podrzędnego automatu FSM w języku Verilog

Fig. 10. Template for lower level FSM description in Verilog

```
module Top (CLK, Reset, SygnałyWejściowe, SygnałyWyjściowe);
input CLK, Reset, SygnałyWejściowe;     // wejścia
wire inSTANK, ..., inSTANL;             // sygnały związane z komunikacją
                                        // między automatem nadrzędnym
                                        // i automatami podrzędnymi
output SygnałyWyjściowe;                // wyjścia
Modul1   FSM_TOP  (CLK, Reset, Sygnały);
Modul2   FSM_1    (CLK, Reset, Sygnały);
[...]
ModulX   FSM_X    (CLK, Reset, Sygnały);
endmodule
```

Rys. 11. Szablon jednostki nadrzędnej „Top” w języku Verilog

Fig. 11. Template for top level FSM description in Verilog

Opisane powyżej przekształcenia modelu prowadzą do stworzenia specjalnego modelu, w którym poszczególne moduły tworzą hierarchiczną strukturę komunikujących się automatów FSM (jak na rys. 8). Automaty na tym samym poziomie hierarchii mogą pracować równolegle, oczywiście – przy aktywności automatu nadrzędnego. Strukturę tę można więc potraktować jako model HCFSM (ang. *Hierarchical Concurrent Finite State Machine*) [1, 14]. Dla przykładu Mieszalnika struktura ta ma tylko dwa poziomy hierarchii, co – rzecz jasna – nie oznacza, że opracowana metoda nie może być stosowana dla układów z bardziej rozbudowaną hierarchią.

W ostatnim etapie każdy automat FSM konwertowany jest do języka opisu sprzętu Verilog. Translacja dokonywana jest na podstawie opracowanych w języku Verilog szablonów automatu nadrzędnego FSM (rys. 9), automatu podrzędnego (rys. 10) oraz jednostki nadrzędnej „Top" (rys. 11).

5. Wyniki eksperymentów

Tak przygotowaną specyfikację w języku Verilog można następnie poddać symulacji i syntezie w zewnętrznych środowiskach, np. *Active-HDL* firmy Aldec lub *Xilinx ISE* firmy Xilinx. W celu określenia przydatności proponowanej metody projektowej wygenerowane modele w języku Verilog zostały poddane syntezie i implementacji z wykorzystaniem oprogramowania *Xilinx ISE* firmy Xilinx. Wyniki syntezy i implementacji przykładowego modelu Mieszalnika zestawiono w tab. 1.

Tabela 1

Wyniki syntezy i implementacji Mieszalnika dla układu XCV50BG256 z rodziny Virtex

#SLICE FLIP FLOPS	%SLICE FLIP FLOPS	#4 input LUTs	%4 input LUTs	#IOB	%IOB	#SLICE	%SLICEs
37	2	79	5	26	14	41	5

6. Wyniki pracy na tle literatury przedmiotu

Problematyka specyfikacji hierarchicznych i współbieżnych maszyn stanów w sposób syntetyczny została przedstawiona w pracy [14]. Realizacja układowa map stanów Harela (diagramów *statechart*) pojawia się m.in. w publikacjach [10, 11]. Sposoby opisu diagramów *statechart*, a następnie maszyny stanowej UML w języku VHDL można znaleźć np. w pracy [24]. Sposób implementacji diagramów *statechart* z wykorzystaniem diagramów OBDD i języka VHDL oraz opis systemu CAD został zamieszczony w [21]. Oryginalna koncepcja autorów została zarysowana m.in. w publikacjach [5, 6]. Najbardziej zbliżone, podobne rozwiązanie metodologiczne zostało ostatnio (listopad 2008) przedstawione w artykule [28]. Dotyczy ono jednak implementacji w języku VHDL i związane jest z odmiennym modelem pośrednim. Projektowanie logiczne współbieżnych maszyn stanów z wykorzystaniem dualnej specyfikacji – diagram UML i sieć Petriego – zostało przedstawione w pracy [2].

7. Podsumowanie

Zaproponowana metoda bazuje na graficznej reprezentacji sterownika logicznego. Zastosowanie diagramów UML pozwala – z jednej strony – w sposób stosunkowo łatwy i intuicyjny na specyfikację behawioralną układu (diagramy maszyny stanowej), z drugiej natomiast – otwiera możliwość wykorzystania do tego celu, często darmowego, oprogramowania UML. Pozwala to na obniżenie kosztów projektowania układów cyfrowych na etapie specyfikacji.

Diagramy maszyny stanowej, będące podstawą opracowanej metody, pozwalają na kompletne i jednoznaczne specyfikowanie zachowania projektowanego systemu, a także wspierają takie jego cechy, jak współbieżność i hierarchiczność. Specyfikacja behawioralna algorytmu sterowania binarnego, przedstawiona w postaci użytecznego, nienadmiarowego i zwartego podzbioru diagramów maszyny stanowej, może być potraktowana jako abstrakcyjna forma programu dla sterownika logicznego.

Zagadnienie behawioralnej syntezy diagramów UML pojawia się w literaturze przedmiotu nieczęsto, jednak opracowanie metod i narzędzi do behawioralnej syntezy jest potrzebne. I tak jak wprowadzenie języków programowania wysokiego poziomu, takich jak Java, powoli wypiera języki niskiego poziomu, takie jak C, podobnie i narzędzia syntezy behawioralnej mają wyprzeć narzędzia syntezy na poziomie RTL (ang. *Register Transfer Level*) [30]. Przedstawiona metoda syntezy behawioralnej modeli opisanych diagramami maszyny stanowej UML jest implementowana i weryfikowana w autorskim systemie CAD (*UML-XML2Verilog*). System ten umożliwia automatyczną translację diagramów UML 2.1.2 opisanych w języku XML do języka opisu sprzętu Verilog. Tak przetworzona specyfikacja może być później użyta do przeprowadzenia symulacji projektowanego systemu, a następnie syntezy i implementacji układu.

Obecna wersja języka UML zawiera całe bogactwo środków służących do modelowania systemów nie tylko informatycznych. Wykorzystanie dodatkowo innych diagramów UML pozwala lepiej określić potrzeby i oczekiwania użytkownika, zaprojektować interfejs (diagramy przypadków użycia), opracować procedury testujące (diagramy czynności, sekwencji), stworzyć pełniejszą dokumentację projektu, a także poprawić komunikację pomiędzy grupami projektantów. Wykorzystanie innych diagramów UML w procesie projektowania programów dla sterowników logicznych oraz weryfikacja ich spójności są przedmiotem dalszych prac autorów.

Literatura

[1] Adamski M., *Petri Nets in ASIC Design*, Applied Mathematics and Computer Science, Vol. 3, No. 1, Proceedings of the ACEP Workshop Borowice, Poland, 1992, Wyższa Szkoła Inżynierska w Zielonej Górze, Zielona Góra 1993, 169-179.

[2] Adamski M., *Logic design of reconfigurable logic controllers*, IEEE Second International Symposium on Industrial Embedded Systems, SIES ’07, Lizbona 2007, 373-376.

[3] Adamski M., Chodań M., *Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC*, Wydawnictwo Politechniki Zielonogórskiej, Zielona Góra 2000.

[4] Bauer N., Engell S., *A Comparison of Sequential Function Charts and Statecharts and an Approach towards Integration*, Workshop INT '02, http://tfs.cs.tu--berlin.de/~mgr/int02/papers/bauer.ps.gz, 58-69.
[5] Bazydło G., Adamski M., *Graficzna specyfikacja programów dla sterowników logicznych z wykorzystaniem języka UML*, Materiały VII Krajowej Konferencji Naukowej Reprogramowalne Układy Cyfrowe 2005 – RUC '05, Szczecin 2005, 65-71.
[6] Bazydło G., *Specyfikacja behawioralna dla rekonfigurowalnych sterowników logicznych z wykorzystaniem diagramów maszyny stanowej z języka UML 2.0*, Pomiary Automatyka Kontrola 5 '2007, Vol. 53, 21-23.
[7] Booch G., Rumbaugh J., Jacobson I., *UML przewodnik użytkownika*, WNT, Warszawa 2001.
[8] Buchenrieder K., Pyttel A., Veith Ch., *Mapping Statechart Models onto an FPGA-Based ASIP Architecture*, EURO-DAC '96 European Design Automation Conference with EURO-VHDL '96, Genewa 1996, 184-189.
[9] Dąbrowski W., Stasiak A., Wolski M., *Modelowanie systemów informatycznych w języku UML 2.1*, PWN, Warszawa 2007.
[10] Drusinsky D., Harel D., *Using Statecharts for Hardware Description and Synthesis*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 8, 1989, 798-807.
[11] Drusinsky D., Harel D., *Electronic Controller Based on the Use of Statecharts as an Abstract Model*, USA Patent nr 4799141, 1989.
[12] Drusinsky-Yoresh D., *A state assignment procedure for single-block implementation of state charts*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 10, 1991, 1569-1576.
[13] Fowler M., *UML w kropelce wersja 2.0*, LTP Media Software, Warszawa 2005.
[14] Gajski D.D., Vahid F., Narayan S., Gong J., *Specification and Design of Embedded Systems*, PTR Prentice Hall, New Jersey, USA 1994.
[15] Harel D., *Statecharts, A visual formalism for complex Systems, Science of Computer Programming*, Science of Computer Programming, Vol. 8, 1987.
[16] Harel D., Politi M., *Modeling Reactive Systems With Statecharts: The Statemate Approach*, McGraw Hill Text, New York, USA 1998.
[17] Harel D., *Rzecz o istocie informatyki, Algorytmika*, WNT, Warszawa 1992.
[18] I-Logix, *Statemate Magnum, HDL Code Generator Reference Manual*, 2000.
[19] Jaragh M., Saleh I.A., *Modeling Computer hardware using the Unified Modeling Language*, TENCON '02. Proceedings, 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering, 2002, Vol. 1, 19-22.
[20] Kol R., Ginosar R., Samuel G., *Statecharts Methodology for the Design, Validation, and Synthesis of Large Scale Asynchronous Systems*, Proceedings of the 2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems – ASYNC '96, 1996, 164-174.
[21] Łabiak G., *Wykorzystanie hierarchicznego modelu współbieżnego automatu w projektowaniu sterowników cyfrowych*, Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, Zielona Góra 2005.
[22] Łach J., Sapiecha E., Zbierzchowski B., *Synteza układów sekwencyjnych w strukturach FPGA z wbudowanymi blokami pamięci*, Przegląd Telekomunikacyjny, Rocznik LXXVI, nr 2–3/2003, 81-86.

[23] Łuba T., Zbierzchowski B., Zbysiński P., *Układy reprogramowalne dla potrzeb telekomunikacji cyfrowej*, Przegląd Telekomunikacyjny, Rocznik LXXV, nr 5/2002, 321-329.
[24] McUmber W.E., Cheng B.H.C., *UML-Based Analysis of Embedded Systems Using a Mapping to VHDL*, High-Assurance Systems Engineering, 1999, 4th IEEE International Symposium, Washington 1999.
[25] OMG, *OMG Unified Modeling Language, Infrastructure, V2.1.2*, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF, 2007.
[26] OMG, *OMG Unified Modeling Language, Superstructure, V2.1.2*, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF, 2007.
[27] OMG, *XMI Specification, V2.1.1*, http://www.omg.org/spec/XMI/2.1.1, 2007.
[28] Wood S. K., Akehurst D. H., Uzenkov O., Howells W. G., McDonald-Maier K.D., *A Model Driven Development approach to mapping UML State Diagrams to synthesizable VHDL*, IEEE Transactions on Computers, Vol. 57, No. 10, 2008.
[29] Wrycza S., Marcinkowski B., Wyrzykowski K., *Język UML 2.0 w modelowaniu systemów informatycznych*, Helion, Gliwice 2005.
[30] Zwoliński M., *Projektowanie układów cyfrowych z wykorzystaniem języka VHDL (wydanie 2)*, WKiŁ, Warszawa 2007.