Grzegorz BAZYDŁO, **Marian ADAMSKI**
UNIWERSYTET ZIELONOGÓRSKI, ul. Podgórna 50, 65-246 Zielona Góra

Obsługa wyjątków w maszynie stanowej UML realizowanej w mikrosystemach cyfrowych

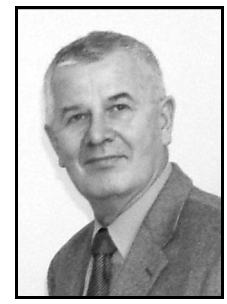
Mgr inż. Grzegorz BAZYDŁO

Mgr inż. Grzegorz Bazydło jest absolwentem Uniwersytetu Zielonogórskiego (2004). Ukończył studia na specjalności Inżynieria Komputerowa. Obecnie jest doktorantem na Wydziale Elektrotechniki, Informatyki i Telekomunikacji UZ. Zainteresowania naukowe koncentrują się wokół nowoczesnych metod projektowania specjalistycznych układów cyfrowych.

e-mail: G.Bazydlo@weit.uz.zgora.pl

Prof. dr hab. inż. Marian ADAMSKI

Dyrektor Instytutu Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Zainteresowania badawcze obejmują projektowanie mikrosterowników logicznych oraz formalnych metod syntezy, analizy i weryfikacji rekonfigurowalnych układów cyfrowych. Członek IEEE, IET, ACM, PTETiS, Polskiego Towarzystwa Informatycznego oraz Brazylijskiego Towarzystwa Mikroelektronicznego.



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

Streszczenie

W referacie przedstawiono i przedyskutowano zagadnienia związane z modelowaniem obsługi wyjątków opierając się na metodzie syntezy behawioralnej sterowników logicznych opisanych diagramami maszyny stanowej UML. Specyfikacją końcową jest modularny opis w języku opisu sprzętu Verilog. Zwrócono uwagę na poprawne stosowanie przejść bezwarunkowych oraz wprowadzanie stanów końcowych, pseudostanów historii oraz niejawnych zdarzeń typu *completion event*. Metoda została poparta stosownymi przykładami.

Słowa kluczowe: UML, sterownik, mikrosystem cyfrowy, Verilog, FPGA.

Exception handling in a state machine realised as digital microsystems

Abstract

The paper presents the design methodology and related framework for deriving Verilog descriptions from UML state machine diagrams in order to capture the behavioral hierarchy in the array structure of an embedded system. The exception handling is introduced at the top level of the graphical specification. As an intuitive example the interrupt is introduced, which illustrates a case of system failure, when the control is temporarily transferred to exceptional safe and determined behavior. The precise semantic interpretation of UML 2.2 state machine diagrams ensures, under the proposed structural design rules, that Verilog description conserves modular properties of an initial specification. The behavioral hierarchy of UML state machine is directly mapped into structural hierarchy inside the designed reconfigurable controller. The tree of properly encapsulated submachines allows independent simulation and modification of particular parts of behavioral model.

Keywords: UML, reconfigurable controller, Verilog, FPGA.

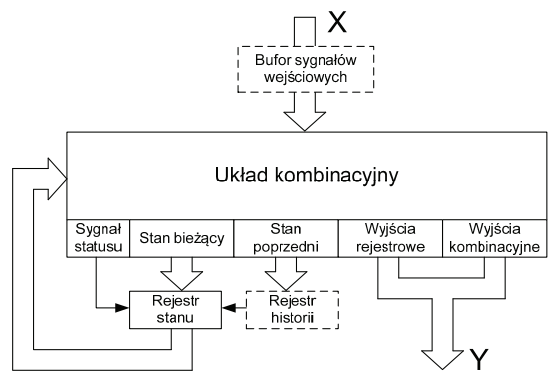
1. Wstęp

W artykule przedstawiono metodę modelowania wbudowanych, rekonfigurowalnych sterowników logicznych, opisanych diagramami maszyny stanowej UML. Szczególną uwagę zwrócono na obsługę sytuacji wyjątkowych [1] takich jak przerwanie (ang. *interrupt*) lub wywłaszczenie (ang. *abortion*). Modelowanie w standardowym języku graficznym, stosowanym już powszechnie w inżynierii oprogramowania ma szereg zalet, stąd zostało ono również zaadaptowane przez światowy przemysł elektroniczny jako zalecana, perspektywiczna forma specyfikacji behawioralnej mikrosystemów wbudowanych. Odwzorowanie współbieżnego, hierarchicznego algorytmu sterowania binarnego w formie strukturalnego, modularnego opisu funkcjonowania w językach opisu sprzętu, takich jak VHDL lub Verilog, jest przedmiotem badań naukowo-technicznych w wielu ośrodkach przemysłowych i uniwersyteckich na świecie [2, 3, 4, 5]. Większość wyników dotychczas opublikowanych prac koncentruje się na stosunkowo prymitywnej formie tekstowej specyfikacji behawioralnej, prowadzącej do rozmycia się starannie przemyślanej przez projektanta hierarchii, współbieżności oraz modularności [2]. Podczas odwzorowywania w strukturze matrycowej FPGA następuje niekontrolowane rozproszenie początkowo sensownie powiązanych i uporządkowanych fragmentów rekonfigurowalnego układu cyfrowego. Zagmatwana struktura implementacyjna znacząco utrudnia modyfikację wyselekcjonowanych bloków układu przez rekonfigurację [6]. Dodatkowo wprowadzana obsługa sytuacji wyjątkowych, polegająca na przykład na wprowadzeniu sterownika do stanu bezpiecznego po wykryciu awarii w systemie, zazwyczaj wymaga przebudowy całego projektu.

2. Wbudowany sterownik cyfrowy

W pracy przedstawiono sprawdzoną eksperymentalnie koncepcję odwzorowania hierarchiczno-współbieżnej maszyny stanów, przedstawionej w sposób behawioralny diagramem UML, na równoważny mu pod względem semantycznym opis tekstowy w języku Verilog na strukturalnym poziome RTL. Uzyskuje się w ten sposób przejrzystą, uporządkowaną, łatwo modyfikowalną budowę sekwensera, skonfigurowanego w formie hierarchiczno-współbieżnych automatów cyfrowych. Składa się ona z automatów (maszyn stanów) realizujących zaplanowane operacje sterownika oraz z nadrzędnego automatu, koordynującego ich pracę. Warto podkreślić, że translacja z plików XML otrzymanych z ogólnodostępnych edytorów diagramów UML, poprzez pośrednią obiektową strukturę danych, na syntezowalny opis w języku Verilog odbywa się w sposób w pełni automatyczny, za pomocą oryginalnego systemu CAD [7].

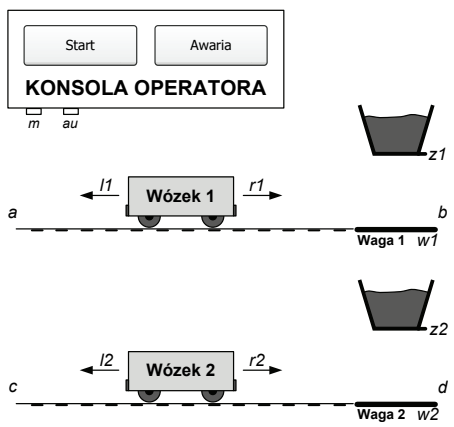
Sekwenser został wyposażony w dwa rozproszone, wirtualne rejestry stanu globalnego. Stan globalny jest traktowany jako superpozycja częściowych stanów lokalnych. Kodowane stany wewnętrzne przedstawiają zarówno pełną aktualną konfigurację stanu globalnego w rejestrze stanu bieżącego jak i zredukowaną tylko do niezbędnych rozmiarów częściową konfigurację poprzednią, zapisaną w *Rejestrze historii* (rys. 1).



Rys. 1. Model strukturalny sterownika jako mikrosystem cyfrowy
Fig. 1. Logic controller as a digital microsystem

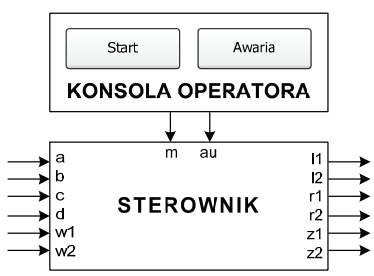
3. Przykład układu sterowania

Praktyczne wykorzystanie metody zaprezentowano na przykładzie procesu sterowania ruchem wózków, wzorowanym na książce [8]. Schemat sterowanego obiektu przedstawiono na rys. 2. Proces rozpoczyna się po naciśnięciu przycisku *Start* (aktywny sygnał *m*). Wózki 1 i 2 startują z położenia *a* oraz *c* i poruszają się w prawo. Po dojechaniu do punktów *b* i *d*, następuje otwarcie zsypów *z1* oraz *z2* i załadowanie wózków. Po załadowaniu powrót wózków następuje w kolejności: najpierw wraca wózek 1, a gdy ten dojedzie do punktu *a*, wówczas wraca wózek 2. Jeżeli w trakcie ruchu w prawo lub ładowania wystąpi awaria (aktywny sygnał *au*), praca systemu jest wstrzymywana. Po usunięciu awarii (nieaktywny sygnał *au* oznaczony jako *!au*) proces jest kontynuowany. Natomiast jeżeli awaria wystąpi w drugiej fazie pracy systemu, a więc w czasie powrotu wózków w lewo, po usunięciu awarii system zaczyna pracę od fazy początkowej, a więc ruchu wózków w prawo i ponownego ładowania.



Rys. 2. Przykładowy proces technologiczny
Fig. 2. Case study of technological discrete process

Schemat blokowy sterownika logicznego dla omawianego przykładu sterowania przedstawiono na rys. 3. Konsola operatora zawiera dwa przyciski, których wciśnięcie powoduje, że aktywne stają się sygnały *m* lub *au*, a wyciśnięcie powoduje dezaktywację tych sygnałów.

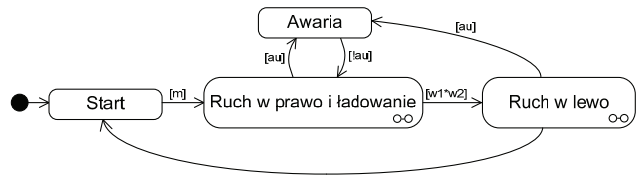


Rys. 3. Część sterująca i operacyjna sterownika
Fig. 3. Control and data part of logic controller

4. Maszyna stanowa UML

Język UML (wersja 2.2 [9]) to graficzny język modelowania, służący do obrazowania, specyfikowania, tworzenia i dokumentowania nie tylko systemów informatycznych [10], ale także mikrosystemów cyfrowych [11]. Podstawowym środkiem oferowanym przez język UML są diagramy, które można traktować jako szkic funkcjonowania systemu. Z punktu widzenia specyfikacji behawioralnej mikrosystemów cyfrowych najbardziej przydatne wydają się być diagramy maszyny stanowej, ponieważ są one graficznym odzwierciedleniem dyskretnego, skokowego zachowania skończonych systemów typu stan-przejście.

Na rys. 4 przedstawiono diagram maszyny stanowej na najwyższym poziomie hierarchii. Stany na diagramie reprezentowane są w postaci prostokątów z zaokrąglonymi rogami, a przejścia oznaczane są strzałkami. Z przejściem może być skojarzone zdarzenie uruchamiające, warunek oraz akcja wykonywana podczas realizacji przejścia. Ponieważ projektowany sterownik realizowany jest jako synchroniczny automat współbieżny z jednym sygnałem zegarowym przyjęto, że zdarzeniami uruchamiającymi są impulsy synchronizujące [12]. W związku z tym o realizacji przejścia między stanami decydują warunki, przedstawiane jako wyrażenia logiczne zbudowane z nazw sygnałów i operatorów logicznych (iloczyn, suma i negacja), np. *[x*!w]* oznacza, że przejście będzie zrealizowane jeżeli w momencie pojawienia się impulsu zegarowego sygnał *x* będzie miał wartość logiczną „1”, a *w* wartość „0”.



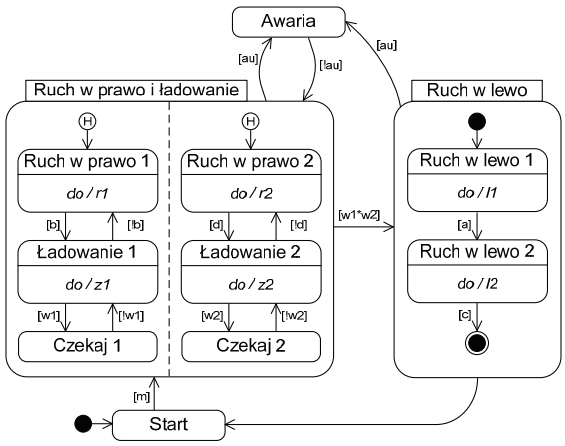
Rys. 4. Diagram maszyny stanowej – najwyższy poziom hierarchii
Fig. 4. State machine diagram – the highest level of hierarchy

Wspieranie przez diagramy maszyny stanowej takich cech układu jak hierarchiczność i współbieżność, pozwala w sposób intuicyjny i czytelny specyfikować zachowanie złożonych systemów współbieżnych na wybranym poziomie uszczegółowienia [13]. W prezentowanym przykładzie stany *Ruch w prawo i ładowanie* oraz *Ruch w lewo* to stany złożone.

Diagramy maszyny stanowej UML pozwalają także na wygodne modelowanie obsługi wyjątków, np. poprzez użycie przejść na wysokim poziomie hierarchii (ang. *high-level transition*). W prezentowanym przykładzie są to np. przejścia wychodzące i wchodzące do stanu *Awaria*. W przypadku wystąpienia awarii w systemie (aktywny sygnał *au*), sterowanie zostanie przekazane do stanu *Awaria*, automatycznie dezaktywując wszystkie podstany stanu złożonego *Ruch w prawo i ładowanie* lub *Ruch w lewo*. Zastosowanie przejść tego typu pozwala uniknąć tworzenia przejść z każdego podrzędnego stanu do stanu *Awaria*, co niepotrzebnie skomplikowałoby model i uczyniłoby go nieczytelnym. Ponadto zaburzyłoby to modularność diagramu (przejścia przekraczające granice stanów) lub oznaczałoby konieczność dodania nadmiarowych stanów w każdym stanie złożonym.

Innym elementem UML ułatwiającym modelowanie obsługi wyjątków jest możliwość modelowania z wykorzystaniem pseudostanów historii [10]. Występujący na rysunku 5 symbol historii płytkiej (litera *H* w okręgu) oznacza, że po uaktywnieniu stanu *Ruch w prawo i ładowanie* pierwszym aktywnym stanem będzie ten, który był aktywny ostatnio w momencie przekazania sterowania. Jeżeli stan *Ruch w prawo i ładowanie* staje się aktywnym po raz pierwszy, sterowanie zostanie przekazane do tego stanu, na który wskazuje pseudostan historii. Diagram maszyny stanowej dla prezentowanego przykładu zawierający wszystkie szczegóły przedstawiono na rys. 5.

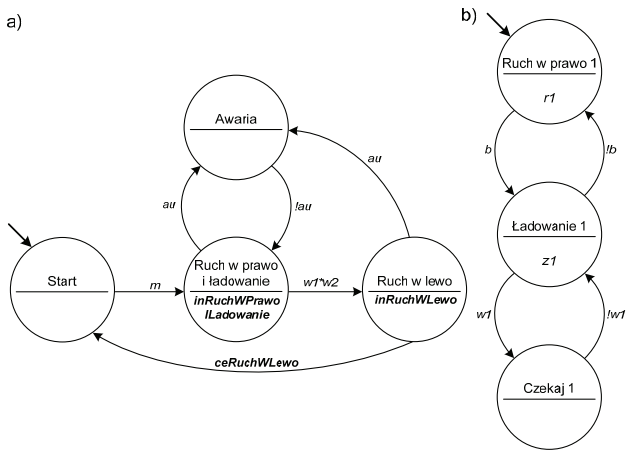
Warto zwrócić uwagę na występujące na diagramie (rys. 4, 5) przejście ze stanu *Ruch w lewo* do stanu *Start*. Mimo, że nie posiada ono żadnego warunku realizacji, nie jest traktowane jako, znane z automatów FSM, przejście bezwarunkowe. Zgodnie z notacją UML [9] przejście ze stanu złożonego posiada niejawne zdarzenie zakończenia (ang. *completion event*). W prezentowanym przykładzie przejście to będzie zrealizowane wyłącznie w przypadku zakończenia działania stanu złożonego *Ruch w lewo*, a więc kiedy sterowanie zostanie przekazane do stanu końcowego (aktywny sygnał *c*).



Rys. 5. Diagram maszyny stanowej – najniższy poziom hierarchii
Fig. 5. State machine diagram – the lowest level of hierarchy

5. Translacja maszyny stanowej na Verilog

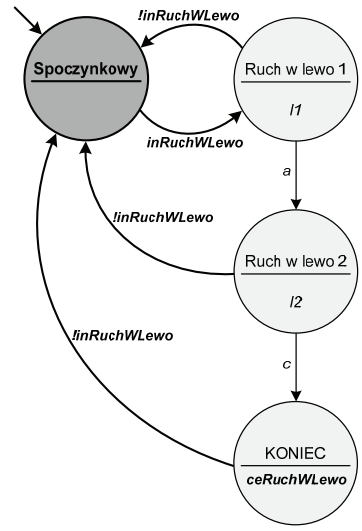
Kiedy sterownik logiczny zostanie zamodelowany za pomocą diagramów UML, kolejnym etapem jest translacja jego graficznej reprezentacji na syntezowalny opis w języku opisu sprzętu Verilog. Opracowana metoda syntezy została opisana w pracy [7]. Pierwszym krokiem etapu translacji jest podział modelowanej maszyny stanowej (diagram UML) na automaty FSM (rys. 6). W prezentowanym przykładzie sterowania model UML dekomponowany jest na jeden automat nadrzędny *Proces* (rys. 6a) oraz trzy automaty podrzędne: *RWPIL_1*, *RWPIL_2* oraz *RWL* (rys. 6b).



Rys. 6. Podział diagramu maszyny stanowej na automaty FSM:
a) automat nadrzędny, b) przykładowy automat podrzędny
Fig. 6. Modular decomposition of UML state machine diagram:
a) top FSM, b) example of lower level FSM

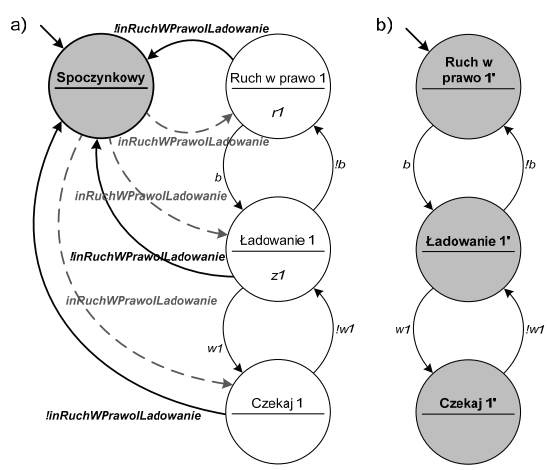
Następnie automaty nadrzędne FSM uzupełniane są o dodatkowe sygnały związane z aktywnością poszczególnych automatów podrzędnych (rys. 6a). Jeżeli automat nadrzędny znajduje się w stanie złożonym *Ruch w prawo i ładowanie* generowany jest sygnał *inRuchWPrawoILadowanie*, co z kolei powoduje aktywność wszystkich automatów podrzędnych, wrażliwych na zmianę tego sygnału. W prezentowanym przykładzie istnieje tylko jeden automat nadrzędny, ale w układach z bardziej rozbudowaną hierarchią mogą występować automaty podrzędne, które są jednocześnie nadrzędnymi wobec innych.

W kolejnym kroku do każdego podrzędnego automatu FSM dodawany jest stan spoczynkowy (neutralny), do którego przekazywane jest sterowanie w czasie, kiedy powiązany z nim automat nadrzędny jest nieaktywny. Oprócz stanu spoczynkowego dodawane są także specjalne przejścia do tego stanu. Zmodyfikowane przykładowe automaty FSM przedstawiono na rys. 7 (*RWL*) oraz rys. 8a (*RWPIL_1*).



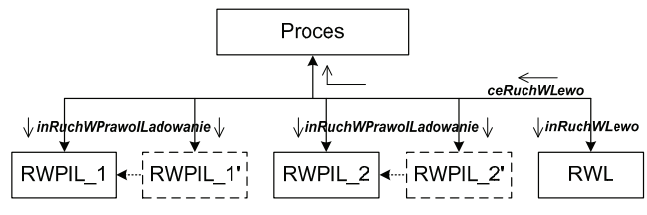
Rys. 7. Wybrany automat FSM z dodanymi przejściami i stanem bezczynności
Fig. 7. Example of FSM with additional special transitions and idle state

Występujące na rys. 8a przejścia oznaczone linią przerywaną związane są z realizacją atrybutu historii (z rys. 5) w automacie FSM i oznaczają alternatywne przekazanie sterowania w zależności od poprzedniej aktywności automatu. Informację o ostatnio aktywnym stanie przechowuje drugi automat (rys. 8b), zbudowany na bazie automatu z rys. 6b (usunięto sygnały wyjściowe stanu).

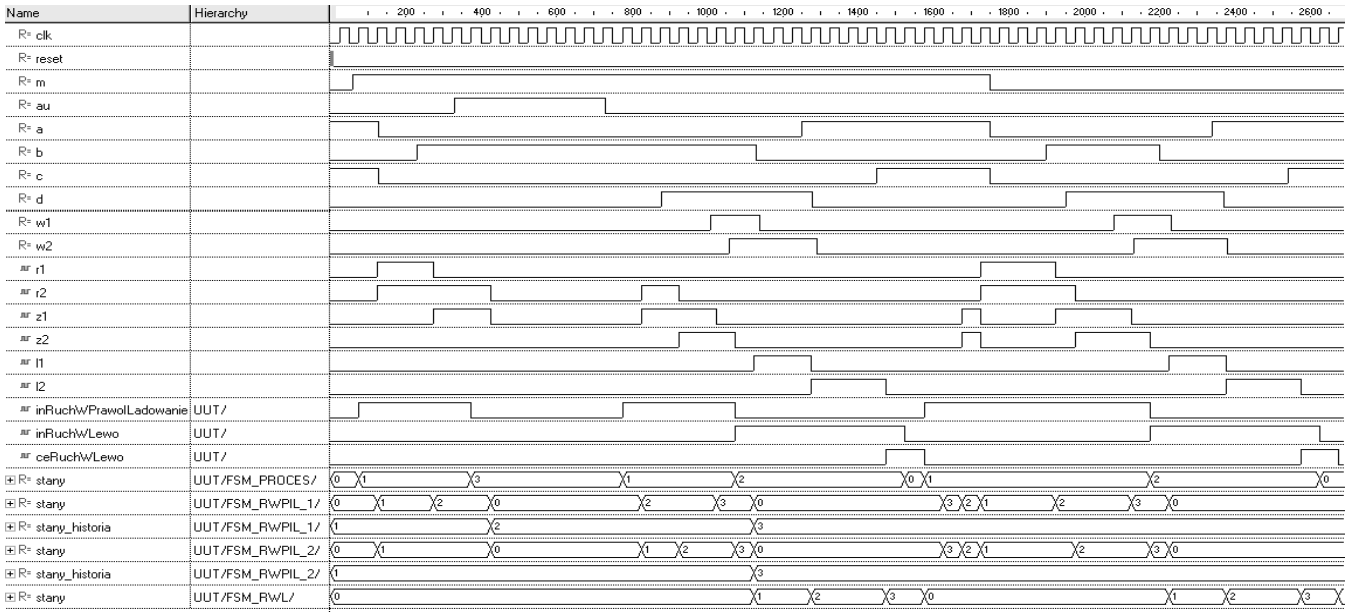


Rys. 8. a) wybrany automat FSM z dodanymi przejściami i stanem bezczynności,
b) dodatkowy automat FSM przechowujący ostatni aktywny stan
Fig. 8. a) example of FSM with additional special transitions and idle state,
b) additional FSM to keep last active state information

Opisane przekształcenia prowadzą do opracowania specjalnego modelu, w którym poszczególne moduły tworzą hierarchiczną strukturę komunikujących się automatów FSM (rys. 9).



Rys. 9. Hierarchiczna struktura połączonych automatów FSM
Fig. 9. Hierarchical structure of linked FSMs



Rys. 10. Wyniki symulacji układu
Fig. 10. Simulation's results

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*) [14, 15]. Dla prezentowanego przykładu struktura ta posiada tylko dwa poziomy hierarchii, co nie oznacza, że opracowana metoda nie może być stosowana dla układów z bardziej rozbudowaną hierarchią. Moduły *RWPIL_1'* oraz *RWPIL_2'* odpowiedzialne są za przechowywanie informacji o ostatnio aktywnych stanach, niezbędnych do realizacji funkcji historii w automacie *RWPIL_1* oraz *RWPIL_2*.

W ostatnim etapie każdy automat FSM konwertowany jest do języka opisu sprzętu Verilog. Translacja dokonywana jest w oparciu o opracowane w języku Verilog specjalne szablony, szczegółowo przedstawione w pracy [7]. Warto zaznaczyć, że struktura modelu HCFSM jest także odwzorowywana, czego efektem jest zbiór plików Verilog zamiast jednego opisu układu (jak ma to miejsce np. w [2]), który ze względu na dużą ilość elementów jest trudny do analizy i mało czytelny. Zaletą proponowanej metody jest możliwość niezależnej symulacji i syntezy każdego z modułów (automatu FSM). Wprowadzanie poprawek do modelu nie musi więc oznaczać konieczności generowania i ponownej symulacji całego układu, a tylko tego automatu, w którym dokonano zmiany. Otrzymaną specyfikację w języku Verilog poddano symulacji i syntezie w zewnętrznych środowiskach *Active-HDL* firmy Aldec oraz *Xilinx ISE* firmy Xilinx. Wyniki symulacji (rys. 10) potwierdzają, że zaprojektowany układ działa zgodnie ze specyfikacją wyrażoną w postaci diagramu maszyny stanowej UML.

6. Podsumowanie i kierunki dalszych prac

Zaproponowana metoda bazuje na graficznej specyfikacji mikrosystemu cyfrowego. Zastosowanie diagramów UML pozwala z jednej strony w sposób stosunkowo łatwy i intuicyjny na specyfikację behawioralną układu, z drugiej natomiast otwiera możliwość wykorzystania do tego celu także darmowego oprogramowania UML (obniżenie kosztów projektowania). Diagramy maszyny stanowej umożliwiają kompletne i jednoznaczne specyfikowanie zachowania projektowanego systemu.

Kierunki dalszych prac skupiają się wokół pełnej automatyzacji procesu projektowania sterowników rekonfigurowalnych, opisanych podzbiorem diagramów maszyny stanowej UML. Obecnie prowadzone prace dotyczą także możliwości wykorzystania innych diagramów UML (np. przypadków użycia, czynności, klas), a także modelowania z użyciem tzw. dualnej specyfikacji [16, 17].

7. Literatura

[1] Adamski M.: Logic synthesis of reconfigurable controllers. IEEE Second International Symposium on Industrial Embedded Systems, SIES'07, Lizbona, 2007, str. 373-376.
[2] Wood S., Akehurst D., Uzenkov O., Howells W., McDonald-Maier K.: A Model Driven Development Approach to Mapping UML State Diagrams to Synthesizable VHDL. IEEE Transactions on Computers, vol. 57, Nr 10, 2008.
[3] Harel D., Politi M.: Modeling Reactive Systems With Statecharts: The Statemate Approach. McGraw Hill Text, 1998.
[4] Łabiak G.: Wykorzystanie hierarchicznego modelu współbieżnego automatu w projektowaniu sterowników cyfrowych. Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, Zielona Góra, 2005.
[5] Adamski M., Karatkevich A., Węgrzyn M. (red): Design of embedded control systems. Springer, New York, 2005.
[6] Adamski M.: Design of reconfigurable logic controllers from hierarchical UML state machines. ICIEA 2009 – 4th IEEE Conference on Industrial Electronics and Applications. Xi'an, Chiny, 2009, str. 82-87.
[7] Bazydło G., Adamski M.: Projektowanie sterowników logicznych opisanych diagramami maszyny stanowej UML. Czasopismo Techniczne, seria Informatyka, Politechnika Krakowska, 1-I/2008, 2008.
[8] Adamski M., Chodań M.: Modelowanie układów sterowania dyskretnego z wykorzystaniem sieci SFC. Wydawnictwo Politechniki Zielonogórskiej, Zielona Góra, 2000.
[9] OMG: OMG Unified Modeling Language. Superstructure. v2.2. http://www.omg.org/spec/UML/2.2/Superstructure/PDF, 2009.
[10] Booch G., Rumbaugh J., Jacobson I.: UML przewodnik użytkownika. WNT, Warszawa, 2001.
[11] Bazydło G.: Synteza behawioralna sterowników rekonfigurowalnych na podstawie modelu maszyny stanowej UML. Pomiary, Automatyka, Kontrola, 7'2009.
[12] Minns P., Elliott I.: FSM based Digital Design using Verilog HDL, John Wiley & Sons Ltd, Chichester, Anglia, 2008.
[13] Harel D.: Statecharts, A visual formalism for complex Systems. Science of Computer Programming, Vol. 8, 1987.
[14] Gajski D., Vahid F., Narayan S., Gong J.: Specification and Design of Embedded Systems. PTR Prentice Hall, New Jersey, USA, 1994.
[15] Adamski M.: Petri Nets in ASIC Design. Applied Mathematics and Computer Science, vol. 3, WSI w Zielonej Górze, 1993.
[16] Basile F., Chiacchio P., Del Grosso D.: Modelling automation systems by UML and Petri Nets. Proceedings of the 9th International Workshop on Discrete Event Systems, Gooteborg, IEEE, 2008.
[17] Doligalski M.: Konwersja wybranych elementów maszyny stanów UML w ramach dualnej specyfikacji. Przegląd Elektrotechniczny, nr 7, 2009.

otrzymano / received: 07.05.2010
przyjęto do druku / accepted: 04.06.2010 *artykuł recenzowany*