**Valery SALAUYOU**, Łukasz ZABROCKI
POLITECHNIKA BIAŁOSTOCKA, WYDZIAŁ INFORMATYKI
ul. Wiejska 45A, 15-351 Białystok

# Wybór sposobów opisu w języku Verilog układów kombinacyjnych przy syntezie automatów skończonych

**Dr hab. inż. Valery SALAUYOU**

Dr hab. Inż. Valery Salauyou w latach 1980-1984 pracował jako programista w Mińsku na Białorusi. W latach 1984-2002 był pracownikiem dydaktycznym w Białoruskim Państwowym Uniwersytecie Informatyki i Radioelektroniki w Mińsku, gdzie uzyskał tytuł doktora habilitowanego nauk technicznych. Jednocześnie od 1992 roku pracuje jako adiunkt Wydziału Informatyki Politechniki Białostockiej. Zainteresowania naukowe: projektowanie systemów cyfrowych na układach programowalnych.

*e-mail: v.salauyou@.pb.edu.pl*

**Mgr inż. Łukasz ZABROCKI**

Mgr Łukasz Zabrocki absolwent Wydziału Informatyki Politechniki Białostockiej. Ukończył studia informatyki o specjalności Grafika Komputerowa. Zainteresowania naukowe koncentrują się wokół nowoczesnych metod projektowania automatów skończonych i języka Verilog.

*e-mail: luka_99@.wp.pl*

**Streszczenie**

Zbadane sposoby opisu układów kombinacyjnych automatów skończonych w języku Verilog, a problem wyboru najlepszego opisu z punktu widzenia kosztów realizacji. Problem został rozwiązany empirycznie. Zaproponowano siedem konstrukcji języka Verilog dla opisu układów kombinacyjnych, z których zostały wybrane dwie najlepsze konstrukcje. Pokazano, że wybór sposobu opisu pozwala zmniejszyć koszt realizacji średnio w 2,71 razy, a dla niektórych przypadków - w 3,40 razy. Praca ma duże znaczenie praktyczne.

**Słowa kluczowe**: układy kombinacyjne, automaty skończone, Verilog.

## Choice of combinational circuit specifications in the Verilog language at synthesis of finite state machines

**Abstract**

In the paper techniques of combinational circuit specifications in the Verilog language at synthesis of finite state machines (FSMs) are examined. The problem of the best specification choice for minimization of an FSM cost is considered. This task was empirically solved by performing a great many experimental researches. There were proposed seven Verilog language constructions for specification of the FSM combinational circuits, four with the statement *if* and three with the statement *case*, from which two best constructions were chosen on a basis of the experimental investigations. For different methods of the FSM description the comparison of the maximum and minimum cost of implementation was made. It was shown that the choice of the specification technique allowed reducing the FSM cost by a factor of 2.71 on the average and sometimes even by a factor of 3.40. This approach is of great practical importance, since it allows reducing the FSM realization cost and raising the FSM speed essentially without any special efforts from designers and application of any special synthesis methods.

**Keywords**: combinational circuits, finite state machines, Verilog.

## 1. Wprowadzenie

Opracowanie współczesnych systemów cyfrowych zwykle wymaga użycia języków projektowania wysokiego poziomu. Na dzień dzisiejszy najbardziej rozpowszechnione są dwa języka projektowania: VHDL i Verilog. Język Verilog pojawił się w środowisku projektantów sprzętu jako alternatywa języka VHDL i szybko zdobył popularność pośród praktykujących inżynierów. Na dzień dzisiejszy opracowano kilka standardów międzynarodowych języka Verilog [1-2], które są wspierane przez większość producentów komputerowego wspomagania projektowania systemów cyfrowych. W szczególności język Verilog jest powszechnie stosowany w projektowaniu systemów cyfrowych opartych na bazie układów programowalnych. Wspierany jest przez narzędzia do projektowania takich firm jak Altera, Xilinx, Cadence, Mentor Graphics, Actel itd.

Z drugiej strony automat skończony jest matematycznym modelem systemów cyfrowych i układów sekwencyjnych. W procesie projektowania systemu cyfrowego inżynierowie często spotykają się z potrzebą skonstruowania różnych automatów skończonych, których skuteczna realizacja znacznie przyczynia się do sukcesu cyfrowego systemu jako całości. Jednak język Verilog zapewnia dużą liczbę różnych sposobów opisu automatów skończonych. W związku z tym obecnie wyzwaniem jest znalezienie efektywnych sposobów opisywania automatów skończonych w języku Verilog.

Od momentu powstania, język Verilog wzbudził wielkie zainteresowanie w środowisku projektantów sprzętu cyfrowego. Napisano szereg książek na temat stosowania języka Verilog przeznaczonych dla użytkowników różnego poziomu, w szczególności dla użytkowników poziomu średniego [3, 4] i zaawansowanego [5, 6].

Analiza znanych publikacji pokazuje że do dnia dzisiejszego nie wykonano badania wszystkich możliwości języka Verilog pod względem opisu automatów skończonych.

W obecnej pracy zbadano sposoby opisu automatów skończonych w języku Verilog i rozpatrzono problem wyboru najlepszego sposobu opisu z punktu widzenia kosztu realizacji i szybkości działania automatu skończonego. Przedstawiony problem rozwiązywano empirycznie przy pomocy wykonania dużej liczby badań eksperymentalnych na wzorcowych przykładach automatów skończonych. Prezentowana praca ma duże praktyczne znaczenie i pozwala bez szczególnych wysiłków ze strony projektantów i bez zastosowania jakikolwiek specjalnych metod syntezy znacznie zmniejszyć koszt realizacji i powiększyć szybkość działania automatów skończonych.

## 2. Cechy opisu automatów skończonych w języku Verilog

W praktyce projektowania inżynierskiego największe rozpowszechnienie uzyskały dwa modele automatów skończonych: automat typu Mealy'ego i automat typu Moore'a. Funkcjonowanie automatu typu Mealy'ego można opisać za pomocą następujących równań:

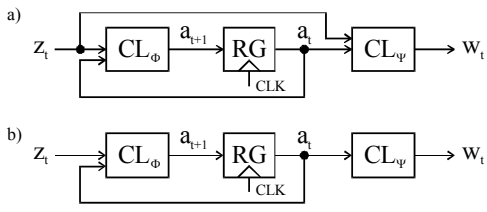
$$\begin{aligned} a_{t+1} &= \varphi(z_t, a_t); \\ w_t &= \psi(z_t, a_t); \end{aligned} \tag{1}$$

gdzie $\varphi$ - funkcja przejść; $\psi$ - funkcja wyjść; $z_t$ – wektor wejściowy (wartość zmiennych wejściowych) ustawiany w czasie $t$ ($t$=1,2,3,...); $w_t$ – wektor wyjściowy (wartość zmiennych wyjściowych), wzbudzany przez automat; $a_t$ – stan obecny automatu skończonego; $a_{t+1}$ – następny stan automatu skończonego.

Funkcjonowanie automatu typu Moore'a można opisać za pomocą następujących równań:

$$\begin{aligned} a_{t+1} &= \varphi(z_t, a_t); \\ w_t &= \psi(a_t); \end{aligned} \tag{2}$$

Charakterystyczną cechą automatu typu Mealy'ego jest to, że wektor wyjściowy wt zależy zarówno od wejściowego wektora $z_t$, jak i od stanu wewnętrznego $a_t$, natomiast w automacie typu Moore'a wektor wyjściowy $w_t$ zależy tylko od stanu wewnętrznego $a_t$. Na rys. 1a i rys. 1b przedstawiono strukturalne modele automatów typu Mealy'ego i Moore'a, gdzie układ kombinacyjny $CL_\varphi$ realizuje funkcje przejść φ, układ kombinacyjny $CL_\psi$ realizuje funkcje wyjść ψ, a rejestr RG – sterowany sygnałem zegarowym CLK – realizuje pamięć automatu skończonego.




Rys. 1. Strukturalne modele automatów skończonych: a – typu Mealy'ego; b – typu Moore'a
Fig. 1. Structural models of FSMs: a – Mealy; b – Moore

W najogólniejszym przypadku rozważany model automatu skończonego typu Mealy'ego (rys. 1a) – funkcjonowanie którego opisują równania (1) – umożliwia trzy sposoby opisu automatów skończonych w języku Verilog: z trzema procesami, z dwoma procesami i z jednym procesem. Opisanie automatu skończonego za pomocą jednego procesu możliwe jest tylko dla automatów typu Moore'a, dlatego dalej będą rozpatrywane tylko dwa sposobu opisu: z trzema i z dwoma procesami.

W sposobie opisu automatów skończonych z trzema procesami (Listing 1) pierwszy proces opisuje przejścia pomiędzy stanami wewnętrznymi (funkcja przejść), które realizuje się układem kombinacyjnym $CL_\varphi$. Drugi proces opisuje sygnały wyjściowe (funkcja wyjść), wzbudzane na przejściach pomiędzy stanami wewnętrznymi i realizowane układem kombinacyjnym $CL_\psi$. Trzeci proces opisuje funkcjonowanie pamięci automatu skończonego. Dla sposobu opisu automatów skończonych z dwoma procesami dwa pierwsze procesy łączą się w jeden proces.

**Listing 1.** Przykład opisu automatu skończonego z trzema procesami

```
module FSM_3 (input clk, reset, input [1:0] in,
              output reg [1:0] out);
    reg [1:0] state, nextstate;
    localparam [1:0] s0=0, s1=1, s2=2;
    always @(*)                    // pierwszy proces
        casex(state)
        s0: if(in==2'b00)    nextstate=s1;
            else if(in==2'b01) nextstate=s2;
        s1: if(in==2'b11)    nextstate=s1;
            else if(in==2'b10) nextstate=s2;
        s2:                  nextstate=s0;
        endcase
    always @(*)                          // drugi proces
        casex(state)
        s0: if(in==2'b00)    out=2'b00;
            else if(in==2'b01) out=2'bx1;
        s1: if(in==2'b11)    out=2'b11;
            else if(in==2'b10) out=2'b1x;
        s2:                          out=2'b00;
        endcase
    always @(posedge clk)                // trzeci proces
        if(~reset) state <= s0;
        else state <= nextstate;
endmodule
```

W ogólnym przypadku język Verilog nie nakłada żadnych ograniczeń na opis funkcjonowania automatu skończonego: można użyć jakichkolwiek operatorów języka Verilog i jakichkolwiek konstrukcji operatorów języka Verilog. Tradycyjnie dla sprawdzenia, że automat skończony znajduje się w odpowiednim stanie używany jest operator *case*, a do sprawdzenia warunków przejść z pewnego stanu może być używany zarówno operator *if*, jak i operator *case*.

*Wyjściowym stanem przejścia* (*present state*) nazywa się taki stan wewnętrzny automatu skończonego, od którego zaczyna się dane przejście. *Następny stan przejścia* (*next state*) to stan, w którym przejście kończy się. Projektanci programu Quartus II [7] polecają przy opisie automatów skończonych z operatorami *if* i *case* zawsze używać dodatkowo konstrukcji *else* i *default*, przy czym jako następny stan przejścia w konstrukcjach *else* i *default* używać wyjściowego stanu danego przejścia.

Dla w pełni określonych (zupełnych) automatów skończonych [8] użycie dodatkowych konstrukcji *else* i *default* w żaden sposób nie wpływa na funkcjonowanie automatu skończonego, ponieważ dane konstrukcje nigdy nie będą wykonywane.

Dla nie w pełni określonych (niezupełnych) automatów skończonych (*incompletely specified finite state machine*) użycie dodatkowych konstrukcji *else* i *default* ustala nieokreślone przejścia – przejściem w wyjściowy stan przejścia. Faktycznie nie w pełni określony automat skończony zastępuje się w pełni określonym automatem skończonym. Ponieważ w niezupełnych automatach skończonych zakłada się, że na wejściach automatu skończonego nigdy nie pojawią się wektory wejściowe, które odpowiadają nieokreślonym stanom przejść, to takie dookreślenie w żaden sposób nie wpływa na funkcjonowanie automatu skończonego.

W ten sposób użycie dodatkowych konstrukcji *else* i *default* nie wpływa na funkcjonowanie automatu skończonego. Jednak w realizacji układów kombinacyjnych $CL_\varphi$ i $CL_\psi$ pozwala na zastosowanie dekoderów zamiast bramek [7], co znacznie obniża koszt realizacji automatów skończonych.

## 3. Wybór konstrukcji języka Verilog do opisu układów kombinacyjnych automatów skończonych

W ogólnym przypadku przy opisie układów kombinacyjnych automatów skończonych możliwe są następujące warianty użycia operatora *if*:

IF_1 – sprawdzenie każdego warunku przejść za pomocą osobnego operatora *if* (polecane przez projektantów programu MAX+PLUS II [9]);

IF_2 – sprawdzenie pierwszego warunku przejść z pewnego stanu za pomocy operatora *if*, oraz sprawdzenie każdego następnego warunku przejść za pomocą konstrukcji *else if* (tradycyjne podejście przy opisie nie w pełni określonych automatów skończonych);

IF_3 – wariant powtarza poprzednią konstrukcję, za wyjątkiem tego, że ostatni warunek przejścia z pewnego stanu realizuje się za pomocą konstrukcji *else* (tradycyjne podejście przy opisie w pełni określonych automatów skończonych);

IF_4 – wariant powtarza konstrukcję IF_2, za wyjątkiem tego, że dodaje się konstrukcję *else*, za pomocą której realizuje się powrót w wyjściowy stan przejścia (przy opisie funkcji przejść) lub zerowy wektor wyjściowy (przy opisie funkcji wyjściowych), polecany przez projektantów programu Quartus II [7].

W podobny sposób przy opisie układów kombinacyjnych automatów skończonych możliwe są następujące warianty użycia operatora *case*:

CASE_1 – sprawdzenie każdego warunku przejścia za pomocą poszczególnej stałej, która odpowiada wartości zbioru zmiennych wejściowych (tradycyjne podejście przy opisie nie w pełni określonych automatów skończonych);

CASE_2 – wariant powtarza konstrukcję CASE_1 za wyjątkiem tego, że ostatni warunek przejścia z pewnego stanu realizuje się za pomocą konstrukcji *default* (tradycyjne podejście przy opisie w pełni określonych automatów skończonych);

CASE_3 – wariant powtarza konstrukcję CASE_1 za wyjątkiem tego, że dodaje się konstrukcję *default*, za pomocą której realizuje się powrót do wyjściowego stanu przejścia (przy opisie funkcji przejść) lub zerowy wektor wyjściowy (przy opisie funkcji wyjściowych), polecany przez projektantów programu Quartus II [7].

W ten sposób mamy 7 wariantów opisu układów kombinacyjnych automatów skończonych w języku Verilog. Listing 2 prezentuje przykład zastosowania wariantu IF_1 dla sprawdzenia trzech warunków przejść z pewnego stanu. Dla symulacji przełączania pomiędzy stanami automatu skończonego w dany opis zostały włączone sygnały wejściowe *clk* i *reset* oraz proces, który tworzy wartości wyjściowego wektora na rejestrze wyjściowym. Przykłady zastosowania innych wariantów użycia operatorów *if* i *case*, dla sprawdzenia innej ilości warunków przejść, buduje się w podobny sposób.

**Listing 2.** Przykład warianta IF_1 dla trzech warunków przejść

```
module IF_1_3     (input clk, reset, input [5:0] in,
                   output reg [5:0] out);
reg [5:0] out_t;

always@(*)                    /* wariant IF_1 */
begin
        if(in==0) out_t=2;
        if(in==1) out_t=1;
        if(in==2) out_t=0;
end

always@(posedge clk)          /* opis funkcjonowania pamięci */
        if(~reset) out<=out_t;
        else       out<=6'bx;
endmodule
```

Badania eksperymentalne efektywności konstrukcji języka Verilog zostały wykonane dla 19 przykładów, każdy z nich różni się od pozostałych liczbą sprawdzanych warunków. Synteza układów kombinacyjnych została wykonana za pomocą programu Quartus II wersii 8.1 dla układów programowalnych rodziny Cyclone III. Jako kryterium optymalizacji został przyjęty koszt realizacji: liczba elementów logicznych potrzebnych dla realizacji układu.

Tab. 1. Efektywność opisów układów kombinacyjnych automatów skończonych
Tab. 1. The efficiency of combinational circuit specifications for FSMs

| ex_*n* | IF_*x* | | | | CASE_*y* | | | $C_{max}$ | $C_{min}$ | $C_{max}/C_{min}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **1** | **2** | **3** | | | |
| 3 | 7 | 7 | 3 | 3 | 7 | 3 | 3 | 7 | 3 | 2,33 |
| 4 | 6 | 7 | 3 | 3 | 3 | 3 | 3 | 7 | 3 | 2,33 |
| 5 | 10 | 10 | 4 | 4 | 10 | 5 | 5 | 10 | 4 | 2,50 |
| 6 | 10 | 10 | 4 | 4 | 10 | 5 | 5 | 10 | 4 | 2,50 |
| 7 | 10 | 10 | 4 | 4 | 11 | 6 | 6 | 10 | 4 | 2,50 |
| 8 | 9 | 9 | 3 | 3 | 4 | 3 | 3 | 9 | 3 | 3,00 |
| 9 | 17 | 17 | 5 | 5 | 13 | 7 | 7 | 17 | 5 | **3,40** |
| 10 | 15 | 14 | 6 | 6 | 13 | 7 | 7 | 15 | 6 | 2,50 |
| 11 | 18 | 17 | 8 | 8 | 14 | 8 | 8 | 17 | 8 | 2,13 |
| 12 | 13 | 13 | 5 | 5 | 9 | 6 | 6 | 13 | 5 | 2,60 |
| 13 | 15 | 18 | 7 | 7 | 14 | 8 | 8 | 18 | 7 | 2,57 |
| 14 | 15 | 14 | 6 | 6 | 14 | 8 | 8 | 15 | 6 | 2,50 |
| 15 | 16 | 17 | 7 | 7 | 9 | 8 | 8 | 17 | 7 | 2,43 |
| 16 | 9 | 12 | 4 | 4 | 5 | 4 | 4 | 12 | 4 | 3,00 |
| 17 | 23 | 21 | 7 | 7 | 11 | 7 | 7 | 23 | 7 | 3,29 |
| 18 | 22 | 17 | 7 | 7 | 10 | 7 | 7 | 22 | 7 | 3,14 |
| 19 | 28 | 23 | 10 | 10 | 11 | 10 | 10 | 28 | 10 | 2,80 |
| 20 | 18 | 18 | 6 | 6 | 8 | 7 | 7 | 18 | 6 | 3,00 |
| 21 | 26 | 22 | 9 | 9 | 11 | 10 | 10 | 26 | 9 | 2,89 |
| mid | 15,11 | 14,5 | 5,68 | 5,68 | 9,84 | 6,42 | 6,42 | | | **2,71** |

Wyniki badań eksperymentalnych przedstawiono w tab. 1, gdzie ex_*n* – nazwa przykładu; *n* – liczba sprawdzanych warunków, $n \in [3,21]$; IF_*x* – warianty opisów z operatorem *if*, $x \in [1,4]$; CASE_*y* – warianty opisów z operatorem *case*, $y \in [1,4]$; $C_{max}$ i $C_{min}$ – maksymalne i minimalne wartości kosztu realizacji dla danego przykładu przy różnych wariantach opisu; $C_{max}/C_{min}$ – stosunek odpowiednich parametrów; mid – średnia wartość parametrów.

Analiza tab. 1 pokazuje, że warianty opisu IF_3 i IF_4 z operatorem *if*, a również warianty CASE_2 i CASE_3 z operatorem *case* dają takie same wyniki. Najlepsze wyniki pod względem kosztu realizacji otrzymano w przypadku wariantów opisu IF_3 i IF_4, dalej warianty opisów CASE_2 i CASE_3. Najgorsze wyniki otrzymano przy użyciu wariantu opisu IF_1 oraz wariantu opisu IF_2. Główny wniosek jaki można wyciągnąć po wykonaniu badań eksperymentalnych: sposób opisu układów kombinacyjnych w znacznym stopniu ma wpływ na koszt realizacji. Na to wskazuje stosunek $C_{max}/C_{min}$, którego średnia wartość równa jest 2,71, a dla niektórych przypadków 3,40. Innymi słowy – odpowiedni wybór sposobu opisu układów kombinacyjnych pozwala średnio 2,71 razy obniżyć koszt realizacji automatów skończonych.

Do dalszych badań opisów automatów skończonych zostały wybrane konstrukcje IF_4 i CASE_3 opisu układów kombinacyjnych, ponieważ:

(1) dane konstrukcje pozwalają na otrzymanie najlepszych wyniki dla operatorów *if* i *case*;
(2) są polecane przez projektantów programu Quartus II [7].

## 4. Wnioski

Rozpatrywana metodyka budowania sposobów opisu automatów skończonych w języku Verilog i efektywność wyboru sposobu opisu była zbadana dla układów programowalnych firmy Altera. Takie same badania mogą być wykonane również dla układów programowalnych innych producentów (na przykład firmy Xilinx).

W podobny sposób proponowaną metodykę można również stosować do badania sposobów opisu automatów skończonych w języku VHDL.

*Praca wykonana przy częściowym finansowym poparciu Politechniki Białostockiej (praca statutowa № S/WI/4/2008).*

## 5. Literatura

[1] IEEE Std 1364-1995, IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language/ The Institute of Electrical and Electronics Engineers, Inc. New York, NY, USA. – 653 p.
[2] IEEE Std 1364-2001 (Revision of IEEE Std 1364-1995), IEEE Standard Verilog Hardware Description Language, IEEE Computer Society / The Institute of Electrical and Electronics Engineers, Inc. New York, NY, USA. – 828 p.
[3] Thomas D.E., Moorby P.R.: The Verilog Hardware Description Language, Fifth Edition. – Kluwer Academic Publishers, New York, USA, 2002. – 381 p.
[4] Palnitkar S. Verilog HDL: A guide to digital design and synthesis, Second Editions. – Prentice Hall PTR, 2003. – 496 p.
[5] Ciletti M.D.: Advanced digital design with the Verilog HDL. - Prentice Hall, New Jersey, USA, 2003. – 985 p.
[6] Ramachandran S.: Digital VLSI system design. A design manual for implementation of projects on FPGAs and ASICs using Verilog. – Springer, Dordrecht, The Netherlands, 2007. – 709 p.
[7] Quartus II Handbook Version 8.1. – Altera Corporation, San Jose, CA, USA, 2008.
[8] Kam T., Villa T., Brayton R., Sangiovanni-Vincentelli A.: Synthesis of FSMs: functional optimization. Norwell, MA: Kluwer Academic Publishers, 1997. – 284 p.
[9] MAX+PLUS II Programmable logic development system. Text editor and AHDL. – Altera Corporation, San Jose, CA, USA, 1991. – 334 p.

*otrzymano / received: 20.05.2013*
*przyjęto do druku / accepted: 03.07.2013* *artykuł recenzowany / revised paper*