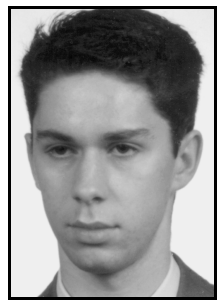
**Kamil MIELCAREK**
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

# Grafy doskonałe metodą dziur nieparzystych w automatycznej syntezie sterowników logicznych

**Mgr inż. Kamil MIELCAREK**

Kamil Mielcarek jest pracownikiem Instytutu Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Zajmuje się zastosowaniem grafów doskonałych w procesach automatycznej syntezy i optymalizacji układów cyfrowych. Jednocześnie zajmuje się zagadnieniami serwerowych systemów sieciowych opartych na systemie OpenBSD jak i ich bezpieczeństwem, oprogramowaniem oraz rozwiązaniami sieciowymi. Zajmował się też systemami czasu rzeczywistego opartymi o systemy BSD i Linux.

*e-mail: K.Mielcarek@iie.uz.zgora.pl*



**Streszczenie**

Obserwowany rozwój elektroniki i wszechobecna miniaturyzacja, objawiającą się coraz większą ilością urządzeń realizujących coraz bardziej skomplikowane zadania. Rozwój pociąga za sobą konieczność opracowywania nowych, bardziej efektywnych metod panowania nad ogromem zależności. Złożoność układów dawno przerosła możliwości pojedynczej osoby i obecnie takimi zadaniami obciąża się specjalizowane systemy komputerowe, jednak tego typu system należy odpowiednio przygotować. Pożądane jest, aby dawał wyniki optymalne w jak najkrótszym czasie. Złożoność problemów występujących w czasie automatycznego przygotowania układów cyfrowych powoduje, że w praktyce stosowane są algorytmy heurystyczne, dające wyniki przybliżone i nie zawsze w najkrótszym możliwym czasie. Pogoń za nowymi rozwiązaniami doprowadziła do pomysłu wykorzystania grafów doskonałych, które przez swoje własności pozwalają zmniejszyć wymagania czasowe a zatem i wymaganą moc obliczeniową, dając w zamian wyniki optymalne. Algorytmy wykorzystujące specyficzne własności grafów doskonałych charakteryzują się złożonością obliczeniową na poziomie wielomianowym. Zanim można będzie operować na grafach doskonałych należy sprawdzić czy dany graf jest grafem doskonałym. Najnowsze prace wskazują, że grafy doskonałe można rozpoznawać (pośród innych metod) z użyciem algorytmów wyszukiwania dziur nieparzystych. Jednocześnie obserwuje się, że znacząca większość grafów opisujących rzeczywiste układy zawiera się w podklasach grafów doskonałych. W roku 1960, Claude Berge wysunął tezę mówiącą ze graf jest doskonały wtedy i tylko wtedy, gdy nie zawiera ani dziur nieparzystych ani anty-dziur nieparzystych. Teza jest znana jako Strong Perfect Graph Conjecture. (Chvátal i Sbihi zaproponowali nazwę grafu Bergea) wg propozycji Dziura natomiast, jest bezcięciwowym cyklem o długości przynajmniej cztery, zaś anty-dziura jest dopełnieniem takiego cyklu. Dziury i anty-dziury są natomiast parzyste i nieparzyste zgodnie z parzystością ich liczby wierzchołków. Nieparzysta dziura jak i nieparzysta anty-dziura nie są doskonałe, bowiem ich liczby klikowe wynoszą odpowiednio 2 oraz 2k+1 natomiast liczby chromatyczne mają odpowiednio 3 oraz k+1, co jednoznacznie uniemożliwia im być grafem doskonałym (liczba chromatyczna grafu doskonałego G, jest równa liczbie kliki grafu, dla każdego indukowanego podgrafu grafu G). Artykuł ma na celu wskazanie potencjalnej użyteczności algorytmów do rozpoznawania grafów doskonałych. Zagadnienia występujące w procesach automatycznej syntezy dają się rozwiązać w czasie wielomianowym, gdzie dla grafów w ogólności problem złożoności należy do klasy problemów NP-zupełnych. Zastosowanie grafów doskonałych jako pośredniego modelu formalnego jest o tyle użyteczne, że znacząca większość zależności, opisujących rzeczywiste układy sekwencyjne, daje w wyniku grafy należące do podklasy grafów doskonałych [5].

**Słowa kluczowe**: grafy doskonałe, algorytmy rozpoznawania grafów doskonałych, teoria grafów, automatyczna synteza sterowników.

## Perfect Graphs Using Odd-Hole Free Graph Properties In Automatic High Level Synthesis

**Abstract**

This paper should to point out potential strenght of prerfect graph algorithms - specially algorithms with use of odd-hole-free graph - for automated synthesis of digital circuits. Typically known problems are NP-complete but using perfect graphs complexity is decreasing to polynomial. Studies in this matter shows that plenty of dependencies describing real sequential circuits can be described using perfect graphs. There shown the analysis of digital controller described in SFC. In analysis was used methods for testing perfect graphs.

**Keywords**: prefect graphs, odd-hole free graphs, prefect graphs recognizing algorithms, graph theory, automatic synthesis of digital controllers.

## 1. Wstęp

Opierając się na własnościach grafów doskonałych, można postawić tezę, że jeśli układ opisany został za pomocą grafów doskonałych to każdy z elementów układu (każdy z podgrafów) również jest opisany grafem doskonałym. Daje to możliwość dekompozycji systemu i weryfikacje poszczególnych modułów. Prowadzi to do stwierdzenia, ze poprawnie zaprojektowany układ, reprezentowany przez graf niebędący doskonałym, potencjalnie zawiera błędy lub nie został zaprojektowany w należyty sposób. Nie jest możliwe w chwili obecnej stwierdzić czy układy zawsze są opisywane grafami doskonałymi, jednak zdecydowana większość należy do podklas grafów doskonałych [5].

Artykuł ma na celu rozwinięcie tematyki wskazanej w [7] oraz potencjału i użyteczności algorytmów do rozpoznawania grafów doskonałych. Zagadnienia występujące w procesach automatycznej syntezy dają się rozwiązać w czasie wielomianowym, gdzie dla grafów w ogólności problem złożoności należy do klasy problemów NP-zupełnych.

Zastosowanie algorytmów rozpoznających grafy doskonałe na podstawie wyszukiwania dziur jest tez pośrednio metodą określenia przynależności grafów do podklasy grafów doskonałych, co jest nie bez znaczenia przy zastosowaniu pośredniego modelu formalnego opartego o grafy doskonałe. Jeśli graf jest grafem doskonałym, całkowicie pozbawionym dziur, to z definicji musi należeć do klasy grafów trójkątnych, będących jedna z podklas użyteczną podczas procesów automatycznej syntezy układów. Jest też wstępem do rozpoznania grafów interwałowych przydatnych w procesach harmonogramowania. Istnieje możliwość rozpoznania przynależności grafu do grafów doskonałych poprzez wyszukiwanie dziur nieparzystych, co powodowałoby dalsze uproszczenie złożoności obliczeniowej algorytmów.

Zastosowanie grafów doskonałych jako pośredni model formalny jest o tyle użyteczne, że znacząca większość zależności, opisujących rzeczywiste układy sekwencyjne, daje w wyniku grafy należące do podklasy grafów doskonałych [5], co pozwala przypuszczać, że sprawdzając powstające grafy opisujące układ będzie można zweryfikować jego poprawność, jak również jego części.

Artykuł jest kontynuacją rozważań zawartych w [8].

## 2. Dziury, Antydziury i Grafy Doskonałe

### Rozpoznawanie przez wyszukiwanie nieparzystych dziur

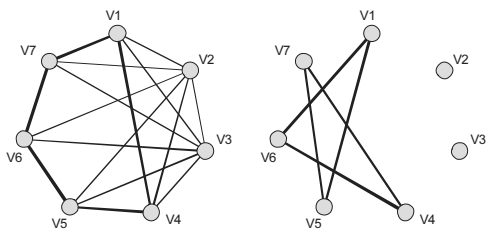
Algorytmy rozpoznawania grafów doskonałych opierają się o metodę poszukiwania nieparzystych dziur (ang. odd-hole) w grafach prostych skończonych [6]. Dowiedziono, że graf jest grafem doskonałym wtedy i tylko wtedy, gdy jest grafem Berge'a. Natomiast graf Berge'a można rozpoznać, gdy ani graf ani jego dopełnienie nie zawierają nieparzystych dziur. Nieparzysta dziura nigdy nie będzie grafem doskonałym. Jeśli więc graf zawiera nieparzystą dziurę to - w myśl definicji - istnieje możliwość indukowania podgrafu niebędącego grafem doskonałym, przez co graf nie może być doskonały.

Dziura, (*ang. graph hole*) to alternatywna nazwa dla bezcięciwowego cyklu o długości przynajmniej 4, dla uproszczenia

nazywanego dziurą (Chvátal). Dziury są nazywane parzystymi, jeśli mają parzystą liczbę wierzchołków oraz nieparzyste, jeśli liczba wierzchołków jest nieparzysta. Dopełnienie dziury jest nazywane anty-dziurą (*ang. graph antihole*). Żadna nieparzysta dziura nie jest grafem doskonałym, (bowiem liczba kliki jest równa *2* a liczba chromatyczna *3*), jednocześnie anty-dziura też nie należy do grafów doskonałych (liczba kliki anty-dziury z *2k+1* wierzchołkami jest równa *k* a jej liczba chromatyczna wynosi *k+1*).

Jednocześnie brak dziur w grafie powoduje, że graf jest albo prosty albo zawiera podgraf 2-gwiezdny (*ang. 2-star*) lub dwudzielny, (*ang. 2-join*). To stwierdzenie jest podłożem wyjścia do algorytmów rozpoznawania, czy graf przynależy do rodziny grafów doskonałych, poprzez rozłożenie grafu *G* na *m* podgrafów i stwierdzenie czy każdy z $G_i$ gdzie *i=1,2,…,m* - prostszy podgraf jest grafem bez nieparzystych dziur. Należy wspomnieć, że wyszukanie 2-gwiazdy jest problemem, który można rozwiązać w czasie wielomianowym. Rozpoznanie 2-gwiazdy opartej o wierzchołki *u* i *v*, w czasie wielomianowym polega na sprawdzeniu czy zostaną rozłączone niesąsiadujące wierzchołki *x* i *y*, przez usunięcie wszystkich sąsiadów *u* i *v* z wyjątkiem *x* i *y*. Natomiast wielomianowy algorytm wyszukiwania podgrafu 2-dzielnego można znaleźć w [11, 12].

Autorzy [6] podają metodę rozpoznawania grafów doskonałych z użyciem dekompozycji 2-gwiazd oraz dekompozycji 2-dzielnej oraz algorytmy tworzenia grafów spełniających inne kryteria na potrzeby tych algorytmów.




Rys. 1. Graf G (a) oraz jego dopełnienie (b) zawierające nieparzystą „dziurę”
Fig. 1. Graph G (a) and complement (b) containing „odd-hole”

Przykładem grafu zawierającego nieparzystą „dziurę” może być graf *G* (rys. 1). Można w nim wyróżnić dziurę, na którą składa się cykl bez cięciw $V_1,V_4,V_5,V_6,V_7,V_1$. Jednocześnie można zauważyć, że w dopełnieniu grafu też można wyróżnić tego rodzaju konstrukcję zawierającą te same wierzchołki. Dla dopełnienia zauważamy cykl $V_1,V_5,V_7,V_4,V_6,V_1$ również będący nieparzystą dziurą. Jak widać - opierając się na definicji grafów Berge'a, ten graf nie jest grafem Berg'a, a co za tym idzie nie jest grafem doskonałym.

W ostatnich latach zaprezentowane były dwa algorytmy rozpoznawania grafów doskonałych o złożoności wielomianowej *„Recognizing Berge Graphs”* [14] - algorytm o złozoności rzedu $O(n^9)$ oraz *„A Polynomial Algorithm for Recognizing Perfect Graphs”* [6] - algorytm o złożoności rzędu $O(n^{20})$, jednak jak widać problem nadal pozostaje złożoność obliczeniowa algorytmów. Problem powstaje przez konieczność wykorzystania „czyszczenia” grafu [14].

Michele Conforti, Gérard Cornuéjols, i Kristina Vušković udowodnili, że każdy graf Berge'a bez kwadratów (dziur o długości 4) przynależy do dwóch podstawowych klas: grafów dwudzielnych oraz liniowych grafów dwudzielnych. Inaczej mówiąc, posiadają jedna z dwóch błędnych struktur „*star-cutset*” lub „*2-join*”. Skoro grafy dwudzielne i liniowe grafy dwudzielne są doskonałe oraz to, że najmniejszy niedoskonały graf Berge'a nie zawiera żadnej z tych struktur, to oznacza, że każdy z grafów Berge'a niezawierający kwadratów jest doskonały.

Do przybliżenia zagadnienia wyszukiwania dziur można posłużyć się poniższym algorytmem [13]. Nie jest to algorytm doskonały, jednak pozwala zrozumieć proces poszukiwań.

```
wejście: graf spójny nieskierowany G
wyjście: tak/nie w zależności czy graf zawiera dziury.

1. Inicjowanie i wyzerowanie tablic not_in_hole[ ] i in_path[ ]
   wylicz macierz sąsiedztwa A[ ] grafu G;
2. dla każdego wierzchołka u grafu G wykonaj
   2.1 in_path[u] ← 1;
   2.2 dla każdej krawędzi vw grafu G wykonaj
       jeśli u jest sąsiadem v I nie jest sąsiadem w i not_in_hole[(u; v);w] = 0 to wtedy
          in_path[v] ← 1;
          process(u, v, w);
          in_path[v] ← 0;
   2.3 in_path[u] ← 0;
3. Wypisz: graf G nie zawiera dziury.

process(a; b; c)
1. in path[c] ← 1;
2. dla każdego wierzchołka d sąsiadującego z c w grafie G wykonaj
   2.1 jeśli d nie jest sąsiadem a ani b w grafie G
       to wtedy {fabcd jest P4 grafu G}
   2.2 jeśli in_path[d] = 1
       to wtedy wypisz graf G posiada dziurę; STOP.
       w przeciwnym razie jeśli not_in_hole[(b; c); d] = 0
       to wtedy process(b; c; d);
3. in_path[c] ← 0;
4. not in hole[(a; b); c] ← 1;
not in hole[(c; b); a] ← 1;
```

Rys. 2. Algorytm wyszukiwania dziur w grafie; [13]
Fig. 2. Hole finding algorithm for general graphs; [13]

Zastosowanie powyższego algorytmu pozwoliłoby znaleźć odpowiedz, czy w grafie znajdują się dziury, co jak powiedziano wcześniej daje odpowiedz czy graf może przynależeć do klasy grafów trójkątnych.

```
wejście: graf spójny nieskierowany G
wyjście: tak/nie w zależności czy graf zawiera dziury.

1. Inicjowanie i zerowanie tablic visited_P3[ ] oraz in_path[ ];
   wylicz macierz sąsiedztwa grafu G;
2. dla każdego wierzchołka grafu G wykonaj
   2.1 in_path[u] ? 1;
   2.2 dla każdej krawędzi vw grafu G wykonaj
       jeśli u nie sąsiaduje z v ani w i visited_P3[(v,w), u] = 0 wtedy
          in_path[v] ? 1;
          process(v, u,w);
          in path[v] ? 0;
   2.3 in_path[u] ? 0;
3. Wypisz: G nie posiada anty-dziur.

process(a, b, c)
1. in path[c] ? 1;
2. visited_P3[(a, c), b] ? 1;
   visited_P3[(c, a), b] ? 1;
3. dla każdego wierzchołka d sąsiadującego z b w grafie G wykonaj
   3.1 jeśli d jest sąsiadem a i nie jest sąsiadem c w grafie G wtedy
       {abcd jest a P4 of G}
   3.2 jeśli in_path[d] = 1 wtedy
       Wypisz: G posiada anty-dziure; stop;
       else
       jeśli visited_P3[(b, d), c] = 0 wtedy process(b, c, d);
4. in path[c] ? 0;
```

Rys. 3. Algorytm wyszukiwania anty-dziur w grafie; [13]
Fig. 3. Anti-Hole finding algorithm for general graphs; [13]

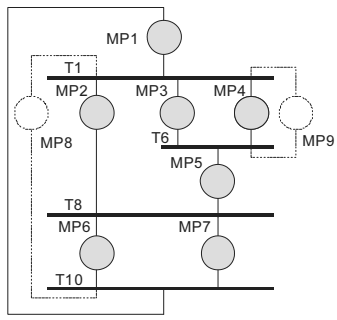
Rozpoznawanie grafu za pomocą dziur nieparzystych sprowadza się do użycia algorytmów opracowanych przez zespoły badaczy – matematyków [5, 14]. Metoda ta jest jednak dość problematyczna, bowiem należy sprawdzać występowanie dziur nieparzystych do pewnej ich wielkości (np. 1, 3, 5, 7, 9...). Algorytmy działające w/g tego założenia mają złożoność obliczeniową, która potencjalnie będzie mogła wykluczyć tę metodę jako użyteczną w szybkiej zautomatyzowanej syntezie. Złożoność w notacji N() sięga $N(n^9)$.

Lepszą byłą by metoda używania zależności wynikających z zależności powstających podczas konstrukcji grafów. Z uwagi na ich specyfikę, gdzie poprawny, indukowany podgraf jest grafem doskonałym, wynika, że każdy mniejszy układ będący częścią większego, opisanego grafem doskonałym będzie tez opisany grafem doskonałym. Istnieje przypuszczenie (prace trwają), że jest możliwość zbudowania zależności pozwalających na odwrócenie procesu dekompozycji. Czyli kompozycja z możliwością wykorzystania zbadanych wcześniej zależności w stosunku do modułu i umieszczeniu go wewnątrz biblioteki. Znając parametry wynikające z zastosowania algorytmów grafów doskonałych, można będzie z ich użyciem budować większe układy spełniające kryte-

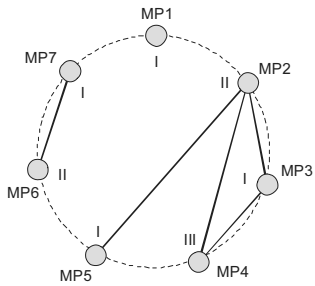
rium doskonałości. Daje to możliwość dalszego zaoszczędzenia czasu na weryfikację już rozpoznanych modułów. Oczywiście koniecznym jest zachowanie pewnych zależności. Jednocześnie jest kolejnym elementem przybliżającym prace nad systemem automatycznej syntezy z użyciem grafów doskonałych.

## 3. Zastosowanie (przykład)

Przedstawione w [7] metody, w połączeniu z metodami opisanymi w innych pracach pozwalają na przeprowadzenie uproszczonej przykładowej analizy. Niech dany jest opis układu w postaci grafu SFC. Za przykład niech posłuży sieć pochodząca z załącznika standardu IEC 1131-3 [15]. Oryginalny sposób interpretacji układowej sterownika rekonfigurowalnego przedstawiono w pracach M. Adamskiego [16, 17]. Poniżej przedstawiono opis sterowania mieszalnikiem już w postaci makrosieci Petriego.




Rys. 4. Makrosieć Petriego odpowiadająca modelowi SFC [15]
Fig. 4. Petrie macro-net eqivalent to SFC model from [15]




Rys. 5. Graf zgodności - modelu z rys. 4
Fig. 5. Compatibility graph of the model from fig. 4

Analiza dyskretna przestrzeni stanów lokalnych automatu współbieżnego, modelującego program sterownika logicznego, generuje grafy, które podczas analizy okazały się być grafami doskonałymi. Po wyznaczeniu relacji współbieżności między miejscami makrosieci, np. analizując graf znakowań makrosieci, uzyskano graf zgodności (współbieżności) między miejscami. Na podstawie analizy stwierdzono, że graf ten jest
- złożony z trzech składowych spójności,
- każda z nich odpowiada składowej automatowej,
- składowe są grafami doskonałymi,
- przynależy do podklasy grafów trójkątnych*,
- przynależy do podklasy grafów porównywalności*.

* (w tym przypadku)

W związku z tym jego kolorowanie daje się wykonać w czasie wielomianowym. W rezultacie kolorowania otrzyma się trzy składowe podsieci automatowe:
I: $\{MP_1, MP_3, MP_5, MP_7\} = \{P_1, P_5, P_6, P_7, P_9, P_{10}, P_{11}, P_{12}, P_{13}\}$
II: $\{MP_2, MP_6\} = \{P_2, P_3, P_4, P_{14}, P_{15}\}$
III: $\{MP_4\} = \{P_8, P_{16}\}$

## 4. Podsumowanie

Metody projektowania układów obecnie używane bardzo szeroko stosują formalne modele pośrednie, do wewnętrznej i zewnętrznej reprezentacji struktur układów logicznych. Powszechnie stosowane są rozwiązania na bazie grafów czy sieci Petriego. Analiza większej liczby sieci Petriego, opisującej rzeczywiste układy sterowania podsuwa przypuszczenie, że relacja uporządkowania przestrzeni stanów lokalnych takiej sieci jest opisana grafem doskonałym. Wniosek ten potwierdza pośrednio [5], w której przeprowadzono znaczną liczbę testów na grafach opisujących rzeczywiste układy cyfrowe. Wyniki prac eksperymentalnych w tej dziedzinie wskazują, że znacząca większość rzeczywistych układów cyfrowych opisanych grafami należącymi do klasy grafów doskonałych daje się właściwie pokolorować w prosty sposób w czasie wielomianowym. Jednocześnie brak jest prostego algorytmu rozpoznawania grafów doskonałych metodą wyszukiwania nieparzystych dziur. Istniejące metody sprowadzają problem do dekompozycji grafu na mniejsze części gdzie dokonywane są sprawdzenia własności, które są wykonywalne w czasie wielomianowym.

Poszukiwane są metody rozpoznawania, czy relacja współbieżności w przestrzeni stanów lokalnych dowolnego sterownika logicznego jest opisana grafem doskonałym. Algorytmy rozpoznawania grafów trójkątnych, porównywalności i interwałowych zostały zrealizowane praktycznie z wykorzystaniem języka C. Jednocześnie trwają prace nad implementacją algorytmów wielomianowych rozpoznawania grafów przez wyszukiwanie dziur nieparzystych [6, 14].

## 5. Literatura

[1] Golumbic M.C.: Algorithmic Graph Theory and Perfect Graphs, Courant Institute of Mathematical Science, New York University, Academic Press 1980.
[2] Lovasz L.: A Characterization of Perfect Graphs, Journal of Combinatorial Theory, Series B, vol.13, 1972, pp.95-98.
[3] Cornuejols G.: The Strong Perfect Graph Conjecture, International Congress of Mathematicians, Bejing, China, 2002.
[4] Cornuejols G.: The Strong Perfect Graph Theorem, Ann. of Math, 2006.
[5] O. Coudert: Exact Coloring of Real-Life Graphs is Easy, Synopsys, Inc. 700 East Middlefield Rd., Moutain View, Proc. of the 34th Design Automation Conference DAC,, Anaheim, CA, USA, 1997, pp. 121-126.
[6] Cornuejols G., Xinming Liu, Vuskovic K.: A polynomial algorithm for recognizing perfect graphs, Proceeding 44th Annual IEEE Symposium, 2003.
[7] Mielcarek K: Grafy doskonałe jako alternatywa dla automatycznej syntezy i optymalizacji układów cyfrowych, Metody i systemy komputerowe w badaniach naukowych i projektowaniu inżynierskim, IV Krajowa Konferencja. Kraków, Polska, 2003, Oprogramowanie Naukowo-Techniczne, 2003.
[8] Mielcarek K: Rozpoznawanie grafów doskonałych na potrzeby automatycznej syntezy układów cyfrowych, KNWS'06, Pomiary Automatyka Kontrola 6bis/2006.
[9] Giovani De Micheli: Synteza i optymalizacja układów cyfrowych, Wydawnictwa Naukowo-Techniczne, Warszawa 1998.
[10] Leuker G. S., Interval graph algorithms. Ph.D. thesis, Princeton University.
[11] Conforti M., Cornuejols G., Kapoor A., Vuskovic K., Even-hole-free-graphs, Part II: Recognition algorithm, Journal of Graph Theory 40, 2002, pp.238-266.
[12] Cornuejols G., Cunnigham W.H.: Compositions for perfect graphs, Discrete Mathematics 55, 1985, pp.245-254.
[13] Nikolopoulosy S.D., Paliosz L.: Hole and Antihole Detection in Graphs, Algorithmica, Springer Science + Busines Media Inc., 2007.
[14] Chudnovsky M., Cornuéjols G., Xinming L., Seymour P., Vuškovič K.: Recognizing Berge Graphs, Princeton University, Carnegie Mellon Princeton University and Princeton, University of Leeds, 2002, revised 2004.
[15] International Electrotechnical Commission: Programable controllers. Part 3 – Programing Languages IEC 1131-3, Genewa, 1993.
[16] Adamski M.: Metodologia projektowania reprogramowalnych sterowników logicznych z wykorzystaniem elementów CPLD i FPGA, Reprogramowalne Układy Cyfrowe RUC 98, Szczecin, 1998, pp.15-22.
[17] Adamski M.: Aplication Specific logic Controler for Safety Critical Systems, IFAC Triennial Congress, Beijing Chiny, 1999, vol. 0, pp.519-529.

*Artykuł recenzowany*