Lech GRODZKI
POLITECHNIKA BIAŁOSTOCKA, WYDZIAŁ ELEKTRYCZNY

Porównanie implementacji programowych automatów sterujących w sterownikach wbudowanych

Dr inż. Lech GRODZKI

Absolwent Wydziału Elektrycznego Politechniki Białostockiej (1985). W 1996r. uzyskał tam również stopień doktora nauk technicznych w dyscyplinie elektrotechnika. Obecnie pracuje jako adiunkt w Katedrze Automatyki i Elektroniki Politechniki Białostockiej. Zajmuje się projektowaniem i programowaniem sterowników mikroprocesorowych.

e-mail: lgrodzki@we.pb.edu.pl

Streszczenie

Programowanie sterowników wbudowanych polega na wykorzystaniu standardowych języków i na-rzędzi programowania. Zadanie sterowania sekwencyjnego realizowanego przez sterownik może być w różny sposób zaimplementowane w wybranym środowisku programowania. W zależności od tego uzy-skuje się programy różniące się między sobą szybkością pracy i zapotrzebowaniem na pamięć operacyj-ną. Parametry te stają się istotne przy doborze optymalnej konstrukcji systemu mikroprocesorowego ste-rownika. Praca zawiera porównanie efektywności różnych wariantów oprogramowania tego samego za-dania sterowania.

Słowa kluczowe: sterowniki wbudowane, automaty, języki programowania.

Comparison of software implementations of controlling automatons in embedded controllers

Abstract

Programming of embedded microcontrollers is based on usage of the standard programming lan-guages and tools: from machine-level symbolic instructions to high-level languages - mainly C. One of the typical control task, which is usually implemented in microcontroller software, is logic sequential control. Programme implementation of that task depends on: used programming language and its possi-ble syntax. Sometimes we can also use dedicated aiding system, which generates software implementation of the control task from its formal description to insert it into controller software. Usage of different programming languages and techniques causes that control software can work slower or faster and needs less or more operating memory. The time and memory complexity of the control software decide about the controller hardware and are important, when this hardware should be optimal. The article contains the comparison of time and memory efficiency of the exemplary control task according to usage of dif-ferent programming languages, techniques and even programming tools. Conclusions pointed in the pa-per can be useful for microcontroller designer and programmer.

Keywords: embedded microcontrollers, automatons, programming languages.

1. Wstęp

Jedną z dziedzin zastosowania sterowników są układy sterowania sekwencyjnego kontrolujące pracę pojedynczych maszyn i urządzeń. Wykorzystuje się do tego standardowe sterowniki PLC lub dedykowane systemy mikroprocesorowe, wbudowane w urządzenia, które mają być sterowane.

Projektant systemu w zależności od rodzaju sterownika (standardowy PLC albo system dedykowany), ma do dyspozycji różne środki uruchomieniowe. Użycie typowego sterownika narzuca projektantowi gotowe środowisko uruchomieniowe, dostarczane wraz ze sterownikiem. Środowisko takie oferuje różne techniki programowania sterownika. W przypadku najprostszych, tzw. kompaktowych sterowników są to przynajmniej: język schematów funkcjonalnych (np. FBD) i drabinkowych (np. LAD). Sterowniki modułowe można programować także przy pomocy innych języków zgodnych standardem IEC1131-3 [1, 2] lub do niego zbliżonych. Inaczej jest w przypadku sterownika będącego dedykowanym, wbudowanym systemem mikroprocesorowym. Tworzenie oprogramowania dla takiego rozwiązania wiąże się z wykorzystaniem klasycznych narzędzi programistycznych. Są nimi kompilatory języka maszynowego (asemblerowego) lub wysokopoziomowego (głównie C), w połączeniu z gotowymi systemami operacyjnymi czasu rzeczywistego.

Projektowi dedykowanego sterownika mogą być narzucone pewne ograniczenia. Pojawiają się one zwłaszcza wtedy, gdy urządzenie docelowe, a co za tym idzie także sterownik, ma być produkowane seryjnie. Względy ekonomiczne nakazują wtedy poszukiwanie możliwie najtańszych rozwiązań konstrukcyjnych. W odniesieniu do systemu mikroprocesorowego oznacza to zalecenie użycia jak najtańszych jego komponentów i w możliwie minimalnej ilości. Składnikami systemu, których użycie w konstrukcji może być optymalizowane są między innymi procesor i pamięć operacyjna. Ich rodzaj i ilość wpływają na koszt wytworzenia systemu. Z drugiej strony procesor o określonej szybkości i pojemność pamięci operacyjnej decydują o mocy obliczeniowej całości. Szacowana moc obliczeniowa sterownika musi spełnić, wynikające z zadania sterowania, kryteria określające przykładowo: maksymalny czas reakcji systemu na zdarzenie, maksymalny czas cyklu, wielkość kodu programu i struktur danych potrzebnych do realizacji procesu sterowania. Spełnienie tych kryteriów przy określonej strukturze systemu mikroprocesorowego zależy od przyjętych rozwiązań programistycznych, w tym od zastosowanych języków i technik programowania.

W dalszej części artykułu zostaną przedstawione: proponowana metoda opisu sterowania sekwencyjnego, przykład jej zastosowania oraz wyniki testów efektywności różnych rozwiązań programistycznych tego zadania sterowania.

2. Opis sterowania sekwencyjnego

Przed przystąpieniem do programowania sterownika mikroprocesorowego należy sprecyzować postawione przed nim zadanie sterownia. W przypadku sterowania sekwencyjnego konieczne jest określenie zbioru X sygnałów wejściowych, będących binarnymi zmiennymi stanu procesu oraz zbioru Y sygnałów wyjściowych służących dwustanowemu sterowaniu obiektem. Należy również opisać pożądane zachowanie się układu sterownik-obiekt, które w formie algorytmu sterowania sekwencyjnego będzie zaimplementowane w oprogramowaniu sterownika.

Jedną z możliwych metod opisu sterowania sekwencyjnego jest metoda wykorzystująca automat skończony, opisana w [3]. Metoda ta zakłada, że sterownik można potraktować jako automat Moore'a, opisany równaniami:

$$s_{t+1} := \delta(s_t, x_t)$$

$$y_t := \lambda(s_t)$$

gdzie:

- δ - funkcja przejść, przyporządkowującą parze (s_t, x_t) w chwili t nowy stan s_{t+1} w chwili następnej t+1;
- λ - funkcja wyjść automatu;
- x_t - słowo wejściowe automatu;
- y_t - słowo wyjściowe automatu.

Przy takim założeniu:

- stany automatu-sterownika odpowiadają stanom pracy sterowanego obiektu;

- słowo wejściowe automatu zawiera zmienne binarne, charakteryzujące bieżący stan obiektu;
- słowo wyjściowe automatu jest zbiorem sygnałów sterujących obiektem oraz innych wielkości pomocniczych.

Zaprezentowane podejście do zagadnienia sterowania sekwencyjnego pozwala na sprawną implementację jego algorytmu w dowolnym języku programowania. Przed przystąpieniem do takiej implementacji należy jedynie zdefiniować zmienne programowe przechowujące kody stanu automatu oraz jego słowa wejściowe i wyjściowe. Przejścia pomiędzy stanami automatu realizuje się, jeżeli są spełnione odpowiednie wyrażenia logiczne - warunki przejść między stanami. Jeszcze prostsza jest realizacja funkcji wyjść automatu sterującego. Ponieważ w automacie typu Moore'a słowa wyjściowe zależą wyłącznie od stanu automatu, ich generowanie w wybranym języku programowania może polegać na sięganiu do uprzednio przygotowanej tablicy, indeksowanej kodem stanu automatu i zawierającej odpowiednie mu słowa wyjściowe.

Opisana metoda została przez autora z powodzeniem wdrożona w różnych aplikacjach: od sterowania małym przyrządem laboratoryjnym do nadzorowania pracy wielu urządzeń współpracujących ze sobą w linii technologicznej.

W celu usprawnienia procesu generacji kolejnych wersji automatu sterującego opracowano również odpowiedni tekstowy język opisu automatu sterującego, jego translator na różne docelowe procesory i języki programowania (zwany dalej translatorem JODA), a także środowisko graficzne (WinJoda) [4], umożliwiające projektowanie automatu sterującego w formie grafu przejść. Wykorzystanie wymienionych narzędzi może polegać albo na uzyskaniu za sprawą translatora odpowiednich struktur danych, stanowiących kompletny, zwięzły opis działania automatu sterującego, albo na poprzestaniu jedynie na tekstowym bądź graficznym opisie pracy automatu i "ręcznym" implementowaniu jego funkcji przejść i wyjść w wybranym języku programowania.

Takie różnorodne wykorzystanie tego samego opisu pracy sterownika musi skutkować zróżnicowanymi efektami pracy projektanta sterownika. Abstrahując od umiejętności programisty, otrzymuje się różniące się pod względem złożoności obliczeniowej oprogramowanie sterujące. Na wspomnianą złożoność składają się:

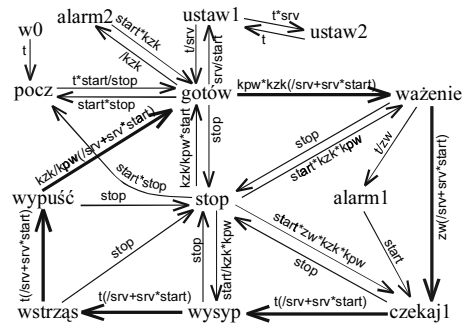
- złożoność czasowa, charakteryzująca wpływ wielkości realizowanego zadania na czas działania algorytmu;
- złożoność pamięciowa, informująca o zapotrzebowaniu na pamięć operacyjną (kod programu i struktury danych).

Moc obliczeniowa współczesnych komputerów pozwala w większość przypadków nie przejmować się różnicami w złożoności obliczeniowej różnych realizacji tego samego zadania programistycznego. Jednak w przypadku systemów wbudowanych, projektowanych ze względów ekonomicznych jako systemy z ograniczonymi zasobami, istotnym jest czy przy założonej strukturze systemu mikroprocesorowego programy odpowiedzialne za sterowanie będą działały dostatecznie szybko i czy wystarczy dla nich pamięci operacyjnej. Dlatego naturalnym jest poszukiwanie możliwie najszybszych i "najmniejszych pamięciowo" metod realizacji postawionego zadania sterowania.

Definicje złożoności obliczeniowych wykorzystują pojęcie tzw. rozmiaru zadania [5]. W przypadku algorytmu sterowania sekwencyjnego, opisanego jako automat Moore'a, rozmiarem tym może być: liczba stanów automatu, wielkość słowa wejściowego (ilość zmiennych binarnych), liczba krawędzi skierowanych w grafie przejść automatu. Złożoność obliczeniową danego zadania staramy się oszacować jako funkcję (np. wielomianową, wykładniczą, itp.), której argumentem jest wybrana wielkość charakteryzująca rozmiar tego zadania. Jej określenie wymaga szczegółowej analizy algorytmu lub wielu badań symulacyjnych programów realizujących zadania o różnych rozmiarach. Dlatego na potrzeby prezentowanego porównania ograniczono się do prostszych kryteriów. Określono je ogólnym mianem efektywności programu, rozumianej jako zajętość pamięci i charakterystyczne czasy jego realizacji dla przykładowego zadania testowego. Porównanie uzyskanych w ten sposób wartości liczbowych jest wystarczające do oceny szybkości pracy różnych realizacji tego samego zadania sterowania sekwencyjnego i optymalnego wyboru struktury systemu wbudowanego.

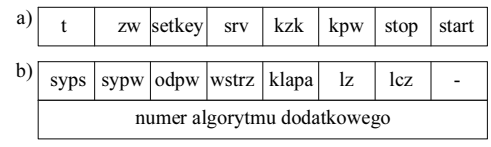
3. Przykładowa aplikacja i kryteria testów

Za przykład testowy posłużył zrealizowany wcześniej mikroprocesorowy sterownik wagi dozującej mieszanki paszowe [6]. Był on jednym z elementów modernizacji całego urządzenia. Sterownik skonstruowano na bazie mikrokontrolera z rodziny MCS51. Przy jego oprogramowaniu wykorzystano opis pracy sterownika jako automatu. Translator JODA przekształcił ten opis we fragment programu w języku maszynowym MCS51, opisujący w formie stablicowanych struktur danych zachowanie się automatu sterującego. Całe oprogramowanie również zostało wykonane w asemblerze. Rysunek 1 przedstawia graf przejść jednej z wersji automatu sterującego wykorzystywanego w testach. Skierowane krawędzie tego grafu symbolizują możliwe przejścia pomiędzy stanami automatu, a widoczne przy nich wyrażenia logiczne - warunki realizacji tych przejść. Podstawowy cykl sterowania sekwencyjnego został wyróżniony pogrubionymi krawędziami.



Rys. 1. Graf przykładowego automatu sterującego
Fig. 1. An exemplary controlling automata graph

Automat wykorzystuje jednobajtowe słowo wejściowe i dwubajtowe słowo wyjściowe obejmujące bajt wyjściowych zmiennych binarnych oraz bajt wartości liczbowej identyfikującej dodatkowe działania związane z przejściem automatu do danego stanu (rys. 2).



Rys. 2. Słowo wejściowe (a) i wyjściowe (b) przykładowego automatu sterującego
Fig. 2. Input (a) and output (b) words of the exemplary controlling automata

Wykorzystując koncepcję automatu sterującego przy programowaniu w języku C, konieczne jest rozróżnienie bieżącego stanu automatu i w zależności od niego - sprawdzenie odpowiednich warunków przejść do stanów następnych. Takiego rozróżnienia można dokonać używając szeregu instrukcji **if** albo instrukcji wyboru **switch**. Dlatego każdy z programów testowych w C został napisany w dwóch wersjach.

Oba środowiska uruchomieniowe oferują różne poziomy optymalizacji kodu kompilowanego programu. Stąd też przeprowadzono kompilacje i symulacje pracy każdego z programów przy różnych poziomach optymalizacji, w celu sprawdzenia jej wpływu na efektywność uzyskiwanego oprogramowania.

Ponieważ program WinJoda umożliwia wygenerowanie struktur opisujących automat sterujący także w języku C, przygotowano odpowiednie programy testowe. W ich treści użyto zarówno in-

strukcji **if** jak i **switch**. Kompilowano je i badano, podobnie jak poprzednie, przy różnych poziomach optymalizacji kodu.

Przeprowadzone badania miały za zadanie porównanie efektywności kodów programów realizujących sterowanie sekwencyjne. Przyjęto następujące założenia:

- kryteria porównawcze:
  - wielkość wygenerowanego kodu programu;
  - czasy pracy programu w części realizującej sterowanie sekwencyjne (czas podstawowego cyklu, czas obsługi stanu oczekiwania "gotów")
- procesor docelowy - standardowy mikrokontroler z rodziny MCS51;
- badane programy mają być ograniczone tylko do części realizującej sterowanie sekwencyjne;
- ewentualne zależności czasowe (minimalne czasy trwania wybranych stanów) są pomijane - chodzi o pomiar rzeczywistego obciążenia procesora;
- ponieważ wykorzystanie pamięci danych przez testowane programy jest niewielkie i prawie jednakowe, nie jest ono kryterium porównawczym.

Porównaniu podlegają:

- kod programu napisanego w asemblerze MCS51 z wykorzystaniem struktur generowanych przez program WinJoda;
- kody programów napisanych w języku C stosujące bezpośrednie wartościowanie wyrażeń logicznych warunkujących przejścia między stanami automatu;
- kody programów napisanych w języku C wykorzystujące stablicowane struktury wygenerowane przez program WinJoda i odpowiednie do nich techniki wartościowania wyrażeń.

Ponadto przy porównaniu programów w języku C należy dodatkowo uwzględnić:

- użycie różnych konstrukcji języka (**if** oraz **switch**);
- wykorzystanie różnych kompilatorów języka C dla procesora docelowego;

Jako środowiska badawcze zastosowano:

1. dla programu w asemblerze:
   - program WinJoda do przygotowania struktur opisu automatu w asemblerze MCS51;
   - metasembler C32;
   - symulator ekranowy AVSIM51.
2. dla programów w C:
   - program WinJoda do przygotowania struktur opisu automatu w C;
   - pakiety IDE firm A i B.

Użycie określeń firm "A" i "B", pod którymi kryją się dwaj wytwórcy, znani na rynku oprogramowania narzędziowego dla MCS51, ma na celu uniknięcie ewentualnej kryptoreklamy ich oprogramowania. Do testów wykorzystano ogólnie dostępne wersje demo, które w zakresie potrzebnym do badań okazały się w pełni funkcjonalne.

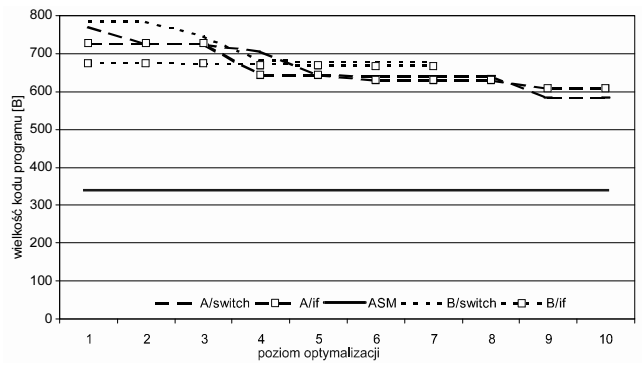
4. Omówienie wyników testów

Kod programu napisanego w asemblerze docelowego mikrokontrolera MCS51 zajmuje 344B. Szybkość jego pracy, mierzona liczbą cykli maszynowych zużywanych przez mikrokontroler wynosi T_{OC}=685c.m. Jest to minimalny czas cyklu obsługi automatu (zrealizowanie jednego cyklu pracy, tzn. jednego przejścia po wyróżnionych na rys. 1 krawędziach);

Podane wielkości kodu programu w asemblerze i czasu jego pracy posłużą jako punkty odniesienia dla testowanych programów w języku C. Syntetyczne wyniki testów w formie wykresów: wielkości kodu programu i czasu cyklu obsługi automatu, w funkcji poziomu optymalizacji, zamieszczono poniżej. Na rysunkach tych linią ciągłą zaznaczono wartości charakteryzujące program w asemblerze.

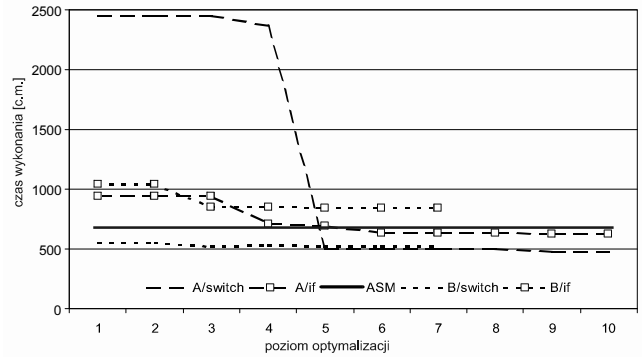
Analiza wykresu z rys. 3 prowadzi do następujących wniosków:

- Bezwzględnie najmniejszy kod można uzyskać w programie asemblerowym (zysk na zajętości pamięci może być nawet dwukrotny).
- Wielkość zoptymalizowanego kodu programu w C generowanego przez dany kompilator, słabo zależy od użytych konstrukcji językowych (różnice < 5%).
- Nieoptymalizowany kod programu w C z rozkazami **switch** jest większy niż programu realizującego te same funkcje przy pomocy instrukcji **if**.



Rys. 3. Rozmiar kodu programów przy różnych: poziomach optymalizacji, technikach i środowiskach programowania - klasyczne programy w C

Fig. 3. Programme code lengths in function of: optimisation level, programming methods and environments - classical C programming



Rys. 4. Czas cyklu obsługi automatu T_{OC} przy różnych poziomach optymalizacji - klasyczne programy w C

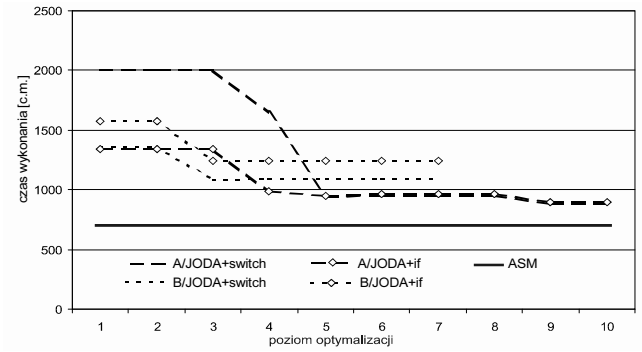
Fig. 4. Automata cycle service time T_{OC} in function of optimisation level - classical C programming

Przeprowadzone symulacje pracy programów pozwoliły ocenić ich szybkość i doprowadziły między innymi do następujących spostrzeżeń:

- Minimalny czas obsługi automatu oprogramowanego w asemblerze jest na ogół krótszy od czasów obsługi niezoptymalizowanych kodów programów w C.
- Zoptymalizowane kody programów w C mogą działać szybciej niż program napisany w asemblerze.

Testowano również efektywność programów napisanych w języku C, ale wykorzystujących stablicowane struktury opisu automatu sterującego. Wielkości kodu programu wahały się od 535B bez optymalizacji do 348B przy najwyższym poziomie optymalizacji. Są to wartości mniejsze niż przy klasycznych technikach programowania w C (rys. 3) i zbliżające się do wielkości kodu programu napisanego asemblerze.

Wyniki testów przedstawione na rys. 5 wskazują na wolniejszą pracę programów napisanych w języku C z wykorzystaniem tablic opisu automatu w porównaniu z programem w asemblerze, nawet przy maksymalnej ich optymalizacji. Programy te ustępują również szybkością programom realizującym sterowanie sekwencyjne klasycznymi technikami programowania w języku C.



Rys. 5. Czas cyklu obsługi automatu T_{OC} przy różnych poziomach optymalizacji - wykorzystanie tablicowych struktur opisu automatu
Fig. 5. Automata cycle service time T_{OC} in function of optimisation level - usage of tabulated automata description structures

5. Podsumowanie

Zaprezentowane wyniki testów różnych wersji programu realizującego sterowanie sekwencyjne, wskazują na zależność efektywności oprogramowania sterownika od użytych techniki i narzędzi programistycznych. W testowanym przykładzie największe "pamięciowo" programy są przeszło dwa razy większe od rozwiązań optymalnych. Jeszcze większy przedział zmienności można zauważyć w przypadku czasów wykonania programów - w jednym z przypadków czas cyklu obsługi automatu (T_{OC}) zmienia się od 477 do 2448 cykli maszynowych.

Dostrzeżone zróżnicowanie wielkości kodu i czasu pracy programów stwarza pewne pole manewru przy doborze optymalnego rozwiązania programistycznego w zależności od przyjętych kryteriów optymalizacji. Dążąc do minimalnego zapotrzebowania systemu mikroprocesorowego wbudowanego sterownika na pamięć operacyjną, warto rozważyć zaproponowaną w rozdziale 2 technikę kodowania pracy automatu przy pomocy odpowiednio przygotowanych tablic. Technika ta ma również taką zaletę, iż bardzo ułatwia aktualizację algorytmu sterowania sekwencyjnego, która sprowadza się wtedy tylko do modyfikacji tablic kodujących automat. W niektórych przypadkach aktualizacji takiej można dokonać nawet bez wstrzymywania cyklu pracy sterownika, wykorzystując tzw. stan spoczynkowy automatu sterującego. Jeżeli głównym kryterium optymalizacji systemu mikroprocesorowego ma być czas, a co za tym idzie szybkość procesora, wtedy klasyczne rozwiązania programistyczne mogą okazać się sprawniejsze, ale pod warunkiem maksymalnego wykorzystania możliwości optymalizacyjnych użytych kompilatorów języka C. Aktualizacja oprogramowania przygotowanego w ten sposób jest jednak trudniejsza, bardziej czasochłonna i wymaga przynajmniej wstrzymania pracy sterownika.

Wyniki tego porównania mogą być pewnymi wskazówkami dla projektanta-programisty systemu wbudowanego.

Prezentowane badania są finansowane z pracy statutowej S/WE/1/06.

6. Literatura

[1] T. Legierski, J. Kasprzyk, J. Wyrwał, J.Hajda: Programowanie sterowników PLC. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 1998.
[2] IEC International Standard 61131-3. Programmable Controllers, Part 3: Programming Languages. Geneva: International Electrotechnical Commision, 1992.
[3] L. Grodzki: Automatas in programming of microprocessor-based binary process controllers. IV MMAR'97, Międzyzdroje.
[4] L. Grodzki: Graficzne środowisko projektowe do programowania niestandardowych sterowników. XII KKADP, Zakopane 2000.
[5] A.V. Aho, J.E. Hopcroft, J.D. Ullman: Projektowanie i analiza algorytmów komputerowych. PWN, Warszawa 1983.
[6] L. Grodzki: Modernizacja układu sterowania wagopakarką. VII KK Automatyzacja i eksploatacja systemów sterowania, Gdynia 1999.

Artykuł recenzowany