Michał KOBYLECKI[1], Dariusz KANIA[2]

Politechnika Śląska, Instytut Elektroniki (1), Politechnika Śląska, Instytut Elektroniki (2)

Dwutaktowa realizacja sterowania bitowego

Streszczenie. *W artykule przedstawiono oryginalną metodę sprzętowej realizacji algorytmów sterowania bitowego zgodną z normą IEC61131-3. Zaproponowane przekształcenia, oparte o rachunek zbiorów oraz grafy, pozwalają na translację oryginalnej postaci programu sterowania do postaci w pełni zgodnej z oryginałem, dającej się jednak sprzętowo realizować za pomocą architektury dwutaktowej. Zastosowanie opisanej metody umożliwia wydajną sprzętową realizację sterowania bitowego, przedstawionego w ustandaryzowanym języku programowania LD, w układach FPGA.*

Abstract*. In the paper there is presented a procedure for the implementation of control algorithms for hardware-bit compatible with the standard IEC61131-3. Described transformation based on the sets calculus and graphs, allow for translation of the original form of the control program to the form in full compliance with the original, giving the architecture represented by two tick. The use of this procedure enables the efficient implementation of the control bits in the FPGA using a standardized programming language LD (**Double-tick realization of binary control program**).*

Słowa kluczowe: logika programowalna, realizacja równoległa, sterownik programowalny, operacje bitowe
Keywords: programmable logic, parallel implementation, programmable controller, bit operations

doi:10.12915/pe.2014.09.56

Wstęp

Sterowniki programowalne (PLC) są w dzisiejszych czasach najbardziej rozpowszechnionymi urządzeniami służącymi do sterowania maszynami oraz procesami przemysłowymi. Niski koszt, prostota obsługi oraz programowania pozwala stosować PLC praktycznie w każdej dziedzinie przemysłu, poczynając od najprostszych aplikacji sterowania pojedynczymi urządzeniami, kończąc zaś na sterowaniu złożonymi procesami technologicznymi. Realizacja tych ostatnich wymaga zazwyczaj dużych mocy obliczeniowych oraz jak najkrótszego czasu wykonania programu. Z uwagi na coraz większe wymagania czasowe, problematyczne staje się wykorzystywanie klasycznej architektury opartej o układy mikroprocesorowe [10]. Ich szeregowo-cykliczny sposób wykonywania zadań prowadzi, w przypadku złożonych algorytmów sterowania, do znaczącego wydłużenia czasu realizacji programu. Alternatywą wydaje się być zastosowanie układów FPGA, które umożliwiają w pełni sprzętową realizację programu sterowania, a co za tym idzie znacząco przyspieszają jego realizację [1,2].

Wykonywanie programu przez klasyczny sterownik polega na cyklicznym powtarzaniu następujących operacji:

- odczyt stanu wejść i jego zapis do specjalnie wydzielonego obszaru pamięci zwanego PII (Process Image Input),
- wykonanie algorytmu sterowania,
- zapis stanu wyjść odczytanych ze specjalnie wydzielonego obszaru pamięci zwanej PIO (Process Image Output),
- wykonanie procedur systemowych sterownika (autodiagnostyka, obsługa komunikacji itp.).

Wykorzystanie układów logiki programowalnej pozwala na znaczące skrócenie czasu wykonania całego cyklu sterowania poprzez sprzętową realizację algorytmu sterowania, dopasowaną do możliwości struktur programowalnych. Realizacja szeregowo-cykliczna, typowa dla systemu mikroprocesorowego, polega na cyklicznym wykonywaniu operacji „krok po kroku”. Wykorzystując układy FPGA algorytm sterowania można wykonać znacznie szybciej. Tego typu realizację można określić mianem realizacji "sprzętowo-równoległej" [8], która prowadzi do znacznego skrócenia czas reakcji na zmiany sygnałów obiektowych.

Celem artykułu jest przedstawienie oryginalnej koncepcji sprzętowej realizacji programu sterowania, dopasowanej do zasobów sprzętowych współczesnych układów programowalnych. Proponowana metoda realizacji sprzętowej zachowuje pełną zgodność wykonania algorytmu sterowania z jego klasycznym, programowym odpowiednikiem. Zachowuje, więc znane inżynierowi charakterystyczne zależności pomiędzy liniami opisywanego programu, a sekwencją operacji na wyjściach układu, uzyskiwaną w klasycznej realizacji programu w sterowniku przemysłowym.

W dalszej części artykułu przedstawiony zostanie sposób zamiany programu sterowania opisanego w ustandaryzowanym normą IEC61131-3 języku LD (Ladder Diagram) [4] na postać „sprzętową”, która pozwala na realizację programu sterowania w czasie dwóch taktów sygnału zegarowego, bez względu na "długość" opisu algorytmu sterowania. Całość rozważań jest zobrazowana na prostym przykładzie, który na wstępie stanowi typowy opis sterowania bitowego, a po licznych modyfikacjach przyjmuje postać, która może być bezpośrednio odzwierciedlona w sprzęcie, tzn. strukturze typowego układu programowalnego.

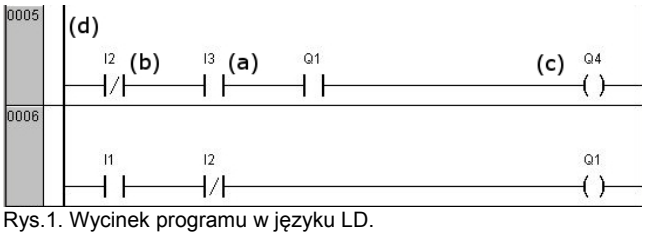
Wprowadzenie do języka LD

Ladder Diagram (LD) jest ujętym w normie IEC61131-3 jednym z najstarszych i najpopularniejszych języków programowania sterowników przemysłowych PLC [3,4,5]. Wywodzi się on bezpośrednio z układów automatyki realizowanych na elementach stykowych typu NO (styk normalnie rozwarty) oraz NC (styk normalnie zwarty). W układach tych poprzez odpowiednio realizowane połączenia w/w elementów można było uzyskać podstawowe elementy logiczne takie jak AND, OR, oraz NOT, co pozwalało na budowę bardziej złożonych układów sterowania.

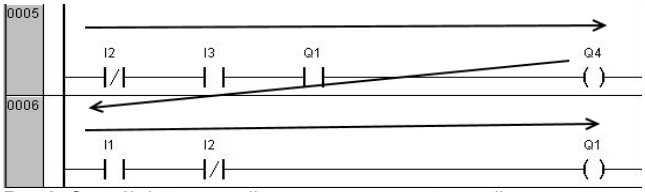
Szybki rozwój elektroniki opartej o mikroprocesory pozwolił przenieść ciężar wykonywania operacji logicznych, z wolnych i stosunkowo dużych układów stykowych, na układ zawierający swobodnie programowalny mikroprocesor. Pozwoliło to znacząco obniżyć koszt przemysłowych układów sterowania, rozszerzając je dodatkowo o możliwości, jakie niesie ze sobą układ mikroprocesora (timery, pamięć, liczniki, magistrale komunikacyjne). Jedyną niedogodnością była konieczność znajomości sposobów i języków programowania mikroprocesorów. Doprowadziło to do opracowania przyjaznych dla użytkownika języków zorientowanych na sterowniki przemysłowe takich jak Ladder Diagram (LD), Function Block Diagram (FBD), Instruction List (IL). Języki te są w pełni równoważne, jednak podobieństwo składni do układów stykowych spowodowało, że LD stał się najpopularniejszym językiem wśród programistów-automatyków [6, 7].

Na rysunku 1 przedstawiono wycinek programu napisanego w języku LD.

Symbole (a) reprezentują zmienne typu NO, natomiast (b) zmienne typu NC. Symbole (c) są zmiennymi wynikowymi, natomiast linia po lewej stronie (d) jest źródłem sygnału. Interpretacja programu polega na analizie przebiegu sygnału od źródła do elementu wynikowego linijka po linijce (Rys.2).



Rys.1. Wycinek programu w języku LD.



Rys.2. Sposób interpretacji programu przez sterownik.

Przedstawiony na rysunku 2 sposób interpretacji programu [5], oraz zasada działania sterownika programowalnego przedstawiona we wstępie niniejszego artykułu pozwala programiście na pewne „sztuczki" programowe takie jak wielokrotne nadpisywanie zmiennej wynikowej podczas „obliczania" sterowania (zmienna wynikowa traktowana jest jak zmienna tymczasowa, aż do ostatecznego jej nadpisania), oraz zapamiętywanie ostatniej wartości zmiennej (poprzez korzystanie z niej zanim zostanie nadpisana).

Zrównoleglenie wykonania programu sterowania

W klasycznym sterowniku programowalnym PLC realizacja całego programu sterowania wymaga wykonania pewnej liczby rozkazów mikroprocesora odzwierciedlającej opisany przez programistę algorytm. Można zauważyć, że czas wykonania całego cyklu sterowania jest więc bezpośrednio związany ze złożonością napisanego programu. Im realizowany opis programu sterowania jest dłuższy, tym cykl sterownika trwa dłużej. Zakładając, że pozostałe operacje przeprowadzane przez sterownik (odczyt PII, zapis PIO, procedury systemowe) są niezależne od programisty, można stwierdzić, że główny wpływ na czas cyklu sterownika ma "długość" programu.

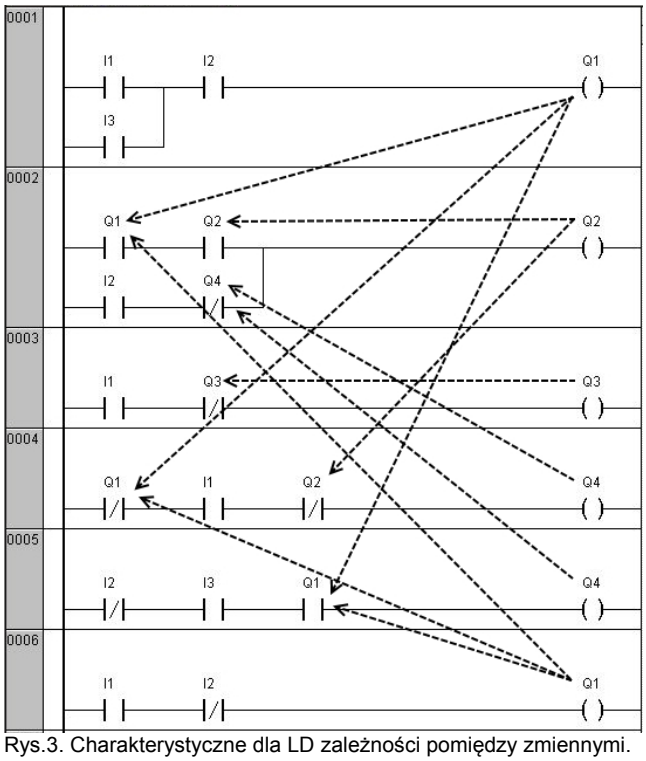
W celu przyspieszenia wykonywania programu można zrównoleglić wykonywanie odpowiednich jego partii poprzez wykorzystanie procesorów wielordzeniowych lub układu wielu procesorów. Nadal jednak czas cyklu sterownika pozostaje zależny od złożoności programu sterowania, a koszty związane z zaimplementowaniem wydajnych procesorów znacząco podnoszą koszty całego systemu.

Zastosowanie układów FPGA [11] oraz odejście od architektury sterownika, w której główną rolę odgrywa mikroprocesor, pozwala na uniezależnienie czasu wykonania cyklu sterowania od złożoności napisanego programu. Dzieje się to kosztem zasobów sprzętowych, jednak na dzień dzisiejszy układy logiki programowalnej są bardzo "pojemne", stąd nie stanowi to dużego problemu technicznego, a skutkuje dużym wzrostem szybkości przetwarzania programu.

Proponowana metoda sprzętowej realizacji programu sterowania umożliwia zrównoleglenie dowolnego programu sterowania logicznego opisanego w LD, zawierającego podstawowe elementy umożliwiające wykonywanie operacji bitowych, takich jak styki NO, NC, cewki wyjściowe, negacje oraz przerzutniki.

Specyficzne własności programów opisanych w języku LD

Rysunek 3 przedstawia przykładowy program sterowania opisany w języku LD wraz z zaznaczonymi specyficznymi dla tego języka zależnościami pomiędzy zmiennymi [9].



Rys.3. Charakterystyczne dla LD zależności pomiędzy zmiennymi.

Można zauważyć, że warunki logiczne zawarte w linii 2, 4 i 5 są bezpośrednio zależne od wykonania linii 1. Dodatkowo do poprawnego wyznaczenia wyniku linii 4 wymagane jest wcześniejsze wykonanie linii 2. Podobnie wygląda sprawa z liniami 2 oraz 3 z tą różnicą, że do ich wykonania niezbędna jest informacja z wcześniejszego cyklu programu. Ostatecznie można zauważyć, że linia 4 nie wpływa w żaden sposób na cząstkowe, ani całościowe wyniki obliczeń, a jedynie na czas wykonania pętli programu (jest zbędna), natomiast linia 1, mimo iż nadpisuje tą samą zmienną co linia 6, wpływa na cząstkowe wyniki obliczeń i jest niezbędna do prawidłowego działania programu sterowania.

Powyższa, zgrubna analiza przykładowego programu wskazuje na możliwość jego modyfikacji, zachowującej behawioralną zgodność sygnałów wyjściowych układu sterowania. Poprzez eliminację sekwencyjności opisu, zaproponowana w dalszej części modyfikacja programu stworzy możliwość jego realizacji sprzętowo-równoległej w strukturze FPGA.

Graf zależności liniowych LP

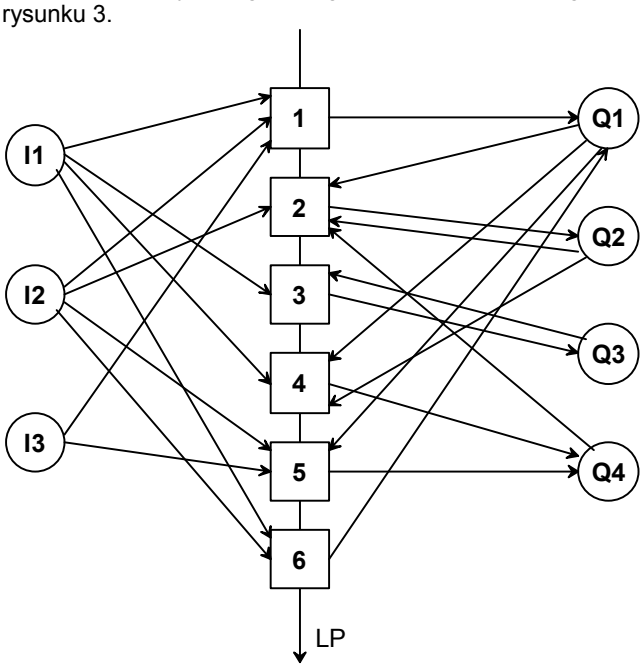
Zależności czasowe w programie przedstawionym na rysunkach 1, 2 i 3 można zobrazować za pomocą zaproponowanego grafu zależności liniowych. Graf ten tworzony jest w oparciu o oś pionową LP (Linii Programu), odzwierciedlającą kolejne segmenty programu. Zwrot tej osi, skierowany w dół, odzwierciedla sekwencję realizacji segmentów w klasycznej realizacji szeregowo-cyklicznej.

Na osi LP znajdują się po kolei numery wszystkich segmentów programu (linii programu).

Po lewej stronie osi LP umieszczono wszystkie zmienne niezależne (I1, I2, I3), natomiast po jej prawej stronie, znalazły się wszystkie zmienne zależne (Q1, Q2, Q3, Q4).

W ogólnym przypadku zmienne niezależne są to wszystkie zmienne użyte w programie, których zmiana nie jest uzależniona od wykonania programu (wejściowe sygnały zewnętrzne, stałe itp.). Zmienne zależne są to wszystkie zmienne użyte w programie, których zmiana może być spowodowana wykonaniem części lub całości programu (wyjściowe sygnały zewnętrzne, zmienne tymczasowe itp.).

Na rysunku 4 przedstawiono graf zależności liniowych LP dla rozpatrywanego programu przedstawionego na rysunku 3.



Rys.4. Graf zależności liniowych LP dla programu z rysunku 3

Strzałki zwrócone od zmiennych w kierunku osi LP informują, że dana zmienna (I1, I2, I3, Q1, Q2, Q3, Q4) jest użyta we wskazywanej linii (numer linii w kwadracie na osi LP), natomiast strzałki zwrócone od osi w kierunku zmiennych informują, że wynik obliczeń z danej linii programu jest zapisywany do wskazywanej zmiennej (Q1, Q2, Q3, Q4).

Algorytm rozdziału segmentów programu na elementy niezależne

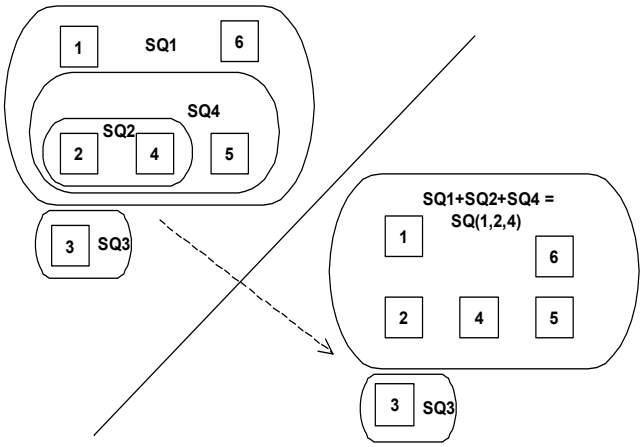
Na podstawie struktury grafu LP można wyznaczyć zbiory segmentów programu (linii programu), dla każdej zmiennej zależnej Qi, takie, że

$$SQi = \{n : n \to Qi \vee Qi \to n\} \quad i = 1,2,3,4 \quad n = 1,2,3,...,6$$

Analiza grafu zależności liniowych LP rozpatrywanego programu prowadzi do utworzenia następujących zbiorów segmentów skojarzonych z poszczególnymi zmiennymi Qi, i=1,2,3,4:

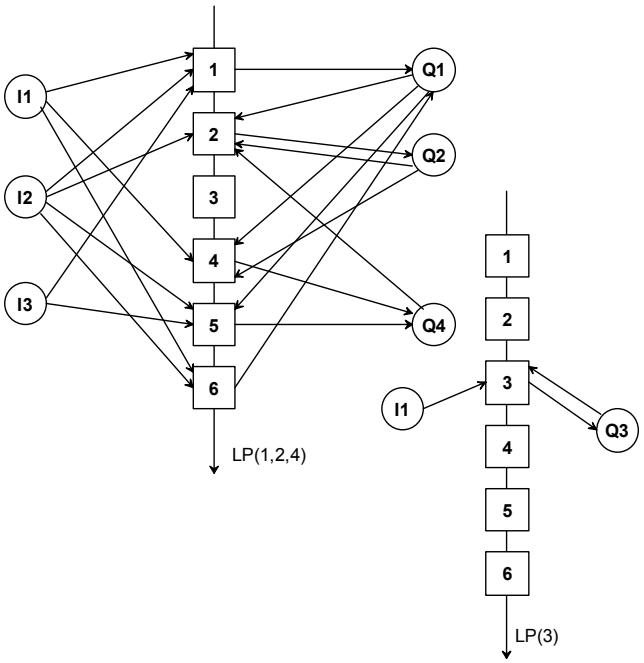
$$SQ1 = \{1,2,4,5,6\}$$
$$SQ2 = \{2,4\}$$
$$SQ3 = \{3\}$$
$$SQ4 = \{2,4,5\}$$

Po utworzeniu zbiorów SQi, i=1,2,3,4, rozpoczynana jest procedura wyszukiwania zbiorów rozłącznych. Kolejno rozpatrywane są wszystkie pary zbiorów w celu sprawdzenia, czy posiadają one części wspólne. Jeżeli przykładowo zbiory SQx oraz SQy mają część wspólną, to tworzony jest zbiór SQ(x,y) = SQx ∪ SQy. Jeżeli część wspólna nie występuje, to rozpatrywana jest kolejna para zbiorów. Wynikiem procedury wyszukiwania zbiorów rozłącznych są zbiory, które nie posiadają części wspólnych. Na rysunku 5 przedstawiono zbiory przed i po zastosowaniu procedury wyszukiwania zbiorów rozłącznych, czyli segmentów programu, które mogą być wykonywane niezależnie.



Rys.5. Ilustracja wyników przeprowadzenia algorytmu rozdziału segmentów programu na segmenty niezależne

Po dokonaniu podziału segmentów na fragmenty niezależne możliwe staje się przekształcenie grafu LP na dwa rozłączne podgrafy LP(1,2,4) oraz LP(3), które przedstawiono na rysunku 6.



Rys.6. Grafy LP(1,2,4) i LP(3) - wynik podziału grafu LP

Analiza przykładowego programu pozwala stwierdzić, że zawiera on dwa w pełni rozłączne, tzn. niezależne fragmenty opisu programu sterowania.

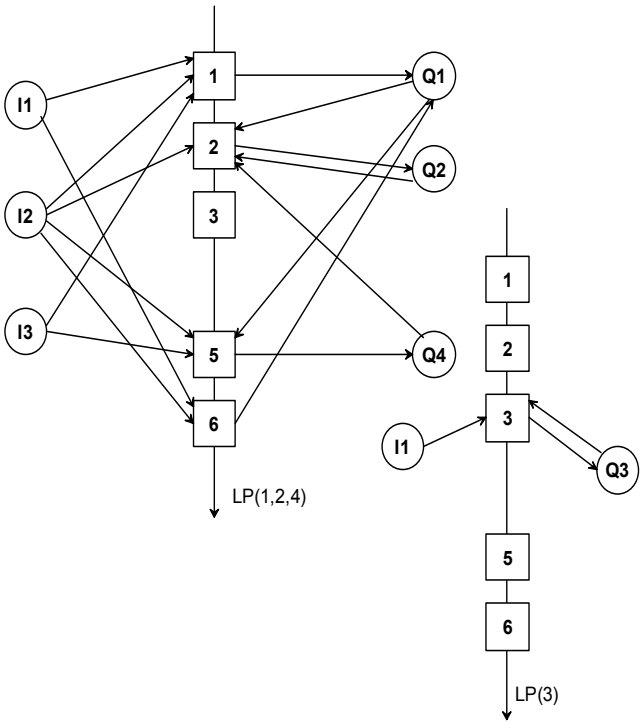
Optymalizacja programu poprzez eliminację segmentów wyjściowo nieistotnych

Sposób realizacji programu opisanego w języku LD stwarza możliwość występowania w opisie algorytmu sterowania fragmentów programu, które nie wpływają na sygnały wyjściowe. Stwarza to możliwość algorytmicznej optymalizacji programu, poprzez wyszukiwanie nieistotnych z punktu widzenia sygnałów obiektowych segmentów programu. Rozważmy analizowany program, czy nie znajdują się w nim segmenty, które są wyjściowo nieistotne i tym samym mogą zostać wyeliminowane z opisu programu. Zazwyczaj są to segmenty, których wyniki obliczeń zapisywane są do zmiennych, które następnie zostają nadpisywane w innych segmentach. W rozpatrywanym przykładzie takim segmentem jest segment 4, w którym wypracowywana jest wartość zmiennej Q4, modyfikowanej w segmencie kolejnym.

W ujęciu ogólnym, segment Si jest segmentem nieistotnym z punktu widzenia klasycznej realizacji programu, gdy istnieje segment Sj (j>i) wskazujący tę samą zmienną zależną Qk co segment Si, przy czym Qk nie wskazuje na żaden segment Sm, taki, że i<m≤j.

Jeżeli spełniony jest powyższy warunek, możliwe staje się uproszczenie programu, poprzez eliminację segmentów wyjściowo nieistotnych.

Rysunek 7 przedstawia grafy LP(1,2,4) i LP(3) po zoptymalizowaniu algorytmu sterowania (po usunięciu z programu segmentu 4).



Rys.7. Grafy LP(1,2,4) i LP(3) po eliminacji segmentu 4

Eliminacja sekwencyjnych uzależnień pomiędzy zmiennymi

Otrzymany program, podzielony na dwa rozłączne podzbiory segmentów, nie jest jeszcze gotowy do realizacji równoległej. Dzieje się tak ze względu na zależności występujące pomiędzy zmiennymi w poszczególnych segmentach. Przykładowo na wartość zmiennej Q1 wpływają warunki logiczne zawarte w dwóch segmentach: 1 i 6, stąd trudno te segmenty realizować równolegle. W ogólnym przypadku problem równoległej realizacji poszczególnych segmentów wynika z zależności czasowych pomiędzy zmiennymi, które mogą być wielokrotnie nadpisywane lub wzajemnie uzależnione ze względu na sekwencyjną formę opisu programu.

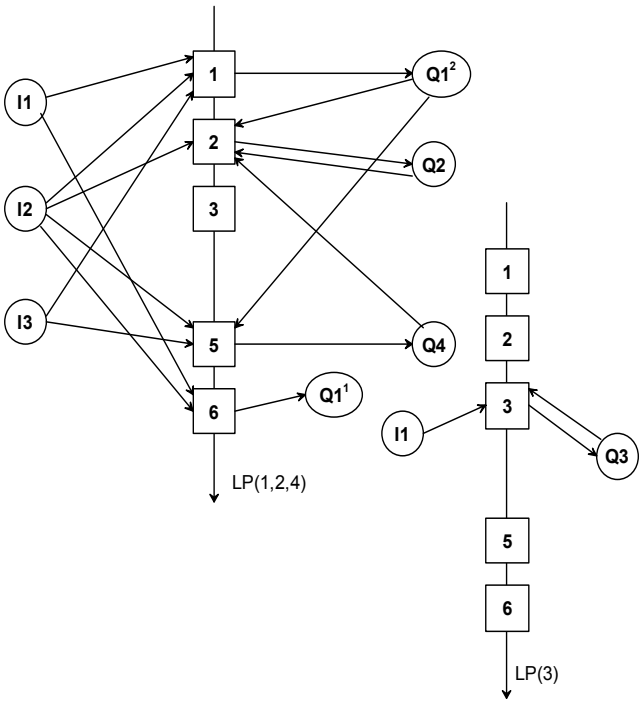
Zmienną Qk nazywamy wielokrotnie nadpisywaną, gdy istnieją przynajmniej dwa segmenty ustawiające tą zmienną.

W analizowanym przykładzie (Rys.7) występuje tylko jedna zmienna wielokrotnie nadpisywana. Jest nią zmienna Q1 ustawiana przez segment S1 oraz S6 (S1 → Q1 i S6 → Q1).

Do rozdzielenia zmiennej wielokrotnie nadpisywanej Qi na wiele zmiennych, służy algorytm rozdziału zmiennych wielokrotnie nadpisywanych, składający się z następujących etapów:

a) poszukiwanie segmentów związanych z rozpatrywaną zmienną wielokrotnie nadpisywaną Qi (analiza od ostatniego segmentu, wskaźnik k = 1).
b) po znalezieniu segmentu ustawiającego rozpatrywaną zmienną wielokrotnie nadpisywaną Qi przypisywana jest jej nowa zmienna Qi^k wraz z wszystkimi jej zależnościami poniżej tego segmentu (wszystkie strzałki na grafie LP „od" i „w kierunku" rozpatrywanej zmiennej Qi, pochodzące od segmentów o wyższych numerach od aktualnie rozpatrywanego segmentu), pozostałe zależności wpływają na wartość zmiennej Qi, wskaźnik k zwiększa się o 1 (k = k + 1).
c) punkt (b) powtarza się do momentu, w którym zmienna Qi nie będzie już ustawiana przez żaden segment Sj. Jeżeli nie istnieją wtedy pomiędzy nią a pozostałymi segmentami żadne zależności, zostaje usunięta. W przeciwnym przypadku staje się ona zmienną niezależną i "przechodzi" na lewą stronę osi LP.

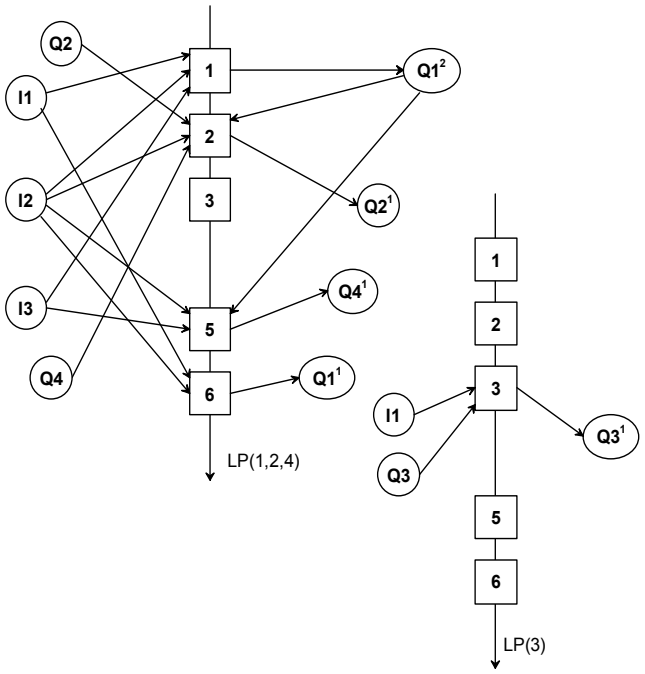
Rysunek 8 przedstawia graf LP opisujący wynik rozszczepienia zmiennych wielokrotnie nadpisywanych na wiele zmiennych.



Rys.8. Wynik rozszczepienia zmiennych wielokrotnie nadpisywanych

Następnym krokiem dostosowywania programu do realizacji równoległej jest identyfikacja tzw. zmiennych zależnych wstecz.

Zmienną Qi nazywamy zależną wstecz wtedy, gdy istnieją przynajmniej dwa segmenty Sj i Sk takie, że Sj ustawia Qi, a Qi wskazuje na Sk, przy czym j ≥ k.



Rys.9. Wynik rozłączania zmiennych i optymalizacji programu w postaci grafów LP(1,2,4) i LP(3)

Zgodnie z powyższą definicją, w analizowanym przykładzie występują trzy zmienne zależne wstecz, którymi są zmienne Q2, Q3, Q4.

Eliminacja w programie zależności wstecz prowadzi do dalszej modyfikacji opisu programu sterowania. W miejsce zidentyfikowanych zmiennych zależnych wstecz Qi (i=2,3,4), wstawiane są zmienne Qi^1 ustawiane przez te same warunki logiczne zawarte w odpowiednich segmentach co Qi. Pierwotne zmienne Qi przenoszone są na stronę zmiennych niezależnych wraz z pozostałymi "grafowymi" powiązaniami.

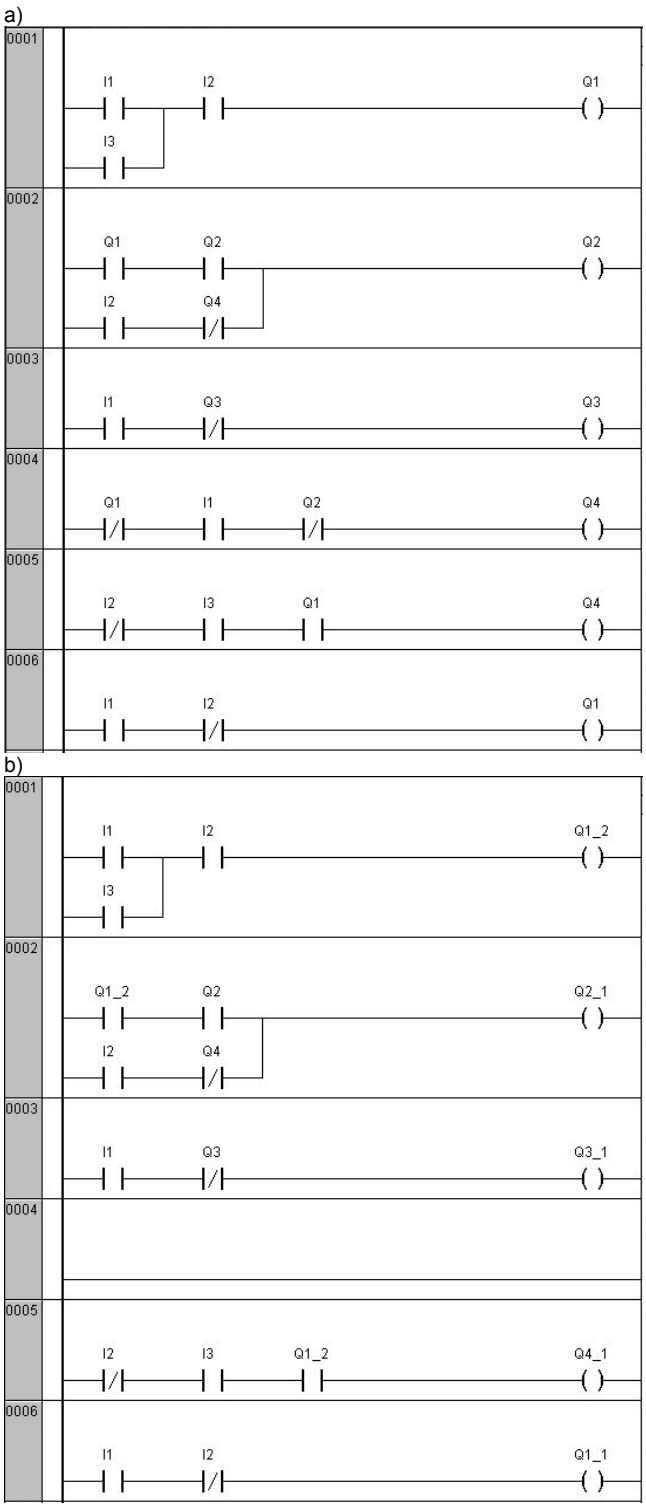
Rysunek 9 przedstawia wynik przekształcenia programu sterowania w postaci grafów LP(1,2,4) i LP(3) powstałych po eliminacji występujących w programie sekwencyjnych uzależnień pomiędzy wyszukanymi segmentami. Tego typu postać grafu umożliwia sprzętowo-równoległą realizację odpowiadającego jej programu sterowania.

Dwutaktowa realizacja programu

Postać grafu przedstawiona na rysunku 9 umożliwia przekształcenie pierwotnego programu (Rys. 10a) do równoważnej postaci (Rys. 10b) opisującej identyczne funkcje, ale możliwej do realizacji w zaproponowanej "dwutaktowej" sprzętowej architekturze układu. Pierwotna postać programu oraz wynik jego modyfikacji, która może być wykonana w sposób automatyczny za pomocą stosownego oprogramowania, są przedstawione na rysunku 10.

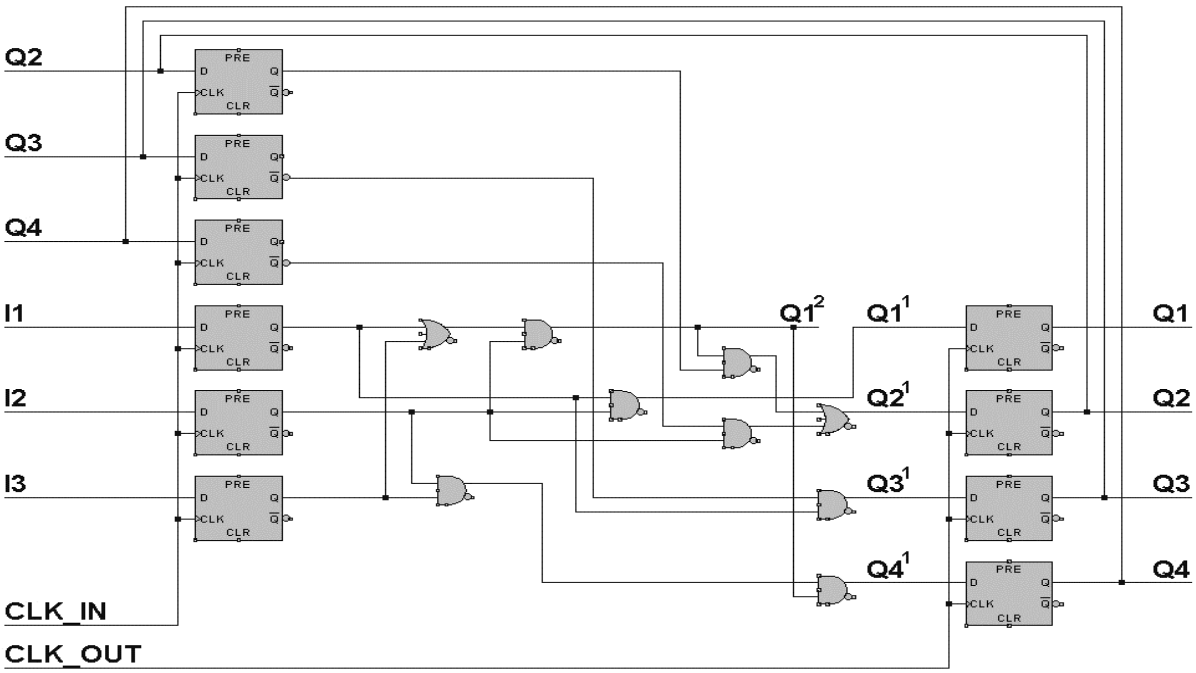
Na rysunku 11 przedstawiona została struktura układu realizującego program sterowania, stanowiąca sprzętowe odzwierciedlenie opisu programu sterowania z rysunku 10b. Należy podkreślić, że uzyskany układ sterowania zapewnia behawioralną zgodność z klasyczną realizacją szeregowo-cykliczną, charakterystyczną dla wszelkiego typu rozwiązań mikroprocesorowych, opisaną za pomocą postaci programu przedstawionej na rysunku 10a.

W celu uzyskania pełnej zgodności ze sposobem działania sterowników programowalnych zastosowano synchroniczne przerzutniki typu D na wejściach i wyjściach układu. Pozwala to na cykliczne odczytywanie wejść i zapisywanie wyjść.



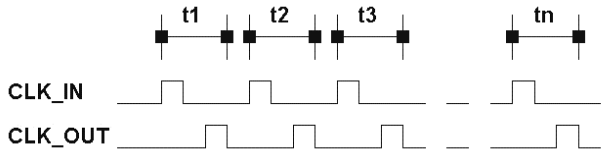
Rys.10. (a) program oryginalny (b) program po modyfikacjach ukierunkowanych na efektywną czasowo realizację sprzętową

Okazuje się, że cały proces wypracowywania sygnałów wyjściowych opracowanego sterownika bitowego może trwać tylko dwa okresy sygnału zegarowego.



Rys.11. Dwutaktowa realizacja programu sterowania

Sygnały wejściowe razem z sygnałami sprzężeń zwrotnych są zatrzaskiwane w przerzutnikach D synchronizowanych sygnałem CLK_IN. Po zatrzaśnięciu sygnałów wyjściowych w kilkuwarstwowym układzie kombinacyjnym wypracowywane są sygnały wzbudzeń przerzutników wyjściowych synchronizowanych sygnałem COK_OUT, przechowujących sygnały wyjściowe sterownika. W zaproponowanym rozwiązaniu czas wykonania jednej pętli programu jest więc niezależny od jego wielkości (Rys. 12).



Rys.12. Sygnały zegarowe układu sterowania o architekturze dwutaktowej

Podsumowanie

Zaprezentowana w artykule koncepcja realizacji programu sterowania stanowi podstawy teoretyczne metody automatycznej konwersji programu sterowania opisanego w postaci LD do opisu układu w postaci języka opisu sprzętu (Verilog, VHDL). Przedstawiona na prostym przykładzie metoda przekształcania programu sterowania, eliminująca sekwencyjność uzależnień poszczególnych zmiennych, stwarza możliwości bardzo efektywnej pod względem czasowym implementacji układowej sterownika bitowego w strukturze FPGA. Proponowane udoskonalenie znanej z literatury sprzętowej koncepcji realizacji programu sterowania dobrze wpisuje się w możliwości dostępnych architektur układów FPGA typu tablicowego. Zasoby sprzętowe produkowanych obecnie struktur FPGA stwarzają możliwość udoskonalenia realizacji nie tylko sterowników bitowych. Przykładowo można myśleć o sprzętowo-równoległej realizacji złożonych operacji arytmetycznych. Między innymi możliwe jest zwielokrotnienie złożonych bloków arytmetycznych w celu równoległego wykonywania operacji matematycznych.

Tego typu zagadnienia będą stanowić obszar dalszych prac, których celem jest opracowanie efektywnej pod względem czasu globalnie asynchronicznej lokalnie synchronicznej architektury sterownika PLC realizowanego w układzie FPGA.

LITERATURA

[1] J. Mocha, D. Kania, Metoda sprzętowej realizacji programu LD z wykorzystaniem układów FPGA, Pomiary Automatyka Kontrola, nr 1, 2012 ss. 88-92

[2] J. Mocha, D. Kania, Sprzętowa realizacja programu sterowania w strukturach FPGA, Przegląd Elektrotechniczny, R.88, Nr 12a, 2012, ss. 95-100

[3] IEC 61131-1 Programmable controllers – Part 1: General information

[4] IEC 61131-3 Programmable controllers – Part 3: Programming languages

[5] M. Chmiel, E. Hrynkiewicz, M. Muszyński „The way of ladder diagram analysis for small compact programmable controller", KORUS 2002, ss. 169-173

[6] Cieniak I, Polski rynek sterowników programowalnych PLC, Control Engineering Polska, lipiec/sierpień 2011

[7] Pietrusewicz K., Największe badanie polskiego rynku sterowników PLC, Control Engineering Polska, luty 2007

[8] D. Kania, Wielokontekstowy sterownik programowalny przyszłości wykorzystujący układy programowalne pSoC, Pomiary, Automatyka, Robotyka nr 1, 2006, ss. 5-12

[9] Miyazawa I, Nagao T , Fukagawa M, Itoh Y, Mizuya T, Implementation Of Ladder diagram for Programmable controller using FPGA, Electronics Lletter, Vol. 34, No. 8, pp 739-741, 1998

[10] Y. Itoh, I. Miyazawa and T. Sekiguchi, Study on execution time of ladder diagram in programmable controller, Proc. of IECON'98, pp. 149-1 54, 1998

[11] Daoshan Du, Xiaodong Xu, Kazuo Yamazaki, A study on the generation of silicon-based hardware Plc by means of the direct conversion of the ladder diagram to circuit design language, Int J Adv Manuf Technol (2010) 49:615–626

***Autorzy**: mgr.inż. Michał Kobylecki, Politechnika Śląska, Doktorant w Instytucie Elektroniki, ul. Akademicka 16, 44-100 Gliwice, E-mail: m.kobylecki.eu@gmail.com. prof. dr hab. inż. Dariusz Kania, Politechnika Śląska, Instytut Elektroniki, ul. Akademicka 16, 44-100 Gliwice, E-mail: dkania@polsl.pl*