Wielokontekstowy sterownik programowalny przyszłości wykorzystujący układy programowalne pSoC

Dariusz Kania*

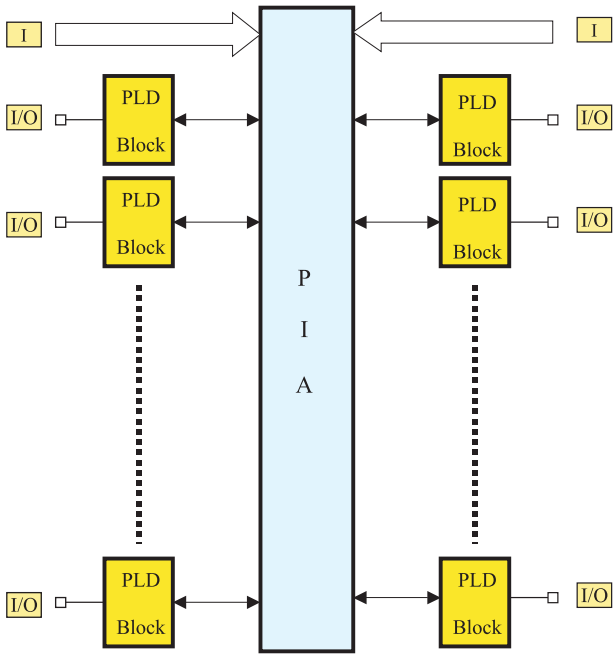
Celem artykułu jest przedstawienie koncepcji działania wielokontekstowego sterownika przemysłowego zrealizowanego z wykorzystaniem struktur programowalnych. W artykule jest przedstawiony proces rozwoju układów programowalnych oraz podstawowe cechy złożonych układów programowalnych, a także sprzętowe i programowe zasoby programowalnych systemów na chipie. Pozwala to na przedstawienie nowej koncepcji realizacji sterownika przemysłowego oraz zaprezentowanie możliwości sterownika na prostym przykładzie sterowania elektrofiltrem.

Złożone struktury programowalne

Zdaniem autora struktury programowalne można obecnie podzielić na cztery podstawowe grupy [1]:
1) proste układy PLD (SPLD – *Simple PLD*)
2) złożone układy matrycowe (CPLD – *Complex PLD*)
3) układy FPGA (*Field Programmable Gate Array*)
4) programowalne systemy na chipie (pSoC – *programmable System on Chip*).

Najprostsze, a zarazem najstarsze układy programowalne powoli wychodzą z użycia, ale występujące w nich sprzętowe rozwiązania zostały z powodzeniem wdrożone w rodziny bardziej złożonych układów. Największą popularność w grupie prostych układów programowalnych zdobyły struktury PAL, których najbardziej zaawansowane odmiany z konfigurowalnymi komórkami wyjściowymi (np. 22V10) są wykorzystywane do chwili obecnej. Tego typu struktura, zawierająca stałe połączenia w matrycy OR i programowalne w matrycy AND, charakterystyczna dla układów PAL stanowi podstawę większości układów matrycowych (CPLD – *Complex Programmable Logic Devices*), a obecnie znalazła również swoje miejsce w programowalnych systemach na chipie (pSoC – *programmable System on Chip*). Układy CPLD zwane układami matrycowymi zawierają programowalną matrycę logiczną oraz konfigurowalne bloki logiczne podobne do prostych układów PLD (rys. 1).

Rozwój technologiczny doprowadził do powstania dwóch głównych grup złożonych układów programowalnych, tj. układów CPLD i FPGA. Drugą – wydaje się, że najbardziej popularną rodziną złożonych układów programowalnych – są struktury FPGA typu tablicowego (*Field Programmable Gate Array*) [1, 6, 7]. Zawierają one konfigurowalne bloki logiczne (CLB – *Configurable Logic Block*) otoczone na obrzeżach blokami wejścia/wyjścia. Pomiędzy konfigurowalnymi blokami logicznymi oraz blokami wejścia/wyjścia występuje przestrzeń, w której biegną różnego typu trakty połączeń (rys. 2).



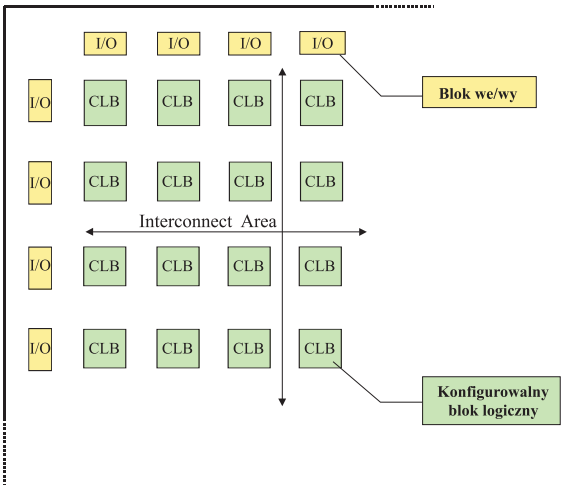
PIA – *Programmable Interconnect Area*

Rys. 1. Architektura układów CPLD

Podstawowa różnica w architekturze układów CPLD i FPGA tkwi w strukturze bloku logicznego. Bloki logiczne występujące w układach CPLD zawierają podobne elementy, do których należy zaliczyć: programowalną matrycę iloczynów, oryginalny dla danej rodziny układów blok rozprowadzania iloczynów, programowalną komórkę logiczną zawierającą element pamięci oraz blok wyjściowy (rys. 3).

Istotna różnica pomiędzy poszczególnymi rodzinami układów CPLD tkwi w sposobie rozprowadza-

* *dr hab. inż. Dariusz Kania – Instytut Elektroniki, Politechnika Śląska*

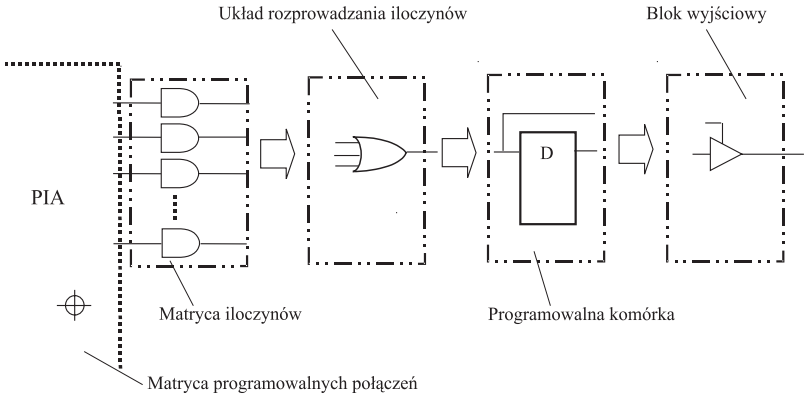


Rys. 2. Architektura układów FPGA

nia iloczynów do komórek programowalnych. W konkretnych rozwiązaniach można znaleźć szereg różnorodnych mechanizmów sprzętowych. Umożliwiają one efektywne wykorzystywanie zawartych w strukturze iloczynów. Do najbardziej popularnych rozwiązań układowych należy zaliczyć stosowane w układach MACH programowalne rozdzielacze oraz występujące w układach MAX dwa rodzaje ekspanderów: ekspandery wspólne oraz ekspandery równoległe [1]. W niektórych strukturach CPLD (*Cypress FLASH370*) można znaleźć jeszcze bardziej efektywny sposób rozmieszczania iloczynów pozwalający na tworzenie bloków logicznych typu PAL zawierających dostosowaną do potrzeb liczbę iloczynów [1].

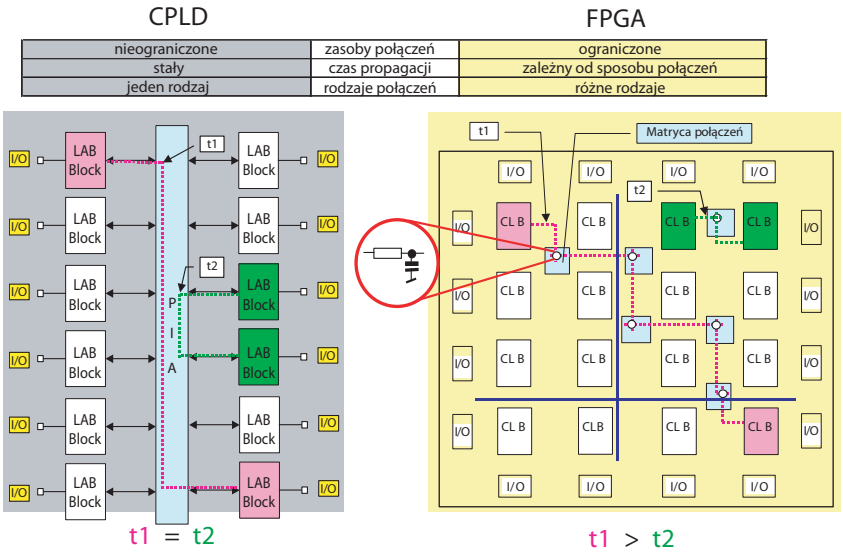
Podsumowując, można powiedzieć, że układy CPLD wykorzystują bloki logiczne typu PAL zawierające niewielką liczbę iloczynów (3–8), w odróżnieniu od układów FPGA, które zawierają tzw. bloki tablicowe (LUT – *Look-up Table*) mające niewielką liczbę wejść (4, 5), będące w zasadzie blokami logicznymi typu PLE, mającymi funkcjonalne możliwości takie jak pamięci typu PROM. Sterowanie elementami pamięci zawartymi w programowalnej komórce układów CPLD i w konfigurowalnym bloku logicznym zawartym w układach FPGA w zasadzie się nie różni. Najczęściej elementem pamięci jest przerzutnik typu D.

Istotne różnice pomiędzy układami CPLD i FPGA są związane z występującymi w nich programowalnymi połączeniami. W strukturach CPLD występuje jeden rodzaj połączeń wprowadzających stałe opóźnienie, niezależne od „odległości" pomiędzy programowalnymi blokami. W strukturze FPGA połączenia mają charakter segmentowy. Każdy punkt połączenia może być modelowany jako układ RC wprowadzający opóźnienie. Fakt ten sprawia, że czas propagacji wnoszony przez ścieżki połączeń zależy od usytuowania łączonych bloków wewnątrz struktury FPGA (rys. 4). Dodatkowo zasoby połączeń w strukturach FPGA są ograniczone, co sprawia, że duże znaczenie ma sposób rozmieszczenia bloków wewnątrz struktury (*placement, fitting*) oraz sposób prowadzenia połączeń (*routing*). Oprócz połączeń ogólnego przeznaczenia w strukturze FPGA występuje ograniczona liczba linii długich, zwykle pionowych i poziomych, które wnoszą mniejsze opóźnienia i są wykorzystywane do rozprowadzania sygnałów krytycznych pod względem



PIA – *Programmable Interconnect Area*

Rys. 3. Architektura programowalnego bloku logicznego układów CPLD

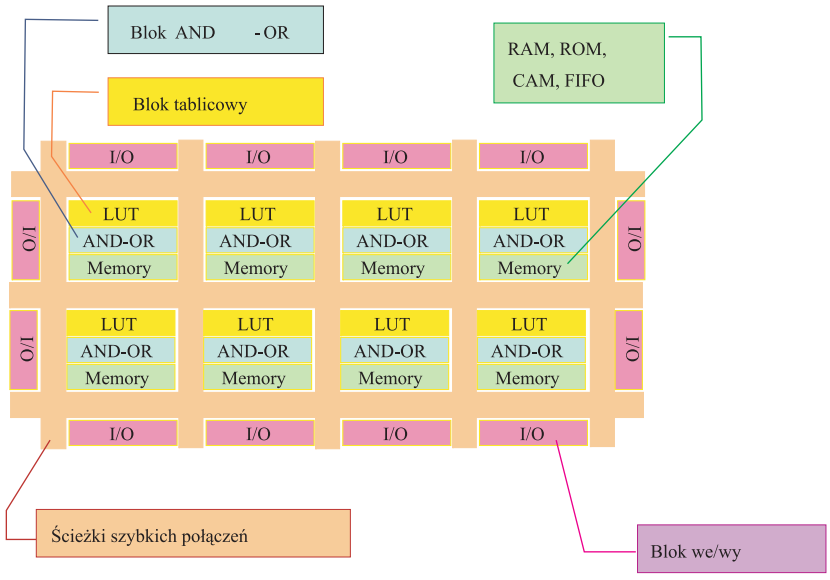


Rys. 4. Programowalne połączenia – podstawowe różnice

czasowym. Podstawowe różnice związane z zasobami połączeń występujących w dwóch głównych rodzinach złożonych układów programowalnych są przedstawione na rys. 4.

Istotna różnica występuje również w sposobie realizacji programowalnych połączeń i związanym z nim procesem konfiguracji układu. Konfiguracja współczesnego układu CPLD najczęściej odbywa się w systemie (ispPLD – *in system programming Programmable Logic Devices*) i polega na programowaniu komórek typu EEPROM. Najczęściej do przesyłania danych używa się firmowego interfejsu. Po wyłączeniu zasilania dane konfiguracyjne są utrzymywane w nieulotnych komórkach EEPROM. Inaczej wygląda sytuacja w przypadku układów FPGA. Dane konfiguracyjne są zawarte w komórkach pamięci typu SRAM i zanikają po wyłączeniu zasilania. Po włączeniu napięcia zasilania są wczytywane ze stałej pamięci zewnętrznej (PROM, EPROM, EEPROM) lub przesyłane z systemu mikroprocesorowego. Proces konfiguracji układów FPGA trwa krócej niż układów CPLD.

Współczesne złożone układy programowalne często zawierają elementy charakterystyczne dla układów CPLD, FPGA oraz bardzo elastyczne bloki pamięci. Przykład najprostszej struktury tego typu jest przedstawiony na rys. 5.

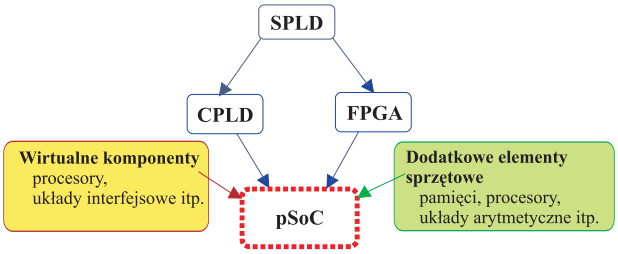


Rys. 5. Architektura układów APEX

Programowalne systemy na chipie

Rozwój technologiczny umożliwia obecnie wytwarzanie układów o coraz większej złożoności. Współczesny układ cyfrowy to w większości przypadków system mikroprocesorowy otoczony peryferyjnym układem cyfrowym. Układy tego typu są zwane systemami na chipie (*System on Chip*).

Współczesny układ programowalny jest wynikiem scalenia w jednej strukturze różnorodnych elementów. Ciągle istnieje w nim możliwość dostosowania struktury do potrzeb użytkownika, cecha charakterystyczna dla układów ASIC (*Application Specific Integrated Circuits*). Jedną z grup systemów na chipie, w której główną rolę odgrywają struktury programowalne są tzw. programowalne systemy na chipie (pSoC – *programmable System on Chip*). Ponieważ tego typu układy obecnie dynamicznie się rozwijają, trudno jednoznacznie zaklasyfikować pojawiające się na rynku różnorodne rozwiązania. Zdaniem autora programowalne systemy na chipie można podzielić na dwie grupy.



Rys. 6. Ewolucja cyfrowych struktur programowalnych

Pierwszą z nich są układy, w których system mikroprocesorowy jest obudowany dostosowywaną do potrzeb strukturą programowalną, np. układy FPSLIC firmy Atmel (FPSLIC – *Field Programmable System-Level Integrated Circuit*). Rozwój języków opisu sprzętu spowodował, że system mikroprocesory niekoniecznie musi być realizowany bezpośrednio na krzemie. Istnieje możliwość użycia tzw. wirtualnych komponentów, np. wirtualnych mikroprocesorów, układów interfejsowych itp., które odpowiednio dobrane i sparametryzowane mogą być następnie realizowane w strukturze programowalnej. Zdaniem autora, tego typu rozwiązania można zaliczyć do drugiej grupy programowalnych systemów na chipie. Proces ewolucji układów programowalnych jest zobrazowany na rys. 6.

W programowalnych systemach na chipie (pSoC) można znaleźć elementy charakterystyczne dla układów CPLD oraz FPGA. W układach tych, oprócz charakterystycznej dla układów CPLD i FPGA części programowalnej, można wyróżnić dwa rodzaje dodatkowych zasobów: realizowane w procesie wytwarzania układu scalonego zasoby sprzętowe oraz implementowane w procesie programowania zasoby programowe, dostępne w postaci różnorodnych wirtualnych komponentów.

Zasoby sprzętowe programowalnych systemów na chipie

Programowalne systemy na chipie mają wbudowanych szereg elementów. Należą do nich m.in.:

- bardzo elastycznie konfigurowalne bloki pamięci
- bloki realizujące złożone operacje arytmetyczne

- elementy umożliwiające pracę w sieci
- procesory i różnego typu systemy procesorowe
- różnego typu układy interfejsowe.

Wbudowywane w struktury programowalne pamięci są najczęściej konfigurowalne i mogą pracować jako pamięci typu RAM, ROM, FIFO lub CAM. Często z elementów pamięci można tworzyć również rejestry przesuwające. Pojemności pamięci układów Stratix firmy Altera sięgają 9 Mbit. Na uwagę zasługują również bardzo rozbudowane bloki arytmetyczne. Na przykład w układach Virtex II firmy Xilinx znajduje się 168 układów mnożących operujących na liczbach 18-bitowych. W układach Stratix firmy Altera blok arytmetyczny zwany blokiem DSP zawiera rejestr, układ mnożący oraz blok dodająco-odejmujący operujący na liczbach 36-bitowych. Może on również pracować jako 52-bitowy blok dodająco-odejmujący. Na uwagę zasługuje możliwość szybkiej zmiany typu wykonywanej operacji (dodawanie/odejmowanie). Układy programowalne mogą stanowić element umożliwiający dołączenie układu cyfrowego do sieci, ponieważ zawierają wbudowane elementy sieciowe. Najczęściej znajdujące się w nich bloki umożliwiają pracę w sieci Ethernet. Programowalne systemy na chipie bardzo często mają wbudowane procesory, czy nawet rozbudowane systemy procesorowe. Można w nich znaleźć bardzo różnorodne rozwiązania, poczynając od układów FPSLIC firmy Atmel, a kończąc na rozbudowanych elementach znajdujących się w strukturach Virtex oraz Stratix. W strukturach FPSLIC wbudowano mikrokontroler AVR, który współpracuje z układem programowalnym. Firma Xilinx w swoich najbardziej rozbudowanych strukturach zastosowała aż cztery procesory Power PC. W układach Stratix firmy Altera znajdują się, oprócz procesora ARM, dodatkowe elementy systemu procesorowego takie jak kontroler przerwań, układ czasowy oraz układ UART. Oprócz wbudowanych różnego typu bloków funkcjonalnych na uwagę zasługują bardzo elastyczne bloki wejścia/wyjścia. Umożliwiają one współpracę współczesnych układów programowalnych z dowolną rodziną układów cyfrowych wytwarzanych w technologii bipolarnej lub CMOS.

Zasoby programowe programowalnych systemów na chipie

Silną stroną programowalnych systemów na chipie jest możliwość implementowania w tych strukturach dowolnych, bardzo złożonych elementów. W ostatnim czasie powstał ogromny, bardzo dynamicznie rozwijający się rynek wirtualnych komponentów. Wytwarzane są one nie tylko przez firmy produkujące układy programowalne. Doszło do sytuacji, w której użytkownik stoi przed trudnym wyborem elementu, tym bardziej, że ma możliwość wykorzystywania darmowych wirtualnych komponentów dostępnych w Internecie. Nie ulega jednak wątpliwości, że największym zainteresowaniem cieszą się „pewne" wirtualne komponenty oferowane przez producentów układów cyfrowych. Trudno w niniejszym artykule przybliżyć czytelnikowi sposób korzystania z wirtualnych komponentów, czy też scharakteryzować poszczególne, dostępne elementy, stąd zamiarem autora jest tylko zwrócenie uwagi na przykładowe, najpopularniejsze wirtualne procesory oferowane przez dwie od lat wiodące firmy produkujące układy PLD, tzn. firmę Altera oraz Xilinx.

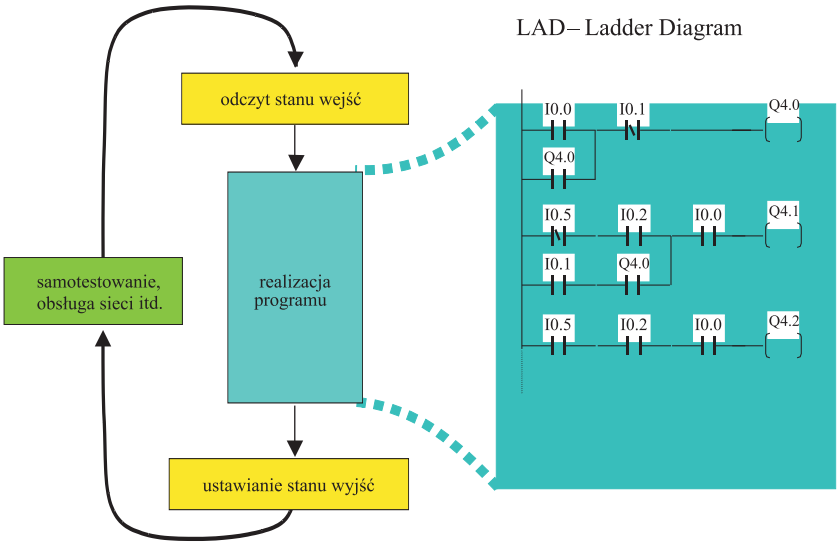
Firma Xilinx w szerokiej ofercie wirtualnych komponentów ma m.in. dwa wirtualne procesory. Pierwszy z nich – PicoBlaze jest najprostszym 8-bitowym mikrokontrolerem. Na drugim krańcu pod względem złożoności jest procesor MicroBlaze – 32-bitowy mikroprocesor o architekturze RISC. Oprócz jądra procesora zawiera on dodatkowo układy peryferyjne. Firma Altera dostarcza procesor Nios. W przypadku tego procesora na uwagę zasługuje możliwość konfiguracji jego zasobów wewnętrznych. W zasadzie patrząc na systemy wspomagające projektowanie firmy Altera i Xilinx, ową specyficzną konfigurowalność można zauważyć już w starszych narzędziach. Zdaniem autora, elementy oferowane przez firmę Altera zawsze były bardziej uniwersalne i często miały możliwość dostosowywania zasobów logicznych do potrzeb. Ta właściwość również jest widoczna w przypadku procesora Nios.

Podsumowując skrótowe przedstawienie zasobów programowalnych systemów na chipie, należy zwrócić jeszcze uwagę na jedną cechę tych układów, ściśle związaną z zasobami sprzętowymi oraz programowymi. Współczesny układ programowalny ma możliwość częściowej rekonfiguracji. Cecha ta stwarza zupełnie nowe możliwości współczesnych układów. Wprowadza również w proces syntezy szereg nowych problemów. W tego typu strukturach programowalnych można realizować tzw. układy wielokontekstowe. Kontekst to jeden wydzielony blok układu, kojarzony z jednym cyklem programowania. Pojawia się jednak szereg nierozwiązanych problemów: jak efektywnie dzielić układ na konteksty, co realizować sprzętowo, co programowo itp.

Zasada pracy współczesnego sterownika programowalnego

Współczesny sterownik przemysłowy to w zasadzie specjalizowany komputer o architekturze bitowo-bajtowej wykonujący program sterowania w sposób szeregowo-cykliczny. W skład zespołu czynności wykonywanych w jednym cyklu, zwanym cyklem programowym, wchodzą kolejno:

- odczyt stanu wejść i zachowanie tego stanu w przygotowanym do tego celu obszarze pamięci danych zwanym obszarem odwzorowania wejść (PII – *Process Image Input*)
- realizacja programu użytkownika – wyniki realizacji programu są zapisywane w odpowiednim obsza-

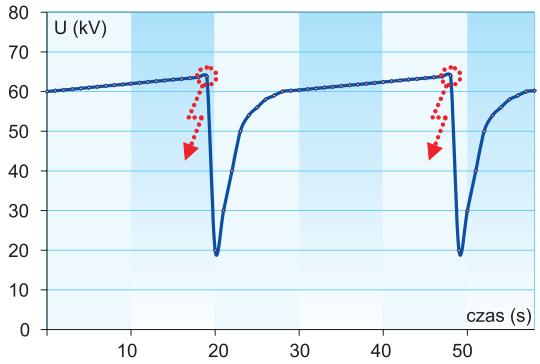


Rys. 7. Cykl programowy sterownika przemysłowego

rze pamięci danych, zwanym obszarem odwzorowania wyjść (PIQ – *Process Image Output*)
- ustawienie wyjść zgodne z zawartością danych w obszarze odwzorowania wyjść
- dodatkowe procedury np. samotestowanie sterownika, obsługa sieci itp.

Istnieje szereg metod opisu programu sterowania [5]. Do najpopularniejszych należą: schemat drabinkowy (LD – *Ladder Diagram*), opis tekstowy składający się z poszczególnych rozkazów (STL – *Statement List*) oraz schemat bloków funkcji (FBD – *Function Block Diagram*). Różnorodność sposobów opisu programu sterowania nie wpływa na sposób jego realizacji. Realizacja programu zawsze sprowadza się do sekwencyjnego wykonywania poszczególnych rozkazów mikroprocesora, który jest głównym elementem każdego współczesnego sterownika. Programowo-cykliczny sposób wykonywania programu wprowadza znaczące ograniczenia w obszarze zastosowań współczesnych sterowników przemysłowych. Główne ograniczenia wynikają ze stosunkowo długiego czasu wykonywania programu. Powoduje to, że obszar zastosowań sterowników obejmuje zagadnienia sterowania stosunkowo wolnymi obiektami. W innych przypadkach użycie sterownika wymaga stosowania wyrafinowanych „sztuczek programowych" pozwalających na sterowanie „szybkimi procesami".

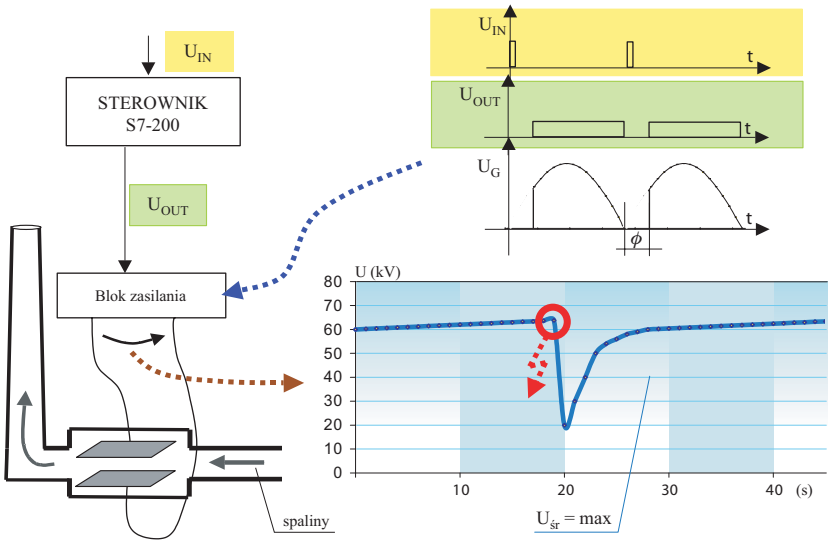
Przykładem powyższych problemów może być sposób realizacji programu sterowania elektrofiltrem zrealizowany z użyciem sterownika programowalnego firmy Siemens rodziny S7200. Istota sterowania sprowadza się do utrzymywania pomiędzy elektrodami elektrofiltru możliwie jak największego napięcia. Praktycznie chodzi o jak największą wartość średnią napięcia w ciągu jednej minuty. W przypadku doprowadzenia do elektrod elektrofiltru zbyt wysokiego napięcia dochodzi do wyładowań elektrostatycznych. Ponieważ wartość napięcia, przy której występują wyładowania zależy od szeregu czynników, istota sterowania sprowadza się więc do utrzymywania możliwie jak najwyższego napięcia, bliskiego wartości napięcia wyładowań. Widać więc, że zachodzi konieczność doprowadzania do wyładowań pomiędzy elektrodami elektrofiltru, ponieważ jest to jedyna metoda potwierdzająca właściwe działanie urządzenia. Zadanie układu sterowania sprowadza się więc do wywoływania ciągłego, powolnego wzrostu napięcia pomiędzy elektrodami elektrofiltru, aż do momentu wystąpienia wyładowania. Po wyładowaniu jest wykonywana tzw. procedura „odbudowy napięcia", uniemożliwiająca wystąpienie wyładowań wtórnych. Przykładowy wykres czasowy napięcia pomiędzy elektrodami elektrofiltru przedstawiono na rys. 8.



Rys. 8. Przykładowy wykres czasowy napięcia pomiędzy elektrodami elektrofiltru

W układzie sterowania sterownik współpracuje z blokiem zasilania wytwarzającym wysokie napięcie. Krytycznym elementem całego programu ze względu na zależności czasowe jest sposób sterowania kątem zapłonu tyrystorów. Sterownik otrzymuje sygnał informujący go o przejściu napięcia sieciowego przez zero i w odpowiednim momencie powinien spowodować załączenie tyrystorów (rys. 9).

Jak widać, korzystne byłoby, gdyby cały program mógł być realizowany w czasie pojedynczych milisekund. To oczywiście jest niemożliwe, ponieważ np. procedura wyznaczania wartości średniej napięcia trwa kilka milisekund [4, 5]. Sterownik oprócz wyznaczania wartości średniej napięcia musi wykonywać szereg innych operacji np. wyznaczać wartość średnią prądu, wartość szczytową napięcia, monitorować układy zabezpieczeń itp. Rozwiązanie przedstawionego problemu na obecnie dostępnych sterownikach przemysłowych wiąże się z rozwiązaniem szeregu problemów. Najciekawszym pomysłem pozwalającym na realizację przedstawionego zadania jest sposób minimalizacji czasu reakcji sterownika przemysłowego na cyklicznie występujące przerwania zewnętrzne oparty na koncepcji synchronizacji głównej pętli programowej z otoczeniem sterownika.



Rys. 9. Idea sterowania elektrofiltrem

Główne elementy wspomnianych pomysłów są przedstawione w pracach [2, 3].

Sterownik przemysłowy przyszłości

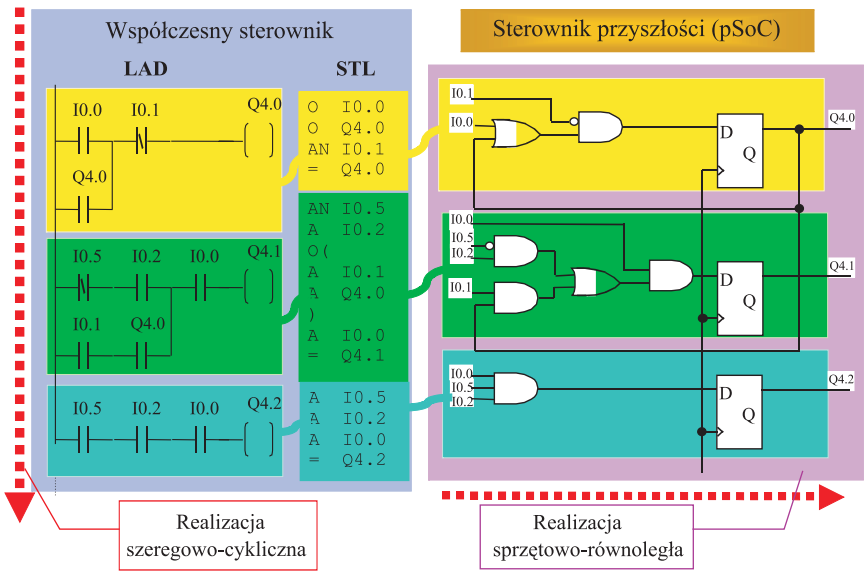
Nie ulega wątpliwości, że przedstawione zagadnienie można by bez problemu rozwiązać, mając do dyspozycji sterownik realizujący program sterowania znacznie szybciej. Możliwość konstrukcji takiego sterownika stwarzają struktury programowalne. Znaczne przyśpieszenie pracy sterownika opartego na programowalnych systemach na chipie (pSoC) wiąże się przede wszystkim z możliwością zamiany szeregowo-cyklicznej realizacji programu na realizację sprzętowo-równoległą.

Na rys. 10 przedstawiono fragment przykładowego programu w postaci schematu drabinkowego oraz listy instrukcji. Poszczególne segmenty programu mogą być bezpośrednio realizowane sprzętowo z wykorzystaniem zasobów sprzętowych dostępnych w strukturach programowalnych. Zastosowane przerzutniki typu D pozwalają na zapamiętanie stanu wyjść i wykorzystanie go w następnym cyklu pracy. Odpowiednia kompilacja opisu programu sterowania może umożliwić wyodrębnienie bloków wykonywanych równocześnie, co powoduje dodatkowe przyśpieszenie pracy sterownika. Nie bez znaczenia jest również możliwość zachowania dobrze znanych form przedstawiania programów sterowania. W zasadzie projektant, mając odpowiedni program przekształcający opis programu sterowania na dane konfiguracyjne sterownika (struktury pSoC), dostałby do ręki sterownik działający wielokrotnie szybciej (2-3 rzędy wielkości). Dodatkowo nie musiałby rezygnować z doświadczeń zdobytych podczas pisania programów sterowania realizowanych przez klasyczne mikroprocesorowe sterowniki przemysłowe.

Nowe możliwości sterownika przemysłowego przyszłości

Nasuwa się pytanie, czy układy programowalne są wystarczająco „duże", aby było można z ich użyciem zrealizować każdy program sterowania. Wydaje się, że na obecnym etapie rozwoju technologicznego wielkość struktur programowalnych jest niewystarczająca. Z drugiej strony, zawsze można znaleźć odpowiednio duży problem, który wymaga użycia więcej niż jednego współczesnego sterownika przemysłowego. Często dekompozycja zadań, na bloki realizowane na poszczególnych sterownikach korzystnie wpływa na pracę, niezawodność, możliwość rozbudowy i łatwość projektowania całego złożonego systemu sterowania. Sterownik, którego realizację oparto na układach programowalnych taki miałby zupełnie nowe właściwości, związane z możliwością jego dynamicznej rekonfiguracji.

Rozważmy problem realizacji przedstawionego w artykule sterowania elektrofiltru na „sterowniku przyszłości".



Rys. 10. Sposoby realizacji programu sterowania

Oprócz krytycznego ze względów czasowych problemu sterowania kątem zapłonu tyrystorów, sterownik musi wykonywać szereg innych zadań. Podzielmy je na porównywalne, ze względu na zasoby programowe, grupy:

A – sterowanie blokiem zasilania, pomiar napięć, prądów, sprawdzanie zabezpieczeń itp.

B – sterowanie układem odpopielania, strzepywania pyłów itp.

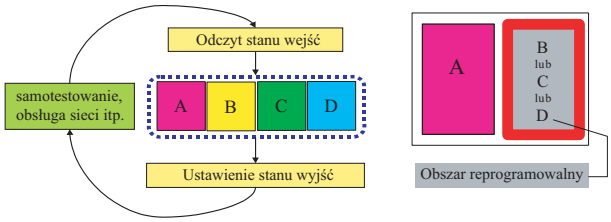
C – obliczenie wartości średniej, maksymalnej prądu (raz na minutę)

D – obliczenie wartości średniej, maksymalnej napięcia (raz na minutę).

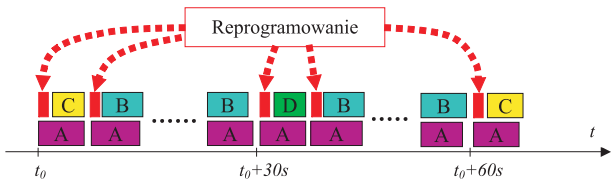
Zadanie A musi być wykonywane w każdym obiegu pętli programowej. Procesy tworzące grupę B są stosunkowo wolne, stąd mogą być obsługiwane znacznie rzadziej. Wyznaczanie wartości napięć i prądów może być wykonywane jednokrotnie w czasie jednej minuty (zadanie C, D).

Odpowiedni podział zadań na grupy o porównywalnej złożoności, różniące się pod względem wymagań czasowych stwarza możliwość realizacji programu sterowania w trybie wielokontekstowym. Każda grupa zadań może być skojarzona z jednym kontekstem tzn. blokiem związanym z jednym cyklem reprogramowania. Zadanie A musi być realizowane w każdym cyklu, stąd ciągle zajmuje pewną część zasobów sprzętowych sterownika. Pozostałe grupy zadań nie muszą być wykonywane w każdym cyklu. Istnieje więc możliwość wykorzystania nowych właściwości sterownika. Można go odpowiednio konfigurować, uwzględniając aktualne potrzeby (rys. 11).

Koncepcję wielokontekstowej pracy sterownika przedstawiono na rys. 12.



Rys. 11. Koncepcja realizacji poszczególnych grup zadań



Rys. 12. Wykres czasowy przedstawiający zasadę pracy wielokontekstowej

Prostokąty z opisem A obrazują grupę zadań wykonywanych w każdej pętli programowej. Podobnie są wykonywane zadania z grupy B. Jednakże, co 30 sekund w proces wykonywania zadań z grupy B są wtrącane na przemian procedury obliczeń, tworzące grupy zadaniowe C i D. Każdą zmianę konfiguracji części B, C lub D (rys. 11) poprzedza proces reprogramowania.

Przedstawiona na rys. 11 i 12 koncepcja pracy sterownika wielokontekstowego, znacznie rozszerza typowy obszar zastosowań sterowników przemysłowych. Pozwala ona wykorzystywać przedstawiony sterownik w układach sterowania nie tylko szybkich, ale również bardzo złożonych obiektów przemysłowych.

Podsumowanie

Rozwój struktur programowalnych znacząco zmienił sposób projektowania i konstrukcji układów elektronicznych. Niektóre nowe cechy układów programowalnych do tej pory nie zostały w pełni wykorzystane. W artykule przedstawiono koncepcję budowy i działania wielokontekstowego sterownika programowalnego, który wykorzystuje jedną z nowych cech programowalnych systemów na chipie. Oczywiście istnieje szereg problemów, których rozwiązanie umożliwi dopiero pełne zastosowanie przedstawionej koncepcji w praktyce przemysłowej. Zdaniem autora, do najważniejszych problemów należy zaliczyć efektywny, algorytmiczny sposób podziału programu sterowania na konteksty. Wydaje się jednak, że korzyści wynikające z przedstawionej koncepcji, doprowadzą w przyszłości do opracowania sterowników przemysłowych znacznie szybszych, zdecydowanie poszerzających zakres ich zastosowań.

Bibliografia

1. Altera, Xilinx, Lattice, Atmel, Actel, Cypress Data Book.
2. D. Kania, K. Pucher, *Realizacja programu z uzależnieniami czasowymi na bazie sterownika S7200 z uwzględnieniem problemów związanych z synchronizacją obiegu programu w stosunku do autotestu, obsługi sieci itp.* PAK 12/1999, s. 31–34.
3. K. Pucher, D. Kania, *Sposób minimalizacji czasu reakcji sterownika przemysłowego na cyklicznie występujące przerwania zewnętrzne*, zgłoszenie patentowe P 337909, 17.01.2000.
4. S7 200 *Programmable Controller System Manual*, Siemens AG 2003.
5. S7 200 *Programming Manual*, Siemens 1998
6. K. Wiatr, *Sprzętowe implementacje algorytmów przetwarzania obrazów w systemach wizyjnych czasu rzeczywistego*, Uczelniane Wydawnictwo Naukowo-Dydaktyczne AGH, Kraków 2002
7. J. Pasierbicki, P. Zbysiński, *Układy programowalne w praktyce*, WKŁ, Warszawa 2002. ■