Dariusz KANIA, Krzysztof PUCHER

POLITECHNIKA ŚLĄSKA
INSTYTUT ELEKTRONIKI

Realizacja programu z uzależnieniami czasowymi na bazie sterownika S7 200 z uwzględnieniem problemów związanych z synchronizacją obiegu programu w stosunku do autotestu, obsługi sieci itp.

Streszczenie

W pracy zostały omówione problemy związane z synchronizacją pętli głównej programu sterownika (OB1) z wbudowanymi procesami systemowymi takimi jak: program autotestu, obsługi sieci i urządzeń peryferyjnych. Analizowane w pracy problemy są kluczowe w przypadku realizacji programu z krytycznymi uzależnieniami czasowymi.

Abstract

Problems associated with synchronization of the main loop of the controller program (DB1) with built-in system processes such as autotest program, network service and peripheral equipment. The problems under analysis are of a key-charakter in the case of realization of the program with critical time dependences.

Wstęp

Stale rosnąca koniunktura na wykorzystywanie sterowników przemysłowych wywiera swoistego rodzaju parcie na projektantów do coraz lepszego i wydajniejszego ich wykorzystywania. Sterownik przemysłowy jest komputerem (czasami o stosunkowo dużej mocy obliczeniowej) dostosowanym do potrzeb przemysłowych. Granice wykorzystania sterowników są trudne do oceny, tak jak trudne są do oceny granice wykorzystania komputera. Fakt ten powoduje, że w systemach opartych na sterownikach przemysłowych tkwią olbrzymie możliwości nie zawsze zauważane przez projektantów. Na temat wykorzystywania sterowników wśród szerokiego grona projektantów funkcjonuje szereg różnych mitów i stereotypów. Przyjęło się, że sterownik do wielu zastosowań nie nadaje się - tu podawane są wszelakiego rodzaju przypadki i sytuacje. W znacznej części z takim zdaniem można się zgodzić, jednak w wielu sytuacjach sądy te, zdaniem autorów tego artykułu w świetle nowych rozwiązań sterowników przemysłowych, są nieprawdziwe. Główne obawy projektantów co do wykorzystywania sterowników związane są z ich pracą w trybie rzeczywistym, cyklicznym modelem wykonywania programu, reakcją na przerwania itp. Wiele z nich ma swoje uzasadnienie, jednak najnowsze rozwiązania techniczne sterowników pokonują w znacznym stopniu niedogodności rozwiązań starszych modeli tych urządzeń. Dokonano szeregu zmian, które uczyniły ze sterownika narzędzie wygodne i wszechstronne. W konstruowanych sterownikach ciągle jednak pozostają pewne rozwiązania, które są specyficzne dla tego typu urządzeń. Najistotniejszą cechą jest to, że w obiegu programu nie jest wykonywany tylko program użytkowy, ale także i inne systemowe procedury takie jak: samotestowanie, czytanie wejść, wyjść i inne podobnego typu procedury związane ze specyficznymi właściwościami danego sterownika. Istotną cechą jest także to, że dodatkowe programy wykonywane są najczęściej na początku lub końcu pętli obiegu programu, którego czas realizacji w każdym jego obiegu może być inny i trudny do przewidzenia, co może mieć istotny wpływ na sposób pisania programu z dużymi uzależnieniami czasowymi. Najczęściej programista nie ma wpływu na wykonywanie tych dodatkowych procedur, które to np. muszą być wykonywane dla prawidłowego funkcjonowania sterownika. A zatem, aby napisać program, który realizuje jakieś krytyczne uzależnienia czasowe, programista musi przewidzieć wykonywanie tych dodatkowych procedur i tak napisać program, aby nie miały one szkodliwego oddziaływania na projektowany system. W kontekście wykorzystania możliwości współcześnie produkowanych sterowników przemysłowych może to być trudne. Uzależnienia czasowe o których mowa, mogą wynikać z konieczności stosowania np. przerwań lub wejść oraz wyjść, dla których istnieje potrzeba szybkiego ich odczytywania lub ustawiania. Prezentowany artykuł traktuje właśnie o takich sytuacjach, zwracając szczególną uwagę na sposób pisania programu pod tym kątem - w warunkach silnych obostrzeń czasowych. Można powiedzieć, że obecnie istnieje możliwość tworzenia systemów sterowania na bazie sterowników przemysłowych bezpośrednio do tego celu nie przeznaczonych, umożliwiając realizację projektu na sterowniku prostszym (tańszym)[1-4]. Prezentowany artykuł przedstawia konkretną aplikację na bazie sterownika S7-200, która realizuje sterowanie układem tyrystorowym zasilacza elektrofiltru przemysłowego. Aplikacja ta wymagała napisania programu z silnymi uzależnieniami czasowymi (o których mowa w dalszej części artykułu).

Problemy związane programowymi uzależnieniami czasowymi

W większości przypadków konstruowane aplikacje na bazie sterowników przemysłowych realizują proste funkcje przełączające, gdzie wystarczające okazuje się użycie sterownika typowego o przeciętnych parametrach czasowych. Są jednak obiekty (np. układy tyrystorowe), gdzie wymaga się szybkiej obsługi wejść oraz wyjść w obecności szybkiego systemu zewnętrznych przerwań sprzętowych. Część współczesnych konstrukcji sterownikowych cechuje właśnie takiego typu rozwiązanie. Przykładem może być sterownik firmy SIEMENS serii S7 - S7 200. Aplikacje realizujące funkcje z silnymi uzależnieniami czasowymi można by (zdaniem autorów) uznać za nietypowe. Realizacja nietypowych aplikacji związana jest z nietypowym wykorzystywaniem sterownika. Opiszmy pokrótce kilka problemów związanych z uruchamianiem wielu nietypowych aplikacji, w których nietypowość głównie kojarzona jest z dużymi wymogami czasowymi sterowanego obiektu.

- **Badanie stanów wejść**
  W wielu konstrukcjach sterowników nie ma możliwości natychmiastowego badania stanu wejść, badanie to możliwe jest tylko raz w ciągu obiegu programu, lecz w wielu aplikacjach konieczne jest natychmiastowe testowanie wejść. Z takimi przypadkami możemy się spotkać np. przy sterowaniu tyrystorami, zliczaniu impulsów o krótkim czasie trwania itp. Współczesne sterowniki posiadają właśnie takie możliwości, gdzie do odczytu stanu wejść używa się specjalnie przeznaczonych do tego celu instrukcji. Wykorzystanie przez programistę takiej cechy sterownika może zadecydować o realizacji projektu z silnymi uzależnieniami czasowymi [1-2].
- **Ustawianie wyjść**
  Podobnie jak dla wejść współczesne sterowniki przystosowane są do szybkiego ustawiania wyjść. Jest to odejście od typowego rozwiązania polegającego na równoległym wpisie wszystkich stanów wyjść (uprzednio przeliczonych i umieszczonych w pamięci odwzorowań) na wyjścia sterownika. Podobnie jak to opisywano w poprzednim punkcie wykorzystanie cechy sterownika polega-

jącej na możliwości korzystania z takich wyjść może okazać się warunkiem koniecznym przy realizacji wielu projektów. Przykładem może tutaj być generacja impulsów o czasie trwania znacznie krótszym niż pojedynczy obieg programu sterownika lub szybkie (bez oczekiwania na koniec cyklu obiegu programu) ustawianie pewnych sygnałów w reakcji na inne [1-2].

- **Reakcja na przerwania**
  Szybka realizacja przerwań to bardzo ważna cecha sterownika. Prosta realizacja znacznej części aplikacji możliwa jest tylko za pomocą przerwań - szybkich przerwań. Współczesne, wybrane szybkie sterowniki posiadają wbudowane systemy przerwań o bardzo krótkich czasach reakcji na przerwanie, porównywalnych z czasem wykonywania pojedynczego rozkazu. Starsze typy sterowników nie posiadały systemów przerwań o takich możliwościach. Sytuacja taka stwarza ich wiele, przybliżając nieco konstrukcje sterowników do rozwiązań na specjalizowanych systemach mikroprocesorowych, które do tej pory były bezkonkurencyjne [1-2].
- **Wykonywanie podstawowego obiegu pętli**
  Współczesne sterowniki cechuje stosunkowo bogaty wewnętrzny system operacyjny. Poza oczywistymi zaletami wynikającymi z tego faktu, istnieją także i wady takiego stanu rzeczy. W pętli głównej, poza wykonywaniem programu głównego, wykonywany jest szereg programów systemowych takich jak: samotestowanie, obsługa sieci, obsługa panelu operatorskiego i wiele innych podobnych procesów. Wykonywanie tych procesów jest często podstawowe dla właściwego funkcjonowania sterownika. Stwarza to jednak pewne ograniczenia czasowe np. w funkcjonowaniu zewnętrznych sprzętowych przerwań. W większości przypadków procedury systemowe nie mogą być przerywane, co ma bezpośrednie konsekwencje w czasie reakcji na dane przerwanie a w wielu przypadkach eliminuje zastosowanie danego sterownika. [1-2].
- **Możliwość generowania impulsów napięciowych o czasie trwania ustalanym sprzętowo**
  Część sterowników posiada możliwość generowania na swoich wyjściach krótkich impulsów, których wygenerowanie drogą programową jest często niemożliwe. Istnienie takiej cechy sterownika może w niektórych przypadkach zadecydować o możliwości realizacji aplikacji na danym sterowniku [1-2].
- **Możliwość programowania czasów rejestracji poszczególnych wejść sterownika**
  Taka szczególna cecha sterowników pozwala na znaczne usprawnienie procedur odczytu stanu wejścia, w przypadku gdy np. drgania styków mogą doprowadzić do błędnego działania projektowanego systemu. Zalety takiej właściwości sterownika ujawniają się szczególnie w przypadku zastosowania szybkiego sterownika (o krótkim czasie obiegu pętli). Konieczne w takim przypadku okazuje się takie ustawienie czasów reakcji od poszczególnych wejść, aby dostosować je do aktualnego czasu obiegu pętli i szybkości zmian sygnałów wejściowych [1-2].

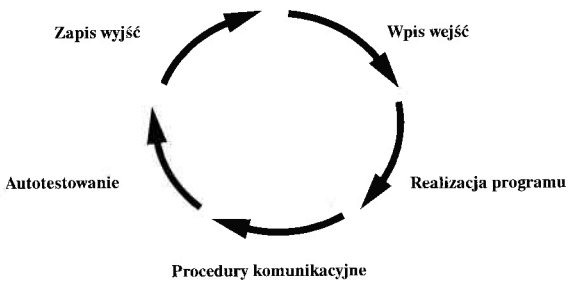
Wyżej opisane szczególne cechy sterowników mogą stanowić o podstawowych wymogach przy realizacji wielu aplikacji z silnymi uzależnieniami czasowymi. Podsumowując, można powiedzieć, że realizacja tego typu aplikacji niesie ze sobą konieczność bardzo skrupulatnego doboru typu sterownika i wykorzystania jego funkcji do granic możliwości.

Problematyka związana z synchronizacją obiegu programu w stosunku do procedur systemowych (autotest, obsługa sieci itp.)

Jak już wspomniano program główny jest tylko jednym z procesów realizowanych w czasie pracy sterownika. Sprzętowe zewnętrzne przerwania mogą „trafiać" w program użytkownika jedynie z pewnym prawdopodobieństwem proporcjonalnym do stosunku czasu wykonywania programu użytkownika do sumy czasów jego i pozostałych procesów. Najczęściej bywa tak, że procedury systemowe nie mogą być przerywane sygnałem zewnętrznego przerwania sprzętowego. W takiej sytuacji może się zdarzyć, że sterownik nie będzie mógł natychmiast zareagować na przychodzące przerwanie czekając na uruchomienie programu w następnym obiegu pętli. Sytuacja taka przy realizacji programu z silnymi uzależnieniami czasowymi może się okazać nie do przyjęcia. Przykładem może tutaj być proces sterowania tyrystorami zasilacza elektrofiltru przemysłowego.

Dla tej aplikacji przerwaniem zewnętrznym jest sygnał impulsowy, który generowany jest przy przejściu przebiegu sinusoidalnego napięcia zasilającego przez zero. Zadaniem sterownika jest natychmiastowe przyjęcie przerwania i możliwie bezzwłoczne wygenerowanie wszystkich niezbędnych sygnałów analogowych oraz cyfrowych sterujących pracą omawianego zasilacza tyrystorowego. Jakiekolwiek zakłócenia - opóźnienia reakcji sterownika na przychodzące przerwanie, może spowodować nieprawidłową pracę tyrystorów, co w konsekwencji może doprowadzić do zachwiania stabilności pracy zasilacza tyrystorowego, powodując szereg różnych konsekwencji elektrycznych.

Celem ilustracji zachodzących zjawisk w procesie przyjmowania przerwana przypomnijmy typowy dla sterowników przemysłowych wykres przedstawiający podstawowy obieg pętli programowej (rys. 1).



Rys. 1. Ilustracja pracy sterownika dla pojedynczego cyklu programowego

Jednym z podstawowych problemów jest tutaj problem związany z czasem reakcji sterownika na zewnętrzne przerwania. Okazuje się, że problem ten staje się podstawowy w przypadku konieczności natychmiastowej obsługi przerwań. Z reguły przerwania występują asynchroniczne w stosunku do głównej pętli programowej. Jeżeli przerwanie wystąpi w czasie testu lub obsługi innej tego typu procedury (sieć DP, panel operatorski itp.) to czas reakcji na przerwanie może się znacznie wydłużyć. Powyższe rozważania przedstawmy na następującym przykładzie.

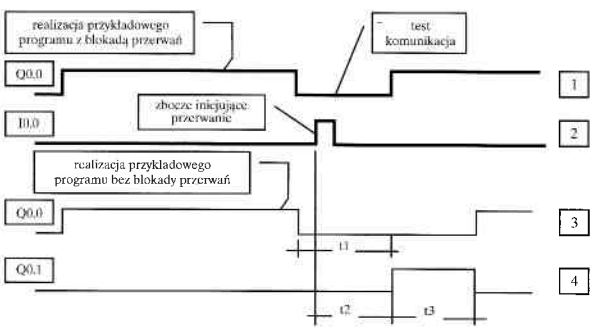
Pętla główna programu OB1			Przerwanie sprzętowe wyzwalane przez narastające zbocze I0.0		
NETWORK 1			NETWORK		
LD	SM0.0		INT	K0	
SI	Q0.0	K1	NETWORK		
.	.	.	LD	SM0.0	
.	.	.	SI	Q0.1	K1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
NETWORK			.	.	.
LD	SM0.0		NETWORK		
RI	Q0.0	K1	LD	SM0.0	
NETWORK			RI	Q0.0	K1
MEND			NETWORK		
			RETI		

Załóżmy, że w sterowniku S7 200 wykonywany jest następujący program:

W powyższych fragmentach programu znaczniki Q0.0, Q0.1 służą do przedstawienia problemów związanych z czasowymi warunkami realizacji poszczególnych części programu:

Q0.0 – reprezentuje obieg programu użytkownika,
Q0.1 – odpowiada realizacji programu obsługującego przerwanie.

Wykres nr 1 (rys. 2) przedstawia realizację głównej pętli programowej i zachowanie się znacznika Q0.0 w przypadku zablokowania systemu przerwań. Na wykresie 2 przedstawiony jest niekorzystny z punktu widzenia czasowego moment wystąpienia przerwania zewnętrznego. W przypadku odblokowania systemu przerwań wydłuża się czas, w którym sygnał Q0.0 przyjmuje wartość 0 (wykres 3). Przyczyną tego przedłużenia jest obsługa zewnętrznego przerwania, które jest obsługiwane po testach i komunikacji (wykres 4). W najgorszym przypadku tzn. wtedy, gdy przerwanie (zmiana na I0.0 z 0 na 1) nastąpi w momencie rozpoczęcia procedur testujących i komunikacyjnych, czas reakcji na przerwanie równy jest czasowi wykonywania procedur testujących i komunikacyjnych (t2=t1). Dla sterowników 57 200 czas przeznaczony na testowanie i procedury komunikacyjne może wynieść kilkadziesiąt procent czasu obiegu programu głównego (OB 1).



t1 – czas realizacji testu i procedur komunikacyjnych
t2 – czas reakcji na przerwanie
t3 – czas obsługi przerwania

Rys. 2. *Wykres czasowy przedstawiający realizację przykładowego programu przy niekorzystnym czasie występowania impulsu przerywającego*

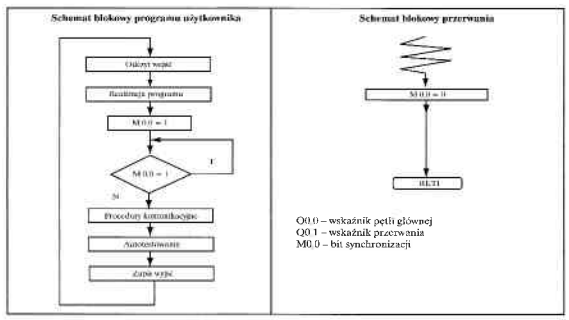
W wielu aplikacjach niedopuszczalne jest tak znaczne przedłużenie czasu reakcji na przerwanie. W omawianym poprzednio praktycznym problemie związanym ze sterowaniem tyrystorów zasilacza elektrofiltru (sterowanie głównie kątem zapłonu) czas obiegu pętli głównej wynosił około 2,5 ms czas obsługi przerwania 8 ms. Przerwania występowały w odstępie 10ms. W przypadku tym istotny staje się fakt synchronizacji głównej pętli programowej (OB 1) z przerwaniem. Idea takiej synchronizacji została przedstawiona na schemacie blokowym rys. 3. Może być ona natychmiast zaimplementowana w przykładowym programie.

Przykładowy program z elementami synchronizacji

Korzyści wynikające z wprowadzenie prostego mechanizmu synchronizacji (bit M0.0) są przedstawione na rysunku nr 4.

Pętla główna programu OB1 *(z elementami synchronizacji)*			Przerwanie sprzętowe wyzwalane przez zbocze narastające I0.0		
NETWORK			NETWORK		
LD	SM0.0		INT	K0	
SI	Q0.0	K1	NETWORK		
.	.	.	LD	SM0.0	
.	.	.	SI	Q0.1	K1
.	.	.	.	.	.
NETWORK			.	.	.
LD	SM0.0		.	.	.
S	Q0.0	K1	NETWORK		
NETWORK			LD	SM0.0	
LBL	K0		R	M0.0	K1
NETWORK			NETWORK		
LD	M0.0		LD	SM0.0	
JMP	K0		RI	Q0.1	K1
NETWORK			NETWORK		
LD	SM0.0		RETI		
RI	Q0.0	K1			
NETWORK					
MEND					

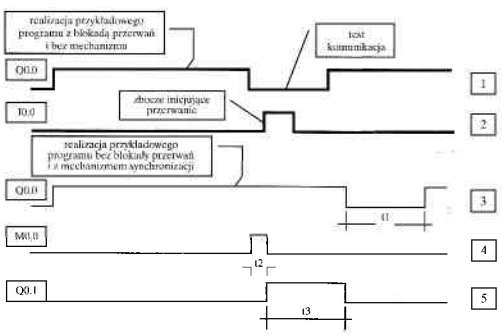
Wykres 1 przedstawia wykres czasowy znacznika (Q0.0) głównej pętli programowej bez mechanizmu synchronizacji w przypadku zablokowania



Rys. 3. *Schemat blokowy przedstawiający ideę synchronizacji głównej pętli programu z odblokowanym systemem przerwań*

przerwań. Na wykresie 2 przedstawiony jest niekorzystny z punktu widzenia czasowego moment wystąpienia przerwania zewnętrznego. W przypadku odblokowania przerwań i wprowadzenia mechanizmu synchronizacji wydłuża się czas, w którym sygnał Q0.0 przyjmuje wartość 1 (wykres 3). Przyczyną tego przedłużenia jest oczekiwanie na skasowanie znacznika synchronizacji M0.0. Znacznik ten ustawiony w pętli głównej programu, kasowany jest dopiero w przerwaniu, co powoduje przedłużenie całej pętli o czas blokady t2 (wykres 4) i czas realizacji przerwania t3 (wykres 5). Dopiero po obsłudze przerwania kończona jest pętla główna programu (patrz program) i uruchamiane są wewnętrzne testy i procedury komunikacyjne.

Przedstawiona idea synchronizacji zapewnia optymalną (najszybszą) obsługę przerwania. Powoduje przedłużenie obiegu głównej pętli programu, co jednak w wielu przypadkach nie jest ograniczeniem. W zrealizowanym układzie główna pętla programowa przerywana była kilkakrotnie, natomiast zaprezentowany mechanizm synchronizacji zapewniał natychmiastową obsługę krytycznego przerwania posiadającego najwyższy priorytet.



Q0.0 – wskaźnik pętli głównej
Q0.1 – wskaźnik przerwania
M0.0 – bit synchronizacji

Rys. 4. *Wykres czasowy przedstawiający realizację przykładowego programu z mechanizmem synchronizacji*

Konkluzja

Współczesne sterowniki przemysłowe częściowo realizują już funkcje specyficzne dla układów bazujących na specjalizowanych szybkich systemach mikroprocesorowych. Warto o tym pamiętać, dokonując wstępnego doboru systemu na jakim będzie realizowany dany projekt. Są to jednak dopiero początki. Najbliższe lata powinny przynieść tutaj jeszcze lepsze efekty.

Literatura

[1] S7 200 Hardware Manual. 1997,
[2] S7 200 Programming Manual 1997
[3] Siemens - Windows Control Center (CD.-ROM) 1998
[4] Siemens - Katalog Wyrobów CA01 1998

Artykuł recenzowany