Marek ŚNIEŻEK
POLITECHNIKA RZESZOWSKA, KATEDRA INFORMATYKI I AUTOMATYKI

Bezpieczna platforma sprzętowa dla aplikacji opisanych w języku FBD

Dr inż. Marek ŚNIEŻEK

Dyplom magistra inżyniera uzyskał w 1990 roku na Wydziale Elektrycznym Politechniki Rzeszowskiej. Pracuje w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Stopień doktora inżyniera uzyskał w roku 1998 w Fernuniversität - Gesamthochschule w Hagen (Niemcy). Zajmuje się konstrukcją i oprogramowaniem sterowników mikroprocesorowych, komunikacją w systemach rozproszonych, sterowaniem bezpiecznym, strukturami FPGA.

e-mail: sniezekm@prz-rzeszow.pl

Streszczenie

W pracy przedstawiono programowalny sterownik logiczny, zachowujący się w sposób bezpieczny. Zachowanie to obejmuje normalną pracę, podczas której wymaga się poprawnego sterowania, jak również stan awarii, w którym wyjścia muszą być automatycznie wyzerowane. Algorytm sterowania jest opisany metodą bloków funkcyjnych FBD i SFC zgodną z normą IEC-61131. Zastosowana architektura sprzętowa sprzyja podzieleniu oprogramowania na dwie części. Pierwsza część - stała, niezależna od aplikacji - obejmuje bibliotekę bloków funkcyjnych. Część druga - zmienna, bezpośrednio zależna od aplikacji - zawiera strukturę połączeń bloków. Obydwie części są wykonywane przez osobne procesory. Do badania poprawności stałej części programu zaproponowano formalną metodę wykorzystującą logikę wyższego rzędu HOL. Część zmienną bada się stosując metodę zróżnicowanej retranslacji.

Abstract

To architecturally support the programming of safety related control applications in the graphical language Function Block Diagram and the verification of such software meeting the requirements of Safety Integrity Level 3, a dedicated, low complexity execution platform is presented. Its hardware is fault detecting to immediately initiate emergency shut-downs in case of malfunctions. By design, there is no semantic gap between the programming and machine execution levels, enabling the safety licensing of application software by extremely simple, but rigorous methods, viz., diverse back translation and inspection. Operating in a strictly periodic fashion, the platform exhibits fully predictable real time behaviour.

Słowa kluczowe: sterowanie bezpieczne, sterownik logiczny, PLC, język bloków funkcyjnych, formalna weryfikacja.

Keywords: safety related control, Safety Integrity Level 3, Function Block Diagrams, software verification, programmable logic controller.

1. Wstęp

Sterowniki bezpieczne są najczęściej wykorzystywane w układach zabezpieczeń, do odpowiedzialnego sterowania logicznego, sygnalizacji przekroczeń i alarmowania. W dotychczasowych zastosowaniach [1] stosowano sterowniki budowane z nieprogramowalnych modułów sprzętowych, tzw. hard-wired. Moduł realizuje prostą funkcję, jak bramka, przerzutnik [2]. Wykonany jest z elementów dyskretnych lub o małym stopniu scalenia. Każda awaria powoduje ustawienie wyjść w stan bezpieczny. Moduły poddawane są badaniom po czym uzyskują certyfikat.

W sterownikach programowalnych o funkcjonalności decyduje oprogramowanie. Jednym z typowych rozwiązań jest wydzielenie w programie bloków funkcyjnych, takich jak AND, OR itd. [3]. Algorytm sterowania jest reprezentowany przez odpowiednią strukturę złożoną z tych bloków. Połączenia są programowe, zatem łatwiej jest wprowadzić ewentualne zmiany. Do przygotowania struktury bloków używa się edytorów graficznych.

Przy opracowywaniu bezpiecznego sterownika programowalnego powstają trzy zasadnicze problemy związane: z konstrukcją sprzętu, potrzebą weryfikacji oprogramowania i certyfikacji.

Stan bezpieczny powinien być osiągany automatycznie po rozpoznaniu awarii w sposób niezależny od wykonywanego programu. Konstruowanie sterownika bezpiecznego jest zadaniem nietypowym z następujących powodów:
- w układach scalonych nie wyróżnia się stanu bezpiecznego,
- w czasie normalnej pracy musi być na bieżąco prowadzone sprawdzanie, że sterownik jest sprawny. Nie może to być zrealizowane metodami programowymi, ale przy zastosowaniu odpowiedniej struktury sterownika.

W przeciwieństwie do sprzętu, błędy oprogramowania mają z zasady naturę systematyczną. Są powodowane przez niewłaściwą specyfikację algorytmu, błędne zapisanie go w programie, mogą zostać wniesione przez kompilatory. Błędy oprogramowania można usunąć przez kompletne testy, ale metoda ta jest użyteczna tylko w przypadku prostych programów. Badaniu poprawności oprogramowania sprzyja modularyzacja przez wydzielenie funkcji programowych. W sterowniku są to bloki funkcyjne.

Zaletą sterownika programowalnego jest możliwość szybkiej zmiany algorytmu. W przypadku sterowania bezpiecznego program musi być zweryfikowany przez uprawnioną instytucję. Wiążą się z tym znaczne koszty. Należy więc dążyć do takiego skonstruowania sterownika, aby część oprogramowania niezależna od konkretnej aplikacji mogła być zweryfikowana tylko raz. Natomiast druga część zależna od aplikacji musi być weryfikowana każdorazowo. Powinna ona być jak najmniej złożona.

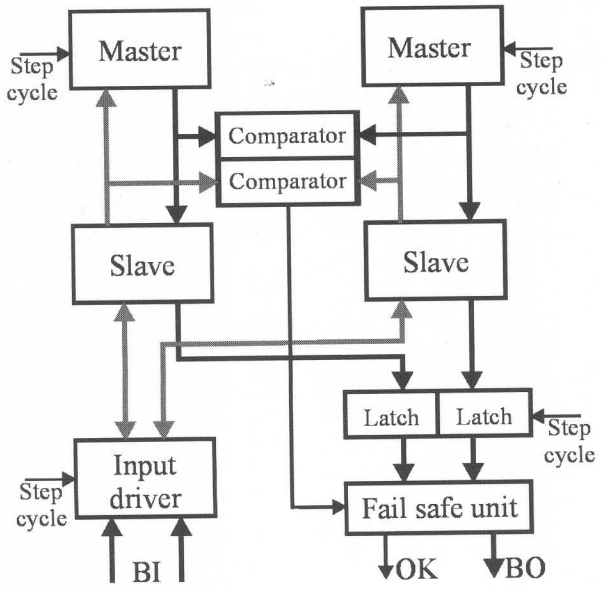
2. Architektura sterownika bezpiecznego

Sterowanie procesem jest realizowane w oparciu o program, którego algorytm został przedstawiony w formie schematu złożonego z bloków funkcyjnych. Oprogramowanie sterownika bezpiecznego zostało podzielone na dwie części:
- zmienną - obejmuje strukturę połączeń bloków funkcyjnych,
- stałą - zawiera bibliotekę funkcji bloków.

Do ich wykonywania wykorzystano procesory:
- master - organizujący obliczenia przez wywołania bloków zgodnie ze schematem,
- slave - wykonujący funkcje przyporządkowane blokom [4].

Program procesora slave nie zmienia się wraz z zastosowaniem, dzięki czemu podlega jednorazowej weryfikacji. Program mastera zależy natomiast od aplikacji. Jest napisany w specjalnie opracowanym prostym języku, co ma ułatwić weryfikację. Język ten zawiera tylko dwie grupy instrukcji. Pierwsza obejmuje kilka instrukcji służących do przesyłania danych. W drugiej grupie jest tylko jedna instrukcja, która wykonuje skok, umożliwiając ponowienie aktywnej sekwencji bloków lub przejście do kolejnej. Schemat blokowy sterownika przedstawia rys. 1, opis zawarto w [5, 6].

Do wymiany danych procesory master i slave wykorzystują dwukanałową magistralę. Master i slave wykonują programy w sposób zsynchronizowany. Master, podejmując obsługę bloku, inicjuje wykonanie funkcji z nim związanej w procesorze slave. Odbywa się to przez wysłanie identyfikatora funkcji bloku i parametrów, którymi są wejścia bloku oraz dotychczasowe stany wewnętrzne (jeśli blok takie posiada). Na podstawie identyfikatora slave rozpoznaje funkcję, kompletuje parametry, po czym oblicza wyjścia i nowe stany wewnętrzne. Wartości te zostają odesłane do mastera, który je przechowuje. Wyjścia bloku mogą być wykorzystane jako wejścia innych bloków. Z założenia pamięć procesora slave jest podręczna i rezerwowana tylko na czas obsługi funkcji. Cykl pracy sterownika jest wyznaczany zmianami sygnału step cycle, synchronizującego pracę mastera.



Rys. 1. Schemat blokowy sterownika bezpiecznego
Fig. 1. Block scheme of the fail-safe controller

Wykrywanie uszkodzeń sprzętowych jest możliwe dzięki zastosowaniu drugiej pary master-slave wykonującej ten sam program i kontroli zgodności wszystkich danych wymienianych przez pary procesorów. Porównania w obu magistralach realizują szybkie komparatory comparator. Ponieważ od poprawności ich pracy zależy prawidłowość porównań, zostały one wykonane w bezpiecznej technologii [5]. Sygnały wyjściowe komparatorów są badane w bezpiecznym układzie fail safe unit [6].

Sterownik komunikuje się z otoczeniem przez porty wejściowe i wyjściowe dołączone do procesorów slave. Wszystkie sygnały są binarne. Stan wejść obiektowych jest zapamiętywany na początku cyklu i przechowywany w układzie input driver. Na żądanie procesorów master każdy slave odczytuje z niego dane.

Stany wyjść są pamiętane w czasie trwania cyklu w sprzętowych buforach obydwu procesorów slave i przepisywane na końcu cyklu do rejestrów latch. Są one porównywane i równocześnie stają się dostępne dla otoczenia sterownika. Porównania i obsługa wyjść są prowadzone w układzie fail safe unit. W nim także, na podstawie sygnałów poprawności ze wszystkich modułów, jest tworzony sygnał statusu pracy OK. W razie awarii zapewnia on ustawienie wyjść w stan wyłączenia, będący z założenia stanem bezpiecznym sterownika.

3. Procesory master i slave

Master nie wykonuje obliczeń, a jedynie steruje przepływem danych. Przetwarzanie zachodzi w procesorze slave, który zgodnie z zadaną przez mastera funkcją przelicza otrzymane dane i zwraca wyniki. W masterze odbywa się to przez wykonywanie instrukcji przesłania. Jest to instrukcja, której wykonanie oznacza wysłanie danej z pamięci procesora master do slave lub jej odebranie z procesora slave i umieszczenie w pamięci mastera. Z kolei makroinstrukcja jest następującym ciągiem instrukcji przesłań:

1. Wysłanie do procesora slave identyfikatora funkcji.
2. Wysłanie danych wejściowych (grupa instrukcji).
   (tutaj następuje obliczenie funkcji bloku w procesorze slave)
3. Odbiór danych wyjściowych (grupa instrukcji).

Identyfikator funkcji definiuje sposób przetwarzania w procesorze slave oraz spodziewaną liczbę danych wejściowych. Ilość i kolejność instrukcji przesłań w krokach 2, 3 zależą od funkcji. Wykonanie makroinstrukcji prowadzi w efekcie do wyznaczenia nowych wartości wyjść i stanów wewnętrznych bloku.

Segment programu jest ciągiem makroinstrukcji służącym wykonaniu grupy bloków. Kolejność makroinstrukcji wynika z przyjętego porządku bloków.

Wprowadza się instrukcję STEP (skok warunkowy) oddzielającą segmenty i sterującą ich wykonaniem. W danej chwili jest aktywny tylko jeden segment. Po jego zakończeniu możne być on powtórzony lub następuje przejście do innego segmentu.

Zgodnie z ideą SFC (Sequential Function Chart) segmenty programu tworzą kroki, a instrukcje STEP przejścia. Zbiór wszystkich segmentów i instrukcji STEP umożliwiający wykonanie zadania sterowania tworzy program. W typowych programach występuje jeden segment zakończony instrukcją STEP.

Zakłada się dodatkowo, że przejścia są możliwe tylko w chwilach czasu, wyznaczanych zmianami sygnału taktującego step cycle. Oznacza to, że instrukcja STEP oprócz skoku wymusza chwilowe oczekiwanie procesora master. Pojawienie się zmiany step cycle w trakcie wykonywania instrukcji innej niż STEP jest błędem przekroczenia czasu (tzw. run time error).

Slave prowadzi wszystkie obliczenia zlecane mu przez procesor master. O ile master może być uznany za procesor nietypowy, bo wykonuje tylko instrukcje przesłań, o tyle slave jest standardowy. Slave realizuje trzy zadania:

- komunikuje się z procesorem master, który zleca mu obliczenia i odbiera wyniki,
- wykonuje funkcje bloków,
- kontaktuje się z otoczeniem.

Slave wykonuje obliczenia na podstawie zadanego przez mastera algorytmu i przekazanych argumentów. Po zakończeniu przetwarzania zwraca wszystkie wyniki. Program procesora slave został podzielony na funkcje, które odpowiadają możliwym blokom funkcyjnym [3]. Zadawanie funkcji z poziomu mastera odbywa się przez przesłanie do procesora slave identyfikatora funkcji. Na tej podstawie slave ocenia, ile bajtów danych będzie przesłanych. Po ich odbiorze i umieszczeniu w buforze wejściowym jest wywoływana funkcja. Wyniki obliczeń zostają umieszczone w buforze wyjściowym, a program zarządzający po zakończeniu funkcji wysyła je do procesora master. Dane te po stronie mastera są rozpoznawane przez odpowiednie instrukcje i umieszczane w zadanych miejscach pamięci. Miejsca te wynikają bezpośrednio ze schematu połączeń bloków i przyjętej mapy odwzorowania „wyjście bloku - adres w pamięci".

Slave po wykonaniu funkcji usuwa z pamięci wszystkie wytworzone dane robocze. Dlatego za każdym razem master musi przesłać komplet danych wejściowych i odebrać wyniki.

Pamięć ROM zawiera funkcje, dane organizacyjne o funkcjach i program zarządzający. Zakłada się, że zestaw funkcji jest niezmienny dla danej klasy zastosowań, tak aby tylko raz podlegał weryfikacji.

4. Sygnały obiektowe

Sterownik komunikuje się z otoczeniem przez wejścia i wyjścia obiektowe, które są umieszczone po stronie procesora slave.

Odczyt wejść jest realizowany, gdy sygnał step cycle jest na poziomie wysokim, a ich stan jest zapamiętywany w układzie input driver. W tym czasie master wykonuje instrukcję STEP i nie odwołuje się do procesora slave. Gdy step cycle zmieni się na poziom niski, rozpoczyna się obsługa bloków funkcyjnych. Master w celu odczytania wejść inicjuje odpowiednie funkcje po stronie procesora slave. Ten odwołuje się do układu input driver, skąd odczytuje dane.

Wyjścia sterownika muszą być ustawione na końcu cyklu, podczas gdy slave ustawia je zgodnie z porządkiem wywoływanych funkcji. Stąd dla wyjść jest konieczne podwójne buforowanie, gdzie rejestr latch (rys. 1) przejmuje ustawiane przez procesor slave wartości w chwili zmiany sygnału taktującego step cycle.

5. Bezpieczne porównania

Głównym problemem w programowalnym sterowniku bezpiecznym jest wykrywanie uszkodzeń sprzętowych, które mogą wystąpić w czasie pracy.

Zastosowano sprzętową detekcję uszkodzeń. Wszystkie dane z dwóch równoległych, odmiennych sprzętowo ale równorzędnych torów obliczeniowych master-slave są porównywane na bieżąco.

Kierowano się założeniem, że uszkodzenia objawiają się rozbieżnością wyników. Gdyby uszkodzenie nie wpłynęło na wyniki, nie mogłoby być wykryte. Do porównywania zgodności wyników służy komparator, którego szczegóły techniczne omówiono w [5].

Komunikację między procesorami master i slave zrealizowano za pomocą magistrali podwójnie buforowanej układami FIFO. Analogiczny układ zastosowano dla drugiego kierunku. Ponieważ procesory master działają niezależnie, mogą istnieć przesunięcia czasowe przy zapisie danych do FIFO. W celu zapewnienia prawidłowości porównań zastosowano dodatkowe rejestry, na wyjściach których dane pojawiają się synchronicznie. Wpisywaniem danych do rejestrów steruje układ, wykorzystujący informację o zapełnieniu FIFO. Dane zatrzaśnięte w rejestrach są porównywane przez komparator generujący sygnał poprawności danych. W przypadku różnicy w danych lub uszkodzenia komparatora sterownik jest zatrzymywany. Jeśli jeden z procesorów master z powodu uszkodzenia nie wyśle kompletu danych, to drugi-sprawny na końcu cyklu zgłosi błąd przekroczenia czasu.

Drugim fragmentem sterownika, gdzie jest niezbędne prowadzenie bezpiecznych porównań, są wyjścia obiektowe generowane przez oba układy slave.

W dodatkowym układzie bezpiecznym jest tworzony globalny sygnał poprawności OK. Jest on podawany zwrotnie do wszystkich procesorów i w razie wykrycia błędu, powoduje ich zatrzymanie i ustawia wyjścia sterownika w stan bezpieczny.

6. Bloki funkcyjne

Opisywany sterownik jest programowany w językach schematów bloków funkcyjnych (FBD - Function Block Diagram) oraz grafu sekwencji (SFC - Sequential Function Chart) zgodnych z normą międzynarodową IEC 61131-3 [3]. Przygotowanie oprogramowania sterownika składa się z dwóch etapów:

- ustalenie i zbudowanie biblioteki bloków,
- wykonanie połączeń między wybranymi blokami.

Biblioteka bloków jest tworzona raz i wiąże się z implementacją bloków funkcyjnych w procesorze slave. Każdemu blokowi odpowiada funkcja programowa. Biblioteka obejmuje następujące grupy bloków: matematyczne, logiczne, wyboru, porównania, liczniki, przerzutniki, układy czasowe, sygnały procesowe, konwersje typów.

Drugi etap programowania jest wykonywany przy nowym wdrożeniu i jest związany z przygotowaniem oprogramowania procesora master. Polega ono na wyborze z biblioteki odpowiednich bloków oraz ich połączeniu. Połączeniom na schemacie odpowiadają programowe „przepływy" wartości zmiennych. Wyjścia bloku są dostępne dla wejść innych bloków.

Następnie wykorzystuje się dodatkowe narzędzie programowe, które na podstawie schematu tworzy program mastera. Każdemu blokowi odpowiada makroinstrukcja w programie zapisanym tekstowo. Z każdym wyjściem bloku jest związane miejsce w pamięci mastera, w którym przechowuje się jego wartość. Wykorzystują ją inne bloki. Stany wewnętrzne bloków nie są udostępniane innym blokom.

7. Weryfikacja poprawności oprogramowania

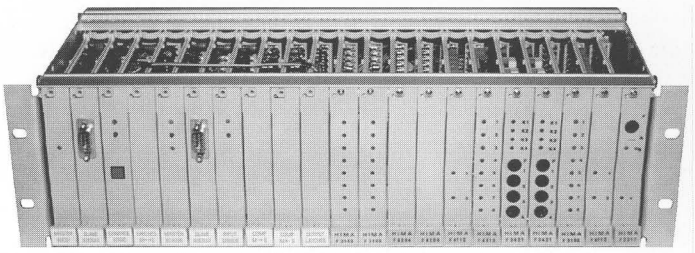
Istotnym elementem wpływającym na bezpieczeństwo pracy sterownika jest poprawność jego programów. Zastosowana architektura pozwala wykrywać ewentualne błędy sprzętowe, a nie programowe. Gwarancją poprawności programów jest pomyślnie przeprowadzona formalna weryfikacja. W tym celu przeprowadzono formalną weryfikację funkcji bloków biblioteki procesora slave. Wybrano prostą teorię typów i obejmującą ją logikę wyższego rzędu HOL (Higher-Order Logic) oraz wspomagający dowodzenie twierdzeń system Isabelle [7].

Do weryfikacji poprawności programu mastera, wyrażonego za pomocą schematu złożonego z połączonych bloków funkcyjnych (FBD) wystarczy wykazać, że kod binarny wykonywany przez procesor zawiera poprawnie zbudowane makroinstrukcje, a powodowany przez nie „przypływ" danych ściśle odpowiada schematowi pierwotnemu. Do tego celu można wykorzystać metodę zróżnicowanego tłumaczenia zwrotnego (diverse back translation). Metoda ta polega na odczytaniu kodu programu umieszczonego w pamięci sterownika, rozpoznaniu rozkazów (disasemblacja) i odtworzeniu kodu programu (dekompilacja). Z kolei na tej podstawie próbuje się wywnioskować jakie zadania program ten wykonuje i czy są one zgodne z pierwotnym zamierzeniem. Aby metoda była wiarygodna proces ten należy przeprowadzić w kilku zespołach pracujących niezależnie.

Mimo, że metoda jest nieformalna, zyskała ona akceptację instytucji przyznających certyfikaty bezpieczeństwa i została pomyślnie zastosowana w kilku przypadkach, np. TÜV Rheinland - Kolonia, Niemcy, Instytut Energetyki - Halden, Norwegia.

W przypadku stosowania procesorów uniwersalnych metoda zróżnicowanego tłumaczenia zwrotnego jest bardzo uciążliwa, czasochłonna i kosztowna. Wynika to z istnienia bariery między rozkazami procesora, a wyrażeniami opisującymi problem, formułowanymi najczęściej w innym języku.

Zupełnie inaczej przedstawia się zastosowanie tej metody w przypadku prezentowanego sterownika, przedstawionego na rys. 2. Przyjęta architektura i opracowany język mastera pozwalają zachować bezpośredni związek między rozkazami procesora, instrukcjami języka i zadaniami przedstawianymi w formie bloków funkcyjnych.



Rys. 2. Prototyp sterownika bezpiecznego
Fig. 2. Fail-safe controller prototype

8. Podsumowanie

Istnieje potrzeba konstruowania bezpiecznych programowalnych sterowników logicznych. W artykule pokazano, iż w obecnym stanie techniki jest możliwe skonstruowanie takiego sterownika. Stosowane metody weryfikacji zarówno sprzętu, jak i oprogramowania są wystarczające do wykazania jego poprawności.

Wybrana metoda graficznego programowania w oparciu o FBD/SFC wydaje się być najbardziej odpowiednia dla sterownika bezpiecznego o opisanej architekturze, gdzie możliwie największa część oprogramowania powinna być niezmienna.

9. Literatura

[1] N. Storey: Safety-Critical Computer Systems. Addison-Wesley, New York 1996.

[2] Paul Hildebrandt GmbH & Co. KG: Main Catalogue - The HIMA-Planar-System. Brochure HK 90.11. Brühl, 1991

[3] IEC International Standard 61131-3. Programmable Controllers, Part 3: Programming Languages. Geneva: International Electrotechnical Commission, 1992

[4] W. A. Halang, S.-K. Jung, B. J. Krämer, J. J. Scheepstra: A Safety Licensable Computing Architecture. World Scientific, Singapore 1993.

[5] M. Śnieżek, W. A. Halang: Bezpieczny programowalny sterownik logiczny. Oficyna Wydawnicza Politechniki Rzeszowskiej. Rzeszów 1998.

[6] M. Śnieżek, J. von Stackelberg: A fail safe programmable logic controller. Annual Reviews in Control. Vol. 27, pp. 63-72, 2003

[7] L. C. Paulson: Isabelle. A Generic Theorem Prover. Lecture Notes in Computer Science, Vol. 828. Springer-Verlag, 1994.

Title: Safety execution framework for FB applications

Artykuł recenzowany