Projekt i budowa uniwersalnego sterownika programowalnego

Sebastian Wójcicki, Tomasz Rutkowski

Wydział Elektrotechniki i Automatyki, Politechnika Gdańska

Streszczenie: Artykuł opisuje projekt i realizację niskobudżetowego ale zarazem funkcjonalnego i uniwersalnego kompaktowego sterownika programowalnego PLC (ang. *Programmable Logic Controller*) oraz dedykowanej aplikacji narzędziowej umożliwiającej jego elastyczne oprogramowanie. Przedstawiany sterownik bazuje na jednostce centralnej w postaci 32-bitowego mikrokontrolera ARM firmy STMicroelectronics oraz wyposażony jest w standardowe peryferia wykorzystywane w przemysłowych instalacjach technologicznych tj.: we/wy cyfrowe, we/wy analogowe czy interfejs enkodera. Do komunikacji sterownika z komputerem PC lub innymi urządzeniami przeznaczone są standardowe interfejsy: USB, RS-232 lub magistrala CAN (ang. *Controller Area Network*). Dedykowana sterownikowi aplikacja narzędziowa napisana została w języku C# i pracuje w środowisku Windows. Umożliwia ona między innymi: tworzenie programów sterujących w języku graficznym FBD (ang. *Function Block Diagram*), programowanie sterownika, prostą diagnostykę oraz podgląd zmiennych w trakcie pracy sterownika (tryb on-line). W celu śledzenia wartości wybranych zmiennych można wykorzystać wbudowany wyświetlacz LCD a za pomocą dostępnych na obudowie przycisków dokonać np. „strojenia" wybranych parametrów wgranego do sterownika algorytmu sterującego.

Słowa kluczowe: PLC, mikrokontroler, przemysłowe systemy sterowania

1. Wprowadzenie

Ciężko wyobrazić sobie współczesne przemysłowe instalacje technologiczne, małe czy duże, z układami sterowania, które nie są wyposażone w sterowniki programowalne. Na rynku można znaleźć sterowniki różnego typu (modułowe, kompaktowe lub wbudowane) dystrybuowane przez duże i znane firmy jak i mniejsze niejednokrotnie lokalne, które sprzedają swoje produkty w znacznie korzystniejszej cenie. Trend ten jest tym wyraźniejszy im dostępność do układów mikroprocesorowych oferowanych przez ich producentów jest coraz większa a one same charakteryzują się coraz większą mocą obliczeniową, bogatszym wyposażeniem i korzystniejszym stosunkiem ceny do możliwości. Niniejszy artykuł wpisuje się również w ten trend, ponieważ opisuje projekt i realizację niskobudżetowego ale zarazem funkcjonalnego i uniwersalnego kompaktowego sterownika programowalnego PLC oraz dedykowanej aplikacji narzędziowej umożliwiającej jego elastyczne i efektywne programowanie.

Projektowanie sterownika programowalnego jest procesem złożonym, w którym bierze udział wielu wyspecjalizowanych w różnych dziedzinach inżynierów. W przypadku planów sprzedaży takiego urządzenia na wolnym rynku jednym z podstawowych czynników, którymi należy się kierować jest norma związana ze sterownikami programowalnymi – IEC 61131. Również autorzy niniejszego artykułu dołożyli wszelkich starań by wypełnić zalecenia wspomnianej normy w kontekście jej części 1 (standardowych własności sterowników programowalnych) i części 3 (programowanie sterownika w jednym z języków graficznych – FBD).

2. Warstwa sprzętowa uniwersalnego sterownika programowalnego

2.1. Specyfikacja sterownika

Jednostkę obliczeniową sterownika stanowi 32-bitowy mikrokontroler ARM STM32F103VGT6 firmy STMicroelectronics [1]. Pracuje on z częstotliwością 72 MHz i wyposażony jest w 1 MB pamięci flash, 96 kB RAM, przetworniki ADC i DAC oraz interfejsy komunikacyjne takie jak USB, CAN, 5×UART. Tak bogate wyposażenie mikrokontrolera daje duże możliwości przy projektowaniu uniwersalnego sterownika programowalnego. Założenia projektowe i specyfikację urządzenia przedstawiono w tab. 1. Natomiast schemat ideowy sprzętowych modułów funkcyjnych sterownika przedstawiono na rys. 1.

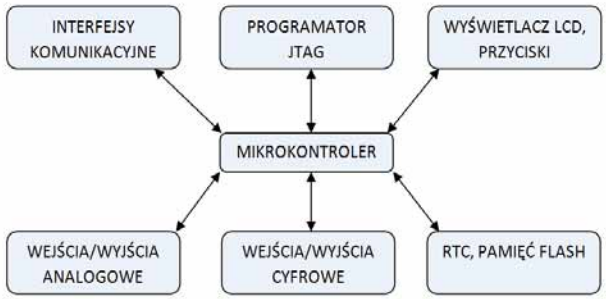
Tab. 1. Specyfikacja sterownika
Tab. 1. Controller specification

Zasilanie:	12/24 V DC
Wejścia cyfrowe:	3 wejścia 10–30 V 3 wejścia TTL
Wyjścia cyfrowe:	4 wyjścia (typ ujście)
Wejścia analogowe:	2 wejścia napięciowe o zakresie 0–10 V
Wyjścia analogowe:	2 wyjścia napięciowe o zakresie 0–10 V
Interfejsy komunikacyjne:	– USB – RS-232 – CAN
Dodatkowe dane:	– zegar czasu rzeczywistego podtrzymywany bateryjnie (około 5 lat) – interfejs enkodera inkrementalnego

	10–30 V – wyświetlacz LCD – 4 przyciski konfigracyjne – obsługa protokołu Modbus RTU – 80 kB pamięci RAM oraz 1 MB FLASH dla programu użytkownika – 256 kB nieulotnej pamięci EEPROM

Części sprzętową sterownika można podzielić na następujące moduły obsługiwane przez mikrokontroler (rys. 1):

- moduł wejść/wyjść analogowych,
- moduł wejść/wyjść cyfrowych,
- moduł zegara czasu rzeczywistego i pamięci FLASH,
- moduł interfejsów komunikacyjnych,
- moduł programatora JTAG,
- moduł wyświetlacza LCD oraz przycisków.

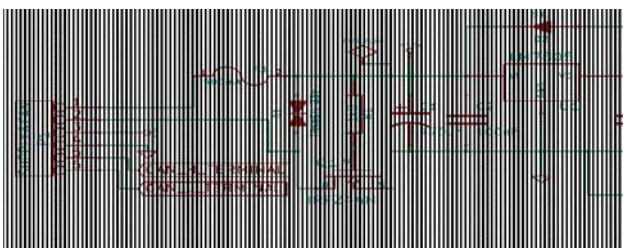


Rys. 1. Schemat ideowy sprzętowych modułów funkcjonalnych sterownika
Fig. 1. Schematic diagram of the controller hardware functional modules

2. 2. Projekt warstwy sprzętowej

2.2.1. Układ zasilania

Zastosowany w prototypowym układzie sterownika mikrokontroler jest zasilany napięciem 3,3 V, sekcja zasilania jest oparta o liniowe stabilizatory napięcia (rys. 2). Sterownik został wyposażony w zabezpieczenie przed odwrotną polaryzacją zasilania [2]. Zostało to osiągnięte przez szeregowe podłączenie „do minusa" tranzystora MOS typu N. W klasycznych układach wykorzystuje się uniwersalne diody np. 1N4007 [3], ale występuje na nich większy spadek napięcia niż ma to miejsce w przypadku tranzystorów MOS.

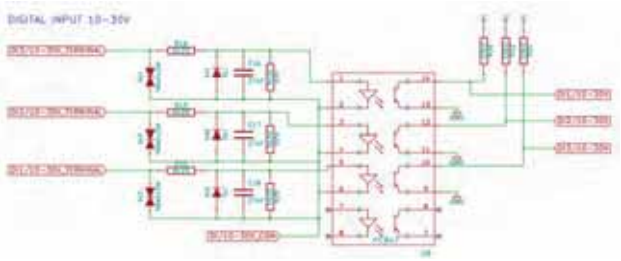


Rys. 2. Schemat ideowy układu zasilania
Fig. 2. Schematic diagram of the supply circuit

Mikrokontroler operuje na sygnałach w zakresie 0–3,3 V. Natomiast uniwersalny sterownik powinien pracować z sygnałami o standardowych poziomach napięć spotykanych w przemysłowych instalacjach sterowania [4]. Dodatkowo istotnym jest zaprojektowanie sterownika w taki sposób, aby jak najbardziej uodpornić go na błędy użytkownika, np. w postaci odwrotnej polaryzacji sygnału podanego na wejście cyfrowe, zwarcia przewodów wychodzących z wyjścia analogowego i masy, przekroczenia zakresu napięciowego wejścia analogowego itp. Z tych powodów dla wejść/wyjść cyfrowych oraz analogowych zaprojektowano układy kondycjonujące, które zabezpieczają sterownik przed nieprawidłowym użytkowaniem i konwertują poziomy napięć sygnałów przesyłanych między otoczeniem sterownika, a mikrokontrolerem [5]. Układy kondycjonujące chronią również przed zniszczeniem sam mikrokontroler, ewentualne uszkodzenie np. transoptora jest tańsze i o wiele prostsze w naprawie niż wymiana mikrokontrolera. Cały układ elektroniczny sterownika został zaprojektowany w taki sposób, aby sterownik w stanach przejściowych (np. reset mikrokontrolera, początkowa faza inicjalizacji przy włączeniu zasilania) nie wystawiał na żadnym z wyjść sygnałów w stanie wysokim.

2.2.2. Układ wejść cyfrowych

Do dyspozycji użytkownika jest 6 optoizolowanych wejść cyfrowych, w tym 3 pracujące zgodnie z standardem TTL. Podanie zbyt wysokiego napięcia (ok. 30 V) na wejście TTL nie powinno spowodować żadnych uszkodzeń, ale nie jest to gwarantowane (prąd transoptora jest bliski wartości granicznej). Na rys. 3 pokazano schemat zestawu wejść cyfrowych 10–30 V. Każde wejście cyfrowe posiada zabezpieczenie przeciwprzepięciowe oraz filtr dolnoprzepustowy o częstotliwości granicznej ok. 1 kHz.



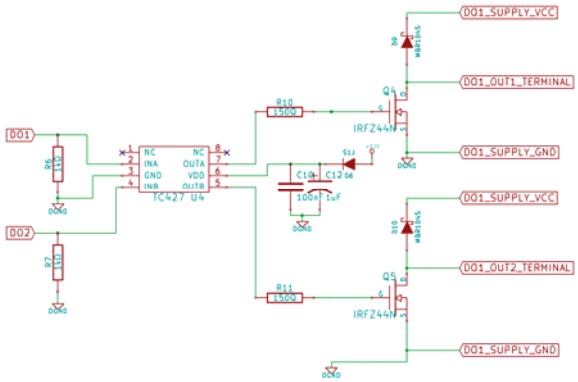
Rys. 3. Schemat ideowy wejść cyfrowych
Fig. 3. Schematic diagram of the digital inputs

2.2.3. Interfejs enkodera inkrementalnego

Dodatkowo sterownik wyposażono w różnicowe wejścia wysokoczęstotliwościowe, gdzie zastosowano szybkie transoptory i możliwe jest podłączenie enkodera inkrementalnego.

2.2.4. Układ wyjść cyfrowych

Wyjścia cyfrowe sterownika (rys. 4) zbudowano na tranzystorach MOSFET IRFZ44N [6] oraz sterownikach tranzystorów TC427 [7]. Zastosowany mikrokontroler może wystawić na swoim wyjściu tylko 3,3 V, co jest niewystarczające do wysterowania zastosowanych tranzystorów.

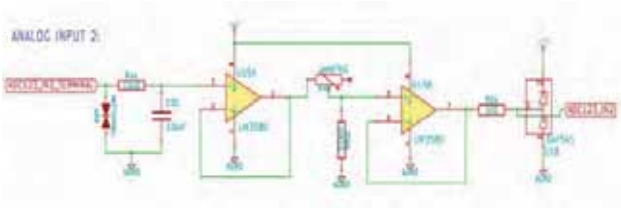


Rys. 4. Schemat ideowy wyjścia cyfrowego
Fig. 4. Schematic diagram of the digital output

Z tego powodu wykorzystano układy TC427, które operują napięciem z zakresu 0-12 V i charakteryzują się większą wydajnością prądową. Pozwala to na szybsze ładowanie i rozładowywanie bramek tranzystorów, co przekłada się na mniejsze straty i wydzielane ciepło na obudowie elementów energoelektronicznych.

2.2.5. Układ wejść analogowych

Zastosowany mikrokontroler ma trzy 12-bitowe przetworniki analogowe-cyfrowe, których zakres pomiarowy wynosi 0–3,6 V (zakres ten jest zbyt wąski). Na rys. 5 przedstawiono tor pomiarowy sterownika. Składa się on z ochrony przeciwprzepięciowej, filtru dolnoprzepustowego o częstotliwości granicznej 48 Hz, wtórników napięciowych i dzielnika napięciowego. Sterownik umożliwia zatem pomiar sygnału napięciowego z zakresu 0–10 V. Dodatkowo dodano zabezpieczenie w postaci dwóch diod Schottkiego, które chronią bezpośrednio port mikrokontrolera przed podaniem zbyt dużego napięcia. Do takiej sytuacji może dojść np. przy ustawieniu nieprawidłowej rezystancji potencjometru w dzielniku napięciowym w trakcie montażu.

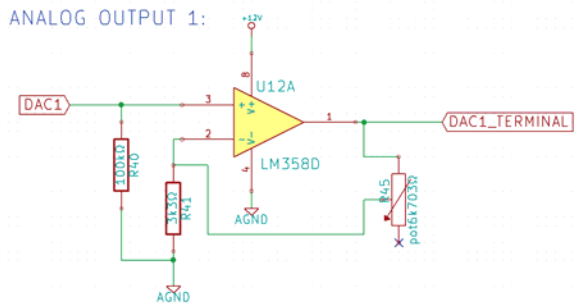


Rys. 5. Schemat ideowy wejścia analogowego
Fig. 5. Schematic diagram of the analog input

2.2.6. Układ wyjść analogowych

Wyjścia analogowe podobnie jak wejścia, operują napięciem w zakresie 0–3,3 V. Zastosowanie wzmacniacza operacyjnego LM358D [8] o odpowiednim wzmocnieniu napięciowym pozwala na wystawienie napięcia w zakresie 0-10 V. Dodatkowo układ LM358D został wyposażony w zabezpieczenie przeciwzwarciowe, więc przypadkowe zwarcia przewodów przy podłączaniu sterownika do innych układów nie spowoduje żadnych uszkodzeń.

Sterownik został wyposażony również w system kontroli wartości napięcia zasilającego. Mikrokontroler STM32 można skonfigurować w taki sposób, że jest w stanie generować przerwanie w przypadku przekroczenia pewnej wartości napięcia mierzonego w wybranym przetworniku A/C. W przypadku awarii zasilania, zastosowanie kondensatorów o dużej pojemności daje czas na przeprowadzenie pewnych operacji, np. odczyt czasu z RTC i jej zapis do nieulotnej pamięci FLASH albo wysłanie krótkich komunikatów do urządzeń współpracujących ze sterownikiem.



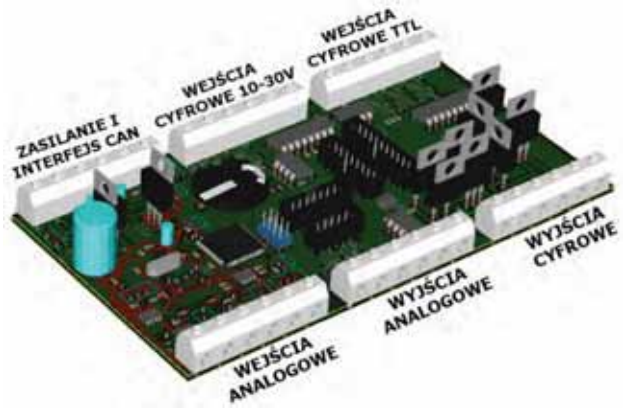
Rys. 6. Schemat ideowy wyjścia analogowego
Fig. 6. Schematic diagram of the analog output

2.2.7. Interfejsy komunikacyjne

Sterownik wyposażono w cyfrowe interfejsy komunikacyjne takie jak RS-232, USB oraz magistralę CAN. Szczególnie zastosowanie magistrali CAN otwiera duże możliwości rozbudowy funkcjonalności sterownika programowalnego działającego „lokalnie" w kierunku układów sterowania z rozproszonymi układami we/wy (np. moduł z dodatkowymi wejściami analogowymi czy dodatkowe wyspecjalizowane moduły sterujące urządzeń wykonawczych). Wyprowadzenia interfejsu CAN udostępnione są na złączach śrubowych sterownika. Odległość między modułami rozproszonymi a sterownikiem byłaby ograniczona przez specyfikację magistrali CAN – od 40 m przy prędkości 1 Mb/s do 10 km przy prędkości 5 kb/s [9]. Na etapie realizacji układów sterowania należy pamiętać, że cyfrowa transmisja danych jest bardziej odporna na zakłócenia i znacznie zwiększa możliwą odległość wymiany informacji między przyrządami pomiarowymi i urządzeniami wykonawczymi a sterownikiem.

2.3. Obwód drukowany i obudowa

Schematy elektroniczne oraz schemat obwodu drukowanego zostały wykonane za pomocą darmowego pakietu oprogramowania Kicad [10]. Wykorzystany pakiet posiada bogatą bazę bibliotek podzespołów elektronicznych a dodatkowo umożliwia tworzenie własnych komponentów. W oparciu o opracowany schemat elektroniczny, oprogramowanie kontroluje proces projektowania obwodu drukowanego. Na rys. 7 przedstawiono model 3D zaprojektowanego i pomyślnie zweryfikowanego obwodu drukowanego PCB przedstawianego sterownika.



Rys. 7. Wizualizacja 3D obwodu drukowanego PCB
Fig. 7. 3D visualization of printed circuit board (PCB)

Układ elektroniczny sterownika wraz z pozostałymi elementami zamknięto w obudowie, którą można zamontować na standardowej szynie montażowej DIN 35 mm. Rzeczywiste zdjęcie opisywanego uniwersalnego sterownika programowalnego znajduję się na rys. 8.

Rys. 8. Rzeczywiste zdjęcie zbudowanego sterownika programowalnego
Fig. 8. Real photo of built programmable logic controller

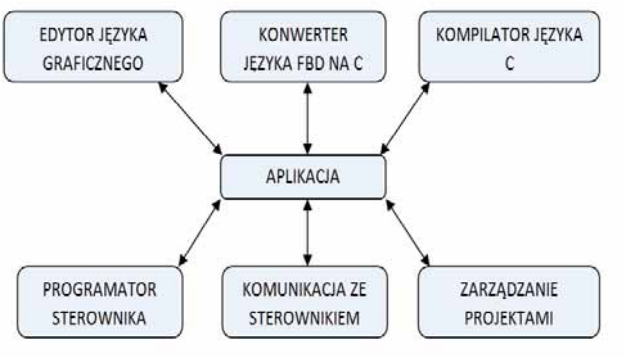
3. Dedykowane środowisko narzędziowe

3.1. Aplikacja na PC

Prosta konfiguracja, uniwersalne układy we/wy, możliwości komunikacyjne, sposób programowania oraz niezbędny czas związany z realizacją zadania budowy układu sterownia są wyznacznikami użyteczności każdego urządzenia sterującego. Aby im sprostać zaprojektowano, zaimplementowano i zweryfikowano zintegrowane środowisko programistyczne dla komputera klasy PC, które umożliwia użytkownikowi przedstawianego sterownika realizację następujących zadań:

- tworzenie oprogramowania sterownika w języku graficznym FBD,
- kompilowanie i wgrywanie oprogramowania do sterownika,
- konfigurację sterownika, dokonywanie zmian w ustawieniach systemu operacyjnego czasu rzeczywistego oraz interfejsów komunikacyjnych,
- monitorowanie wartości zmiennych oraz możliwość zmiany ich wartości (praca w trybie on-line).

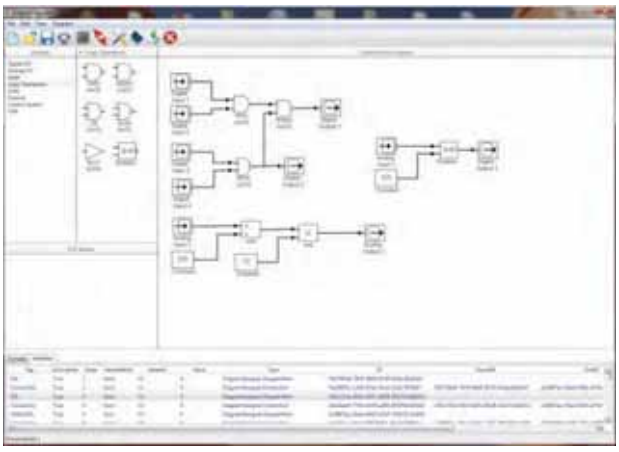
Opracowana w języku C# aplikacja bazuje na technologii WPF (ang. Windows Presentation Foundation) firmy Microsoft [11]. Jej podstawowe funkcje przedstawiono na rys. 9.



Rys. 9. Funkcje aplikacji (środowiska narzędziowego)
Fig. 9. Application (programming environment) functions

W aplikacji wykorzystano gotowy „silnik" WPF Diagram Designer [12], który umożliwia umieszczanie i łączenie ze sobą bloków funkcyjnych w przestrzeni roboczej. Niemniej jednak wymagał on licznych modyfikacji. Między innymi dodano system identyfikacji bloków i relacji między nimi w przestrzeni roboczej oraz opracowano i zaimplementowano algorytm konwertujący program użytkownika z postaci graficznej FBD na zapis w języku C. Aplikacja umożliwia wymianę informacji o zmiennych procesowych ze sterownikiem przy pomocy protokołu Modbus [13]. Z oprogramowaniem aplikacji został zintegrowany kompilator dla mikrokontrolerów ARM oraz program umożliwiający programowanie pamięci flash.

Przykładowy ekran opracowanego środowiska narzędziowego przedstawiono na rys. 10. Można na nim odnaleźć między innymi: obszar roboczy do programowania sterownika za pomocą języka FBD, bazę podstawowych bloków funkcyjnych języka FBD oraz podgląd zmiennych związanych z programem sterującym (zmienne zdefiniowane przez środowisko po kompilacji programu i przesłaniu go do sterownika).



Rys. 10. Przykładowy ekran aplikacji narzędziowej na PC
Fig. 10. The example screen of the utility application for PC

Bloki graficznego języka FBD podzielono na następujące kategorie funkcyjne:

- Digital I/O – obsługa wejść/wyjść cyfrowych,
- Analog I/O – obsługa wejść/wyjść analogowych,
- Math – zestaw funkcji matematycznych,
- Logic – zestaw bramek logicznych i operatorów relacji,

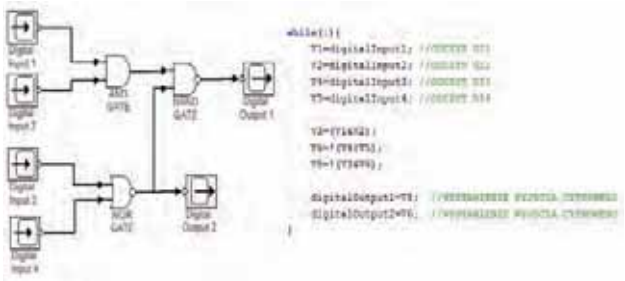
– Sources – źródła sygnałów np. stała lub generator
– Sinks – wyświetlacz, zapis do pliku,
– Control System – zestaw regulatorów, regulator dwustanowy i PID,
– CAN – bloki obsługujące sieć CAN.

Ponieważ np. bardziej złożone algorytmy efektywniej wprowadza się w formie tekstowej w kolejnej wersji oprogramowania narzędziowego planuje się udostępnienie możliwości pisania własnych skryptów za pomocą języka C (prace w toku). Ich integracja z programem napisanym w języku graficznym FBD będzie możliwa za pomocą dodatkowego bloku funkcyjnego.

3.2. Konwerter języka FBD i C oraz programator sterownika

Język graficzny FBD jest przyjazny użytkownikom (szczególnie słabo znającym języki programowania), ale kompletnie niezrozumiały mikrokontrolerom. Program opracowany w języku FBD należy przekonwertować na kod maszynowy, postać akceptowaną przez mikrokontroler. Konwersja taka przebiega w dwóch etapach.

Pierwszy etap polega na przetłumaczeniu programu z języka graficznego FBD na język C. W tym celu opracowano i zaimplementowano dedykowany algorytm, który wg określonych zasad tłumaczy kolejne bloki znajdujące się w przestrzeni roboczej wraz z relacjami między nimi na kod w języku C. Prosty przykład translacji języka graficznego FBD na język C przedstawiono na rys. 11.



Rys. 11. Przykład translacji języka graficznego FBD na język C
Fig. 11. The example of translation from FBD graphic language to C language

Wyjścia ze wszystkich bloków są przedstawione za pomocą zmiennych o określonym typie, deklaracja zmiennych jest generowana przed pętlą główną programu.

Kolejność wykonywania poszczególnych bloków programu jest związana bezpośrednio z ich położeniem w przestrzeni roboczej, w której edytuje się program za pomocą języka FBD (rys. 10). Algorytm konwertujący skanuje program stworzony w języku graficznym FBD z góry na dół i z lewej do prawej strony obwodu. Wartość żadnego z elementów obwodu nie jest wyznaczona, dopóki nie zostaną wyznaczone wartości wszystkich jego wejść. Kiedy algorytm napotka blok o niewyznaczonych wejściach, to zaczyna analizę bezpośrednio poprzedzających go bloków, których wyjścia stanowią jego wejścia. Algorytm w tej postaci w sposób poprawny uwzględnia ewentualne sprzężenia zwrotne występujące w programie (np. możliwe jest zbudowanie na dostępnych bramkach logicznych przerzutnika asynchronicznego typu RS).

Kod uzyskany w wyniku konwersji z języka graficznego FBD na język C stanowi jedynie część oprogramowania przeznaczonego dla mikrokontrolera. Na tym etapie jest on „wbudowany", dołączany do kodu systemu operacyjnego nadzorującego pracę sterownika.

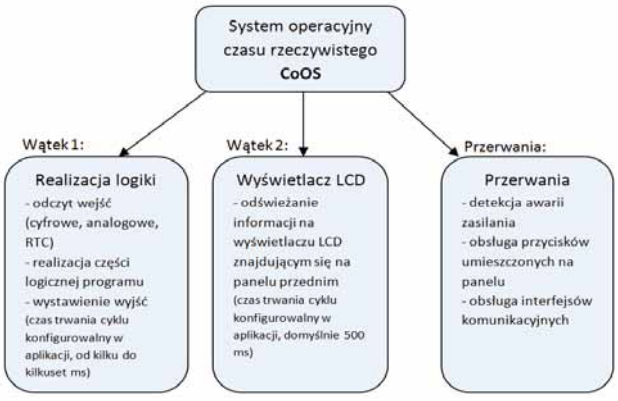
Drugi etap konwersji (program i system operacyjny) tym razem już na kod maszynowy polega na wykorzystaniu przez aplikację kompilatora dedykowanego procesorom ARM Sourcery G++ Lite [14], który kompiluje kod zapisany w języku C na kod maszynowy zapisany w formacie Intel hex. Uzyskaną postać kodu można następnie wykorzystać do zaprogramowania przedstawianego w artykule sterownika za pomocą opisywanej aplikacji narzędziowej.

3.3. System operacyjny czasu rzeczywistego dla sterownika programowalnego

Przedstawiany sterownik programowalny pracuje pod kontrolą systemu operacyjnego czasu rzeczywistego (ang. Real Time Operating System) CooCox CoOS [15]. Jest to system charakteryzujący się prostą implementacją, dobrą dokumentacją oraz biblioteką rozbudowanych funkcji zarządzających: czasem, pamięcią, wątkami oraz ich synchronizacją. Dodatkowo, co nie jest bez znaczenia, system ten jest darmowy i otwarty. Właśnie te cechy zadecydowały o jego wyborze spośród wielu dostępnych RTOS na rynku dla potrzeb budowanego sterownika.

CooCox CoOS w sterowniku programowalnym po jego inicjalizacji (konfiguracja peryferii mikrokontrolera) zarządza dwoma podstawowymi wątkami odpowiedzialnymi za realizację logiki programu sterującego oraz obsługę wbudowanego wyświetlacza LCD. Natomiast do obsługi przycisków oraz interfejsów komunikacyjnych wykorzystano mechanizmy przerwań.

Schematycznie zadania RTOS jak i organizacja pracy sterownika zostały przedstawione na rys. 12.



Rys. 12. Zadania systemu operacyjnego czasu rzeczywistego dla sterownika programowalnego
Fig. 12. Real time operating system tasks for programmable controller

CooCox CoOS jako RTOS wspiera dwa podstawowe rodzaje algorytmów przełączania pomiędzy współbieżnymi wątkami: algorytm karuzelowy (ang. *round robin*) oraz algorytm uwzględniający priorytety. O ile pierwszy algo-

rytm stosuje się typowo w przypadkach, kiedy wszystkie wątki mają ten sam priorytet, to drugi z nich stosuje się typowo, gdy występują wątki o różnych priorytetach. W algorytmie karuzelowym algorytm harmonogramowania zadań dzieli równo czas procesora i przydziela go kolejnym wątkom. Natomiast gdy w systemie występują wątki o różnym priorytecie, planista przełącza wątki w trzech następujących sytuacjach:

– wątek o wyższym priorytecie niż obecnie wykonywany przechodzi w stan gotowości,
– wykonywany wątek przechodzi w stan oczekiwania lub zatrzymania,
– wątek o tym samym priorytecie co wątek aktualnie wykonywany zgłasza stan gotowości, a czas przydzielony aktualnemu wątkowi dobiega końca.

W sterowniku wątek wykonujący część logiczną programu sterującego posiada wyższy priorytet niż wątek odpowiedzialny za obsługę wbudowanego wyświetlacza LCD.

3.4. Komunikacja ze sterownikiem – praca w trybie on-line

Monitorowanie wartości zmiennych oraz możliwość zmiany ich wartości w trakcie pracy sterownika (tryb on-line) przekładają się na większą efektywność programowania sterownika (np. szybsze odnajdywanie błędów w programie sterującym). Dodatkowo gdy funkcjonalność tą uzyska się dzięki popularnemu w aplikacjach przemysłowych protokołowi komunikacyjnemu (np. Modbus RTU), to otwierają się dodatkowe możliwości związane ze stosowaniem dodatkowego dowolnego oprogramowania obsługującego ten protokół, w tym oprogramowania typu SCADA, które może współpracować ze sterownikiem.

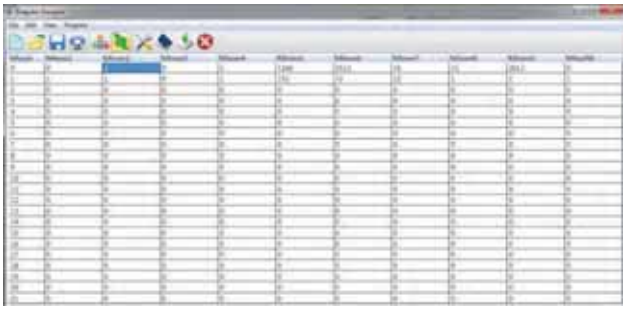
Wspomniany wyżej protokół Modbus RTU został zaimplementowany zarówno w aplikacji narzędziowej (instalowanej na komputerze PC) jak i w ramach systemu operacyjnego RTOS kontrolującego sterownik programowalny. Po stronie aplikacji narzędziowej wykorzystano bibliotekę NModbus [16], która zawiera funkcjonalny pakiet klas pozwalający w prosty sposób wykorzystać protokół RTU do „czytania" i „zapisywania" odpowiednich danych. Przykładową, prostą funkcję służącą do odczytu rejestrów przedstawiono w postaci kodu w C# na rys. 13.

```
public ushort[] ModbusRead(string portName, byte slaveID, ushort adres, ushort noReg) {
    ushort[] result = { 0 };
    SerialPort port = new SerialPort();
    IModbusSerialMaster master = ModbusSerialMaster.CreateRtu(port);
    adres--;
    port.PortName = portName;
    port.BaudRate = BaudRate;
    port.DataBits = DataBits;
    port.StopBits = stopB;
    port.Parity = pari;
    if (!port.IsOpen)
    {
        port.Open();
        result = master.ReadHoldingRegisters(slaveID, adres,noReg);
        port.Close();
   }
    return result;
}
```

Rys. 13. Przykładowy kod w C# wykorzystania biblioteki NModbus
Fig. 13. Example of C# code with use of NMbus library

Jak można zauważyć (rys. 13), odczyt danych przy wykorzystaniu biblioteki NModbus sprowadza się do stworzenia obiektu klasy ModbusSerialMaster, powiązania go ze skonfigurowanym portem szeregowym i wywołaniu funkcji ReadHoldingRegisters, która jako parametry przyjmuje numer identyfikacyjny urządzenia slave oraz adres i ilość zmiennych do odczytu. W analogiczny sposób realizowana jest operacja zapisu do rejestrów oraz innych operacji charakterystycznych dla protokołu Modbus RTU.

Natomiast na rys. 14 przedstawiono przykładowy ekran aplikacji narzędziowej pracującej w trybie on-line, trybie monitorowania wartości zmiennych, które zostały zdefiniowane w przykładowym prostym programie sterującym użytkownika. Przy czym poszczególne zmienne identyfikowane są po adresach, które są definiowane na etapie konwersji jeżyka FBD na C i kompilacji programu użytkownika.



Rys. 14. Monitoring zmiennych procesowych trybie on-line (aplikacja narzędziowa na PC)
Fig. 14. Variables monitoring in on-line mode (utility application for PC)

Odpowiednio po stronie sterownika programowalnego wykorzystano odpowiednio zmodyfikowany i przystosowany, pod względem zastosowanego w sterowniku mikrokontrolera, pakiet FreeModbus [17].

Przeprowadzone testy potwierdzają poprawność implementacji protokołu Modbus RTU zarówno po stronie aplikacji narzędziowej jak i po stronie sterownika programowalnego.

4. Podsumowanie

W artykule przedstawiono pozytywnie zweryfikowaną prototypową konstrukcję uniwersalnego sterownika programowalnego, który został zbudowany w oparciu o łatwo dostępne podzespoły elektroniczne, a koszt jego budowy w porównaniu do cen sterowników programowalnych o podobnych możliwościach jest bardzo atrakcyjny (budowa jednego sterownika od podstaw na dzień opracowania artykułu zamyka się w kwocie 600 zł). Część sprzętowa sterownika została zaprojektowania w taki sposób, aby był on: jak najbardziej odporny na błędy użytkownika; prosty w ewentualnych naprawach; oraz mógł współpracować z szeroką gamą urządzeń wykorzystywanych w typowych instalacjach przemysłowych.

W artykule przedstawiono również funkcjonalność opracowanego równolegle ze sterownikiem, zintegrowanego środowiska narzędziowego do jego obsługi, które ma decydujący wpływ na możliwości i efektywność wykorzystania opracowanego sterownika programowalnego w realnych układach sterowaniach (poczynając od najprostszych, jak np. sterowniki bram, przez bardziej złożone, jak np. sterowniki kolektorów słonecznych, czy np. sterowniki pieców

c.o., wentylacji i klimatyzacji lub kotłowni przemysłowych). Opracowane środowisko narzędziowe zostało również pozytywnie zweryfikowane w wyniku szeregu testów użytkowych.

Dalsze prace nad zaprezentowanym uniwersalnym sterownikiem programowalnym oraz jego oprogramowaniem narzędziowym planuje się skierować głównie w stronę rozwoju oprogramowania narzędziowego oraz konstrukcji dodatkowych rozproszonych układów we/wy.

Bibliografia

1. [www.st.com/internet/mcu/product/247492.jsp] – Opis oraz dokumentacja techniczna mikrokontrolera STM32F103VG (18 listopada 2012).
2. Pławsiuk K., *Zabezpieczenie przed odwrotną polaryzacją*, „Elektronika Praktyczna", 4/2003, 76.
3. [www.fairchildsemi.com/ds/1N/1N4001.pdf] – Nota katalogowa diody 1N4007 (18 listopada 2012).
4. Legierski T., Kasprzyk J., Hajda J., Wyrwał J., *Programowanie sterowników PLC*, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 1998.
5. Hadam P., *Projektowanie systemów mikroprocesorowych*, Wydawnictwo BTC, Warszawa 2004.
6. [www.irf.com/product-info/datasheets/data/irfz44n.pdf] – Nota katalogowa tranzystora IRFZ44N (18 listopada 2012).
7. [www.denizyildirim.org/mylibrary/data/tc427.pdf] – Nota katalogowa sterownika tranzystorów MOSFET (18 listopada 2012).
8. [www.datasheetcatalog.org/datasheet/stmicroelectronics/2163.pdf] - Nota katalogowa wzmacniacza operacyjnego LM358D (18 listopada 2012).
9. *Urządzenia diagnostyczne do sieci CAN*, „Elektronika Praktyczna" 3/2011, 22-28.
10. [www.kicad-pcb.org] – Oficjalna strona oprogramowania KiCad (18 listopada 2012).
11. [http://msdn.microsoft.com/en-us/library/ms742119.aspx] – Windows Presentation Foundation – Getting Started (18 listopada 2012).
12. [www.codeproject.com/Articles/22952/WPF-Diagram-Designer-Part-1] - Silnik wyświetlania bloków WPF Diagram Designer (18 listopada 2012).
13. [http://modbus.org] – Oficjalna strona organizacji zajmującej się protokołem Modbus (18 listopada 2012).
14. [www.mentor.com/embedded-software/codesourcery] - Strona projektu Codesourcery (18 listopada 2012).
15. [www.coocox.org/CoOS.htm] – CooCox CoOS (18 listopada 2012).
16. [http://code.google.com/p/nmodbus/] – A C# implementation of the Modbus protocol (18 listopada 2012).
17. [http://freemodbus.berlios.de/] – A Modbus ASCII/RTU and TCP implementation for embedded systems (18 listopada 2012).

Project and implementation of a universal programmable controller

Abstract: The paper describes project and implementation of a low-cost but functional and universal programmable logic controller PLC in compact case and dedicated utility application for that PLC, which allow user to its effective programming. The main central processing unit of presented controller is based on 32 bit ARM STMicroelectronics microcontroller. Presented PLC controller is equipped with standard peripherals, which are used in industrial technological installations: digital I/O, analog I/O, encoder interface. The controller may communicate with PC computers or other devices via USB and RS-232 interfaces and CAN bus. Presented in article utility application is developed in C# language for Windows platform. Utility application allow user to: develop control programs with graphical programming language FBD, controller programming, simple diagnosis and variables monitoring during running controller (on-line mode). In order to track the values of selected variables the built-in LCD display may be used by user. Additionally, using the buttons available on the controller case the user, for example, may perform "tuning" of the selected parameters of running control program.

Keywords: PLC, microcontroller, industrial control systems

inż. Sebastian Wójcicki

W 2012 r. uzyskał tytuł inżyniera na Wydziale Elektrotechniki i Automatyki Politechniki Gdańskiej. Jego zainteresowania naukowe obejmują projektowanie i zastosowanie układów mikroprocesorowych w systemach automatyki.

e-mail: WojcickiS@gmail.com

dr inż. Tomasz Rutkowski

W 2004 r. uzyskał stopień doktora nauk technicznych w dziedzinie automatyka i robotyka nadany przez Wydział Elektrotechniki i Automatyki Politechniki Gdańskiej. Jego obecne zainteresowania naukowe obejmują zaawansowane algorytmy sterowania, algorytmy estymacji, techniki inteligencji obliczeniowej oraz przemysłowe systemy sterowania.

e-mail: t.rutkowski@eia.pg.gda.pl