Przyspieszenie wymiany danych w protokole Modbus między PLC a HMI wykorzystującymi pakiet inżynierski CPDev

Dariusz Rzońca
Politechnika Rzeszowska, Katedra Informatyki i Automatyki, ul. W. Pola 2, 35-959 Rzeszów

Streszczenie: W artykule przedstawiono możliwości poprawy parametrów czasowych komunikacji między sterownikiem przemysłowym PLC a panelem operatorskim HMI. Jak pokazano, niekiedy odpowiednia konfiguracja zadań komunikacyjnych, zmniejszająca liczbę poleceń przesyłanych w protokole Modbus kosztem konieczności transmisji dodatkowych danych, może prowadzić do minimalizacji łącznego czasu cyklu komunikacyjnego. Przedstawione rozwiązanie zostało zaimplementowane w pakiecie inżynierskim CPDev.

Słowa kluczowe: sterownik przemysłowy, PLC, panel operatorski, HMI, komunikacja, Modbus, CPDev

1. Wprowadzenie

W rozproszonych systemach automatyki często zachodzi konieczność wymiany danych między sterownikiem przemysłowym PLC (ang. *Programmable Logic Controller*) lub PAC (ang. *Programmable Automation Controller*) a panelem operatorskim HMI (ang. *Human-Machine Interface*). Artykuł dotyczy wybranych aspektów takiej komunikacji, zwłaszcza występujących w pakiecie inżynierskim CPDev podczas komunikacji PLC z HMI za pomocą protokołu Modbus. Można go zasadniczo traktować jako kontynuację pracy [1], gdzie skupiono się na analizie sytuacji, gdy panel HMI jest urządzeniem nadrzędnym cyklicznie odpytującym sterownik PLC. Tu przedstawiono odmienne podejście, równie często występujące w praktyce inżynierskiej, gdy to HMI jest urządzeniem podrzędnym, do którego PLC cyklicznie zapisuje dane.

W kolejnym rozdziale zamieszczono krótki przegląd literatury związanej z problematyką poruszaną w artykule. Szczególną uwagę poświecono zagadnieniom związanym ze środowiskiem CPDev, normą IEC 61131-3 i komunikacją z panelami HMI. Trzeci rozdział przedstawia proponowane podejście, a także prostą analizę pokazującą, kiedy grupowanie zmiennych prowadzi do skrócenia cyklu komunikacyjnego. Ostatni rozdział zawiera zwięzłe podsumowanie.

Autor korespondujący:
Dariusz Rzońca, drzonca@kia.prz.edu.pl

Artykuł recenzowany
nadesłany 08.10.2022 r., przyjęty do druku 13.11.2022 r.

2. Przegląd literatury

Pakiet inżynierski CPDev (ang. *Control Program Developer*) jest opracowanym w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej środowiskiem pozwalającym na programowanie sterowników przemysłowych w językach normy IEC 61131-3 [17], przyjętej w Polsce jako PN-EN 61131-3. Prace nad pakietem CPDev rozpoczęły się kilkanaście lat temu [2], środowisko jest ciągle rozwijane i rozbudowywane. Przebieg prac nad jego implementacją, jak też aktualny opis środowiska CPDev można znaleźć m.in. w [3, 4]. Obecnie CPDev jest wdrożony przez kilku producentów systemów automatyki, czterech polskich i dwóch zagranicznych (Hiszpania, Holandia).

Norma IEC 61131-3 [17] definiuje pięć specjalistycznych, dziedzinowych języków programowania, przeznaczonych dla sterowników PLC i PAC. Są to języki:
- ST – *Structured Text*,
- IL – *Instruction List*,
- FBD – *Function Block Diagram*,
- LD – *Ladder Diagram*,
- SFC – *Sequential Function Chart*.

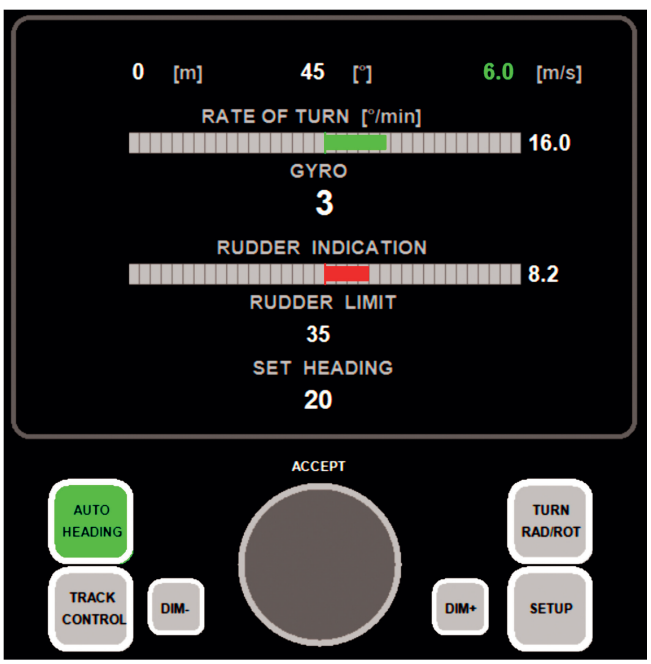
Dobry opis języków normy można znaleźć w książce [5]. Wszystkie z wymienionych języków wspierane są przez środowisko CPDev.

Pakiet CPDev został zaprojektowany w taki sposób, by ułatwić jego implementację przez różnych producentów PLC, niezależnie od zastosowanej platformy sprzętowej. Jest to osiągane dzięki odseparowaniu środowiska graficznego od oprogramowania sterownika. Programy sterowania w językach normy są kompilowane do opracowanego na potrzeby CPDev kodu pośredniego VMASM (ang. *Virtual Machine Assembler*), a następnie wykonywane przez specjalizowany interpreter wchodzący w skład oprogramowania wbudowanego (ang. firmware) PLC, zwany maszyną wirtualną [6]. Kod źródłowy maszyny wirtualnej został przygotowany w języku C, dzięki czemu może być łatwo zaimplementowany na praktycznie dowolnym mikroprocesorze.

Istniejące wdrożenia obejmują zarówno sterowniki PLC bazujące na niewielkich ośmiobitowych mikrokontrolerach (np. AVR), jak też większe (ARM), a także rozwiązania soft-PLC pracujące na komputerach z architekturą x86.

Komunikacja w rozproszonych systemach automatyki zazwyczaj wykorzystuje dedykowane sieci przemysłowe [7]. Są to sieci polowe [8], a w nowszych rozwiązaniach coraz częściej Ethernet [9]. W przypadku sieci polowych stosowane są rozwiązania opisane w normie IEC 61158 [18]. Można także dostrzec tendencję do coraz szerszego wykorzystywania w komunikacji przemysłowej rozwiązań heterogenicznych, integrujących różne rodzaje sieci [10], zwłaszcza w dużych systemach. Niewielkie rozproszone systemy automatyki często jednak nadal bazują na szeregowej magistrali komunikacyjnej (np. RS-485) z protokołem Profibus lub Modbus. Zwłaszcza otwarty protokół Modbus jest bardzo popularny wśród producentów niewielkich aparatowych urządzeń automatyki.

Panele operatorskie HMI są często stosowane m.in. do wizualizacji danych procesowych, ustawiania parametrów [11, 12]. Środowisko inżynierskie CPDev również pozwala na projektowanie prostych ekranów graficznych dla panelu HMI za pomocą narzędzia CPVis [13] wchodzącego w skład CPDev. Jego funkcjonalność w tym zakresie opisywano także w [14]. Przykładowy ekran procesowy HMI, przygotowany za pomocą CPVis dla autopilota okrętowego [15], pokazano na rysunku 1.



Rys. 1. Ekran HMI autopilota okrętowego opracowany w CPVis
Fig. 1. HMI screen of ship autopilot developed in CPVis

W module CPVis zastosowano podobną koncepcję z interpreterem przetwarzającym kod pośredni, tym razem dotyczącym grafiki. CPVis składa się z graficznego edytora uruchamianego na PC i środowiska wykonawczego (interpretera) uruchamianego na docelowym panelu HMI. Ekrany procesowe są przygotowywane w edytorze CPVis z elementów bibliotecznych, prostych (jak np. linia, prostokąt) oraz złożonych (np. wykres). Poszczególne parametry tych elementów mogą być stałe lub uzależnione od wartości wizualizowanych zmiennych, np. wysokość prostokąta może odzwierciedlać poziom cieczy w zbiorniku, jego kolor temperaturę itp. Interpreter CPVis na panelu HMI cyklicznie generuje bieżący obraz na podstawie aktualnych wartości wizualizowanych parametrów. Proces ten szczegółowo omówiono w [16].

3. Komunikacja PLC z HMI w CPDev

Sterownik PLC wymienia dane z panelem HMI za pomocą magistrali komunikacyjnej, zazwyczaj RS-485, a w nowszych rozwiązaniach Ethernet. W przypadku sieci polowych często stosowane są protokoły takie, jak np. Profibus czy Modbus. Na potrzeby dalszych rozważań przyjmijmy komunikację za pomocą bardzo popularnego protokołu Modbus RTU. Jest to protokół oparty na paradygmacie *m*aster-slave, co oznacza, że jedno z urządzeń jest urządzeniem nadrzędnym (ang. master) inicjującym transmisję, a pozostałe są urządzeniami podrzędnymi (ang. slave) odpowiadającymi na polecenia. Warto zauważyć, że w komunikacji paneli HMI ze sterownikiem PLC stosuje się zarówno takie rozwiązania, gdzie HMI jest urządzeniem nadrzędnym cyklicznie odpytującym PLC o wartości wizualizowanych zmiennych, jak też takie, gdzie PLC pełni rolę *mastera* przesyłając do HMI aktualne wartości zmiennych po każdym cyklu obliczeń. Pierwszy z przypadków został przeanalizowany w [1], tu zajmiemy się tylko drugim podejściem.

Protokół Modbus określa szereg funkcji służących do wymiany danych. Najpopularniejsze z nich to:
- FC1 – odczyt zmiennych binarnych,
- FC2 – odczyt wejść binarnych,
- FC3 – odczyt rejestrów,
- FC4 – odczyt rejestrów wejściowych,
- FC5 – zapis pojedynczej zmiennej binarnej,
- FC6 – zapis pojedynczego rejestru,
- FC15 – zapis wielu zmiennych binarnych,
- FC16 – zapis wielu rejestrów.

Zakładając, że sterownik PLC jako urządzenie nadrzędne zapisuje do panelu HMI wielobitowe zmienne, wykorzystywane będą głownie polecenia FC6 i FC16. Funkcja FC6 pozwala na zapis pojedynczego szesnastobitowego rejestru, natomiast polecenie FC16 zapisuje wiele następujących po sobie rejestrów. Ich ramki przedstawiono odpowiednio na rysunkach 2 i 3.

Adres *slave*	Kod funkcji	Adres rejestru	Wartość	CRC
1B	1B (0×06)	2B	2B	2B

Rys. 2. Ramka polecenia FC6 Modbus RTU
Fig. 2. Frame of Modbus RTU FC6 function

Odpowiedź na funkcję FC6 przesyłana przez urządzenie podrzędne, zakładając poprawne wykonanie zapisu, jest kopią przesłanego polecenia. Ramka odpowiedzi na funkcję FC16 zawiera liczbę zapisanych rejestrów, przedstawiono ją na rysunku 4.

Można zauważyć, że długość ramki odpowiedzi w obu przypadkach (FC6 i FC16) jest identyczna i wynosi 8 bajtów. Rozmiar ramki polecenia FC6 również wynosi 8 bajtów, natomiast długość ramki funkcji FC16 jest zmienna i zależy od liczby zapisywanych rejestrów. Oczywiste jest, że w przypadku, gdy chcemy zapisać tylko jeden rejestr, to użycie dedykowanego polecenia FC6 pozwoli na przesłanie krótszego komunikatu, a w onsekwencji na skrócenie czasu transmisji w porównaniu do bardziej uniwersalnego, ale dłuższego FC16. Z drugiej strony, jeżeli zachodzi konieczność zapisu dwóch lub większej liczby kolejnych rejestrów, to szybsze będzie użycie jednego wywołania FC16 niż kilku FC6, zapisując rejestry pojedynczo. Może się jednak zdarzyć, że rejestry, które chcemy zapisać w jednej transakcji komunikacyjnej nie następują bezpośrednio po sobie. Oczywiście najwygodniej byłoby, aby wszystkim zmiennym wizualizowanym na HMI przydzielić kolejne adresy, np. za pomocą modyfikatora AT w języku ST. Niestety nie zawsze jest to możliwe, gdyż te same zmienne mogą być stosowane na kilku ekranach procesowych. Mogą być także elementami większych zmiennych złożonych (tablic,

Adres *slave*	Kod funkcji	Adres początkowy	Liczba rejestrów	Liczba bajtów	Dane	CRC
1B	1B (0×10)	2B	2B	1B	n * 2B	2B

Rys. 3. Ramka polecenia FC16 Modbus RTU
Fig. 3. Frame of Modbus RTU FC16 function

Adres *slave*	Kod funkcji	Adres początkowy	Liczba rejestrów	CRC
1B	1B (0×10)	2B	2B	2B

Rys. 4. Ramka odpowiedzi FC16 Modbus RTU
Fig. 4. Frame of reply to Modbus RTU FC16 function

struktur), co narzuca ich odpowiednie rozmieszczenie w pamięci sterownika. W naturalny sposób pojawia się pytanie, czy w pewnych wypadkach nie byłoby korzystne przesłanie także dodatkowych zmiennych, poza wymaganymi, tylko po to, by użyć pojedynczego polecenia FC16 zamiast kilku FC6.

Rozważmy sytuację, gdy potrzebujemy przesłać dwie szesnastobitowe zmienne (dwa rejestry) rozdzielone w pamięci sterownika przez k innych rejestrów, przeanalizujmy dwa sposoby by to osiągnąć.

Przyjmijmy następujące oznaczenia:
- t_b – czas transmisji jednego znaku (zależny od ustalonej prędkości łącza komunikacyjnego),
- t_m – czas potrzebny na przygotowanie polecenia przez urządzenie nadrzędne (*master*),
- t_s – czas potrzebny na przygotowanie odpowiedzi przez urządzenie podrzędne (*slave*),
- t_{cycle_i} – łączny czas cyklu komunikacyjnego w i-tym przypadku.

Pierwszy sposób wymaga użycia dwóch osobnych poleceń FC6 aby zapisać rejestry pojedynczo. Po przygotowaniu pierwszego polecenia przez urządzenie nadrzędne (co trwa t_m) rozpoczyna się transmisja ramki. Ramka FC6 liczy 8 bajtów, a więc czas potrzebny na jej przesłanie to $8 * t_b$. Protokół Modbus RTU definiuje detekcję końca ramki jako wykrycie ciszy na magistrali trwającą $3{,}5 * t_b$, dopiero więc po tym czasie urządzenie podrzędne zacznie przetwarzać polecenie i przygotowywać odpowiedź. Po czasie t_s rozpocznie się transmisja odpowiedzi trwająca $8 * t_b$, a następnie cisza przez $3{,}5 * t_b$. Po zakończeniu transmisji pierwszego rejestru w podobny sposób zostanie przesłany kolejny, finalnie otrzymujemy więc:

$$t_{cycle_1} = 2\left(t_m + 8t_b + 3{,}5t_b + t_s + 8t_b + 3{,}5t_b\right) = 46t_b + 2t_m + 2t_s$$

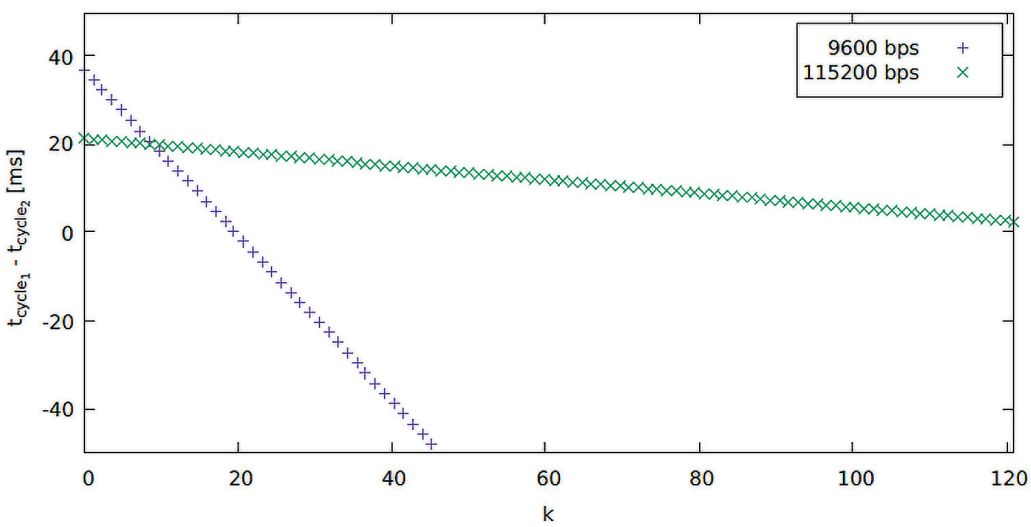
Drugi sposób zakłada transmisję za pomocą jednego polecenia FC16 dwóch pożądanych rejestrów i k nadmiarowych rejestrów, leżących między nimi. Oznacza to, że ramka polecenia FC16 będzie mieć rozmiar $13 + 2k$ bajtów. Oczywiście jest to możliwe dla $k \le 121$ z uwagi na ograniczenie długości całej ramki Modbus do 255 bajtów. Finalnie uzyskujemy:

$$t_{cycle_2} = t_m + (13 + 2k)t_b + 3{,}5t_b + t_s + 8t_b + 3{,}5t_b = (28 + 2k)t_b + t_m + t_s$$

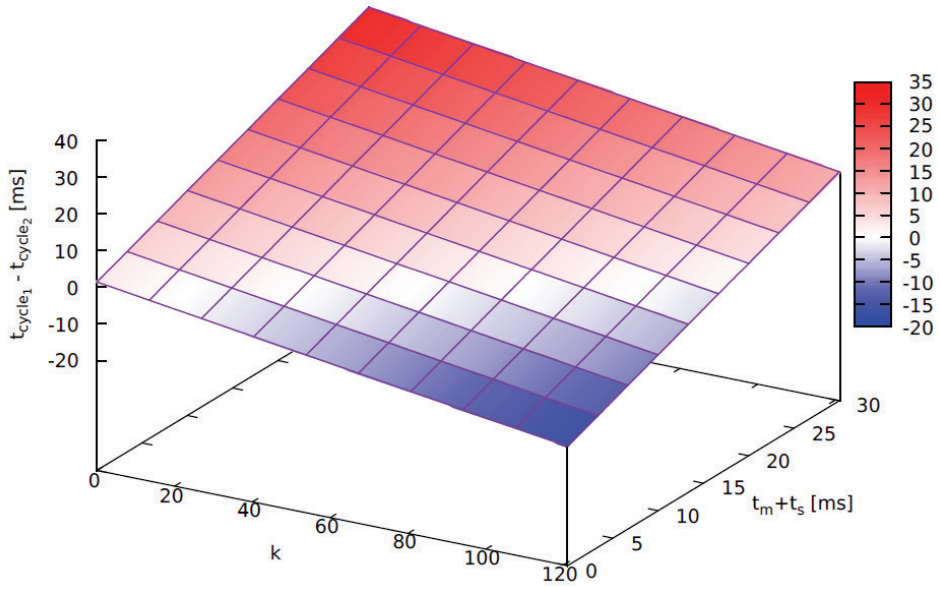
Obliczmy różnicę $t_{cycle_1} - t_{cycle_2}$. Jeżeli jest ona dodatnia, to znaczy, że korzystniejszy jest przypadek drugi, jeżeli ujemna – pierwszy.

$$t_{cycle_1} - t_{cycle_2} = (18 - 2k)t_b + t_m + t_s$$

Łatwo zauważyć, że dla $k \le 9$ optymalne będzie wykorzystanie pojedynczego polecenia FC16, niezależnie od wartości czasów t_b, t_m, $t_s > 0$. Osiągane parametry czasowe są lepsze, mimo konieczności przesłania nadmiarowych danych (dodatkowych rejestrów). Dla $k \ge 10$ wybór optymalnego sposobu komunikacji zależy od konkretnych wartości t_b, t_m i t_s. Kontynuujmy analizę dla przykładowych wartości tych parametrów. Czas transmisji jednego znaku t_b zależy od ustalonej prędkości łącza komunikacyjnego. Przykładowo dla transmisji z prędkością 9600 bps w formacie 8N1 (osiem bitów danych, brak kontroli parzystości, jeden bit stopu) t_b wynosi 0,9375 ms, a dla transmisji z prędkością 115 200 bps w formacie 8N1 $t_b = 78{,}125$ µs. Można zauważyć, że dla wyższych prędkości transmisji dodatkowy narzut czasowy potrzebny na przesłanie nadmiarowych rejestrów będzie stosunkowo niewielki. Czasy t_m, t_s są wielokrotnie wyższe, rzędu kilku lub kilkunastu milisekund. Przyjmijmy przykładowo $t_m = t_s = 10$ ms. Obliczoną rożnicę $t_{cycle_1} - t_{cycle_2}$ dla dwóch prędkości transmisji pokazano na wykresie (Rys. 5).



Rys. 5. Różnica $t_{cycle_1} - t_{cycle_2}$ dla dwóch prędkości transmisji
Fig. 5. Difference $t_{cycle_1} - t_{cycle_2}$ for two baudrates



Rys. 6. Różnica $t_{cycle_1} - t_{cycle_2}$ dla prędkości 115 200 bps w zależności od $t_m + t_s$ i k
Fig. 6. Difference $t_{cycle_1} - t_{cycle_2}$ for 115 200 bps baudrate, depending on $t_m + t_s$ and k

Jak widać, dla przyjętych t_m i t_s przy prędkości 115 200 bps nawet dla bardzo dużych wartości k, możliwych do obsłużenia pojedynczą ramką Modbus, korzystniejsza będzie transmisja zgrupowanych zmiennych jednym poleceniem FC16, wraz z dodatkowymi rejestrami. Jeśli używana jest prędkość transmisji 9600 bps to już przy $k = 20$ lepiej będzie transmitować rejestry pojedynczo, osobnymi poleceniami FC6.

Zmiana t_m i t_s również będzie wpływać na wybór korzystniejszego sposobu transmisji, odpowiednio przesuwając próg k, dla którego grupowanie rejestrów jest opłacalne. Można potraktować sumę $t_m + t_s$ jako jedną zmienną, k jako drugą i rozważać różnicę $t_{cycle_1} - t_{cycle_2}$ jako funkcję tych dwóch zmiennych przy ustalonej prędkości łącza. Odpowiedni wykres dla prędkości 115 200 bps przedstawiono na rysunku 6. Odcieniami czerwonego oznaczono zakres, w którym lepsze jest użycie funkcji FC16, a niebieskiego – FC6, na biało wynikowe $t_{cycle_1} - t_{cycle_2}$ w pobliżu zera.

4. Podsumowanie

Zapis danych ze sterownika PLC do panelu operatorskiego HMI niejednokrotnie można zrealizować na kilka sposobów, odpowiednio grupując zmienne i wykorzystując różne polecenia Modbus. W artykule przeanalizowano przypadek, gdy sterownik PLC jest urządzeniem nadrzędnym, a zapis może przebiegać z użyciem funkcji Modbus FC6 lub FC16. Jak pokazano, niekiedy korzystne może być przesyłanie dodatkowych rejestrów, poza wymaganymi. Mimo transmisji nadmiarowych danych, zmniejszenie liczby komunikatów przesyłanych w ramach jednego cyklu komunikacyjnego, może prowadzić do poprawy osiąganych parametrów czasowych.

Podziękowania

Projekt finansowany w ramach programu Ministra Edukacji i Nauki pod nazwą „Regionalna Inicjatywa Doskonałości" w latach 2019–2023, nr projektu 027/RID/2018/19, kwota finansowania 11 999 900 zł.

Bibliografia

1. Rzońca D., *Poprawa wydajności komunikacji sterownika przemysłowego z panelem operator skim HMI w środowisku inżynierskim CPDev*, „Pomiary Automatyka Robotyka", Vol. 24, No. 1, 2020, 35–40, DOI: 10.14313/PAR_235/35.
2. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Mini-DCS system programming in IEC 61131-3 structured text*, "Journal of Automation, Mobile Robotics and Intelligent Systems", Vol. 2, No. 3, 2008, 48–54.
3. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Implementacja* środowiska inżynierskiego na przykładzie pakietu *CPDev*, „Pomiary Automatyka Robotyka", Vol. 24, No. 1, 2020, 21–28, DOI: 10.14313/PAR_235/21.
4. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Developing a Multiplatform Control Environment*, "Journal of Automation, Mobile Robotics and Intelligent Systems", Vol. 13, 2019, 73–84, DOI: 10.14313/JAMRIS/4-2019/40.
5. Kasprzyk J., *Programowanie sterowników przemysłowych*. Wydawnictwa Naukowo-Techniczne, 2006.
6. Trybus B., *Development and Implementation of IEC 61131-3 Virtual Machine*, "Theoretical and Applied Informatics", Vol. 23, No. 1, 2011, 21–35.
7. Silva M., Pereira F., Soares F., Leão C.P., Machado J., Carvalho V., *An Overview of Industrial Communication Networks*, [In:] *New Trends in Mechanism and Machine Science* (Flores P., Viadero F., eds.), (Cham), 933–940, Springer International Publishing, 2015, DOI: 10.1007/978-3-319-09411-3_97.
8. Thomesse J., *Fieldbus Technology in Industrial Automation*, "Proceedings of the IEEE", Vol. 93, No. 6, 2005, 1073–1101, DOI: 10.1109/JPROC.2005.849724.
9. Gaj P., Jasperneite J., Felser M., *Computer Communication Within Industrial Distributed Environment – a Survey*, "IEEE Transactions on Industrial Informatics", Vol. 9, No. 1, 2013, 182–189, DOI: 10.1109/TII.2012.2209668.

10. Scanzio S., Wisniewski L., Gaj P., *Heterogeneous and dependable networks in industry – A survey*, "Computers in Industry", Vol. 125, 2021, DOI: 10.1016/j.compind.2020.103388.
11. Fiset J.-Y., *Human-Machine Interface Design for Process Control Applications*. Instrumentation, Systems and Automation Society, 2009.
12. Zhang P., *Human–machine interfaces*, [In:] *Advanced Industrial Control Technology* (Zhang P., ed.), 2010, 527–555, Oxford: William Andrew Publishing.
13. Jamro M., Trybus B., *Configurable Operator Interface for CPDev Environment*, "Pomiary Automatyka Robotyka", Vol. 17, No. 2, 2013, 426–431.
14. Jamro M. Trybus B., *IEC 61131-3 programmable human machine interfaces for control devices*, [In:] 6th International Conference on Human System Interactions (HSI), 2013, 48–55, DOI: 10.1109/HSI.2013.6577801.
15. Rzońca D., Sadolewski J., Stec A., Świder Z., Trybus B., Trybus L., *Ship Autopilot Software – A Case Study*, [In:] *Advanced, Contemporary Control* (Bartoszewicz A., Kabziński J., Kacprzyk J., eds.), (Cham), 2020, 1499–1506, Springer International Publishing, DOI: 10.1007/978-3-030-50936-1_124.
16. Rzońca D., Stec A., Trybus B., *Control Program Development in CPDev Using SFC Language, HMI and Runtime Environment*, [In:] *Automation 2018* (Szewczyk R., Zieliński C., Kaliczyńska M., eds.), (Cham), 2018, 223–232, Springer International Publishing, DOI: 10.1007/978-3-319-77179-3_21.

Inne źródła

17. IEC, "IEC 61131-3 – Programmable controllers – Part 3: Programming languages", 2003.
18. IEC, "IEC 61158 – Industrial Communication Networks – Fieldbus Specifications", 2007.

Acceleration of Modbus Data Exchange between PLC and HMI Using the CPDev Engineering Environment

Abstract: The paper presents the possibilities of improving the time parameters of communication between the industrial PLC and the HMI operator panel. As shown, sometimes the appropriate configuration of communication tasks, reducing the number of commands sent in the Modbus protocol at the expense of the necessity to transmit additional data, may lead to the minimization of the total communication cycle time. The presented solution has been implemented in the CPDev engineering environment.

Keywords: industrial controller, PLC, operator panel, HMI, communication, Modbus, CPDev

dr inż. Dariusz Rzońca
drzonca@kia.prz.edu.pl
ORCID: 0000-0001-5724-0978

Licencjat matematyki (Uniwersytet Rzeszowski 2002), magister inżynier informatyki (Politechnika Rzeszowska 2004), doktor nauk technicznych w dyscyplinie informatyka, specjalność przemysłowe systemy informatyki (Politechnika Śląska 2012). Od 2004 roku asystent, a od 2013 adiunkt w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego zainteresowania naukowe koncentrują się na kolorowanych sieciach Petriego oraz zagadnieniach związanych z komunikacją w systemach automatyki.