STUDIA INFORMATICA 2008
Volume 29 Number 4B (81)

Dariusz RZOŃCA, Bartosz TRYBUS
Rzeszow University of Technology, Department of Computer and Control Engineering

APPLICATION OF COLOURED PETRI NET FOR DESIGN OF SMC CONTROLLER COMMUNICATION SUBSYSTEM [1]

Summary. Timed CPN model of SMC controller communication subsystem is described. SMC controller operates as central unit in a small distributed control-and-measurement system from LUMEL Zielona Góra. SMC is programmed in ST language available in CPDev design environment. Implementation of the communication subsystem involves communication tasks created during the design stage and executed by SMC taking into account priorities and timeouts.

Keywords: CPN, CPDev, SMC

ZASTOSOWANIE KOLOROWANYCH SIECI PETRIEGO DO PROJEKTOWANIA PODSYSTEMU KOMUNIKACYJNEGO STEROWNIKA SMC

Streszczenie. Opisano model podsystemu komunikacyjnego sterownika SMC, opracowany w czasowych kolorowanych sieciach Petriego. SMC, produkowany przez LUMEL Zielona Góra, przeznaczony jest do działania jako jednostka centralna w niewielkich, rozproszonych systemach kontrolno-pomiarowych. SMC programowany jest w języku ST w środowisku projektowym CPDev. Implementacja podsystemu komunikacyjnego wykorzystuje zadania komunikacyjne tworzone w fazie projektowania i obsługiwane przez SMC cyklicznie, w zależności od priorytetów.

Słowa kluczowe: CPN, CPDev, SMC

[1] Research supported by MNiSzW grant no. R02 058 03

1. Introduction

Domestic control-and-measurement industry manufactures transmitters, actuators, drives, PID and PLC controllers, recorders, etc. Connected into distributed systems, they are used for automation of small and medium scale processes. SMC controller (Fig. 1a) is a control-and-measurement device introduced recently by LUMEL Zielona Góra. It is equipped with 8-bit AVR ATmega 128 microcontroller [1]. SMC has two RS-485 and RS-232/485 serial ports and USB interface, but does not have external inputs and outputs of its own. Therefore it is used as a central module in distributed mini-systems for control and monitoring of small technological processes. Fig. 1b shows example realization of a mini-DCS system with SMC controller shown in Automaticon 2008.

a) b)

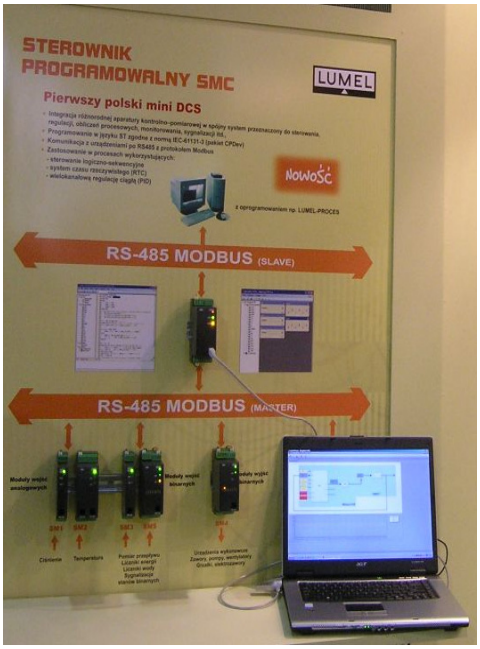

Fig. 1. a) SMC programmable controller from LUMEL Zielona Góra; b) Test set-up of mini-DCS system with SMC controller and SM I/O modules as shown in Automaticon 2008

Rys. 1. a) Sterownik programowalny SMC z firmy LUMEL Zielona Góra; (b) Testowy system rozproszony ze sterownikiem SMC i modułami we/wy SM zaprezentowany na targach Automaticon 2008

Typical configuration of SMC-based mini-system involves an SMC controller (master) and several input/output modules (slaves). Usually the modules SM1 to SM5 are used, but other devices can also be connected through RS-485 interface using industry standard MODBUS RTU protocol [6].

However such structure might cause communication bottleneck if too many messages are sent at the same time, so SMC communication subsystem must be flexible enough to overcome this problem. To achieve this, hierarchical coloured Petri net [4] has been developed to model the subsystem (Sec. 3). Coloured Petri nets provide concise system description, efficient tools for analysis, and time extensions. Formalism is fairly broad,

capable of handling large amount of information. Assignment of values to markers is basic extension of CPNs in comparison to classical Petri nets. Here, the CPN net is used at the design stage to examine and simulate the communication subsystem behaviour. As the result, appropriate software modules for the SMC controller has been developed and presented at Computer Networks'08 conference [11].

2. SMC controller software

A prototype design environment CPDev [8] for programming the SMC controller in Structured Text (ST) language of IEC 61131-3 standard [3] is employed. The environment is open what means that the generated code can be handled by other hardware platforms as well. The code is executed by an interpreter that translates instructions of this code into instructions of the machine language. Each of the platforms must have its own interpreter, here called a virtual machine.

2.1. CPDev design environment

Structure of the CPDev environment is shown in Fig. 2. It involves separate logic and hardware layers what simplifies programming and compiling for different hardware platforms.

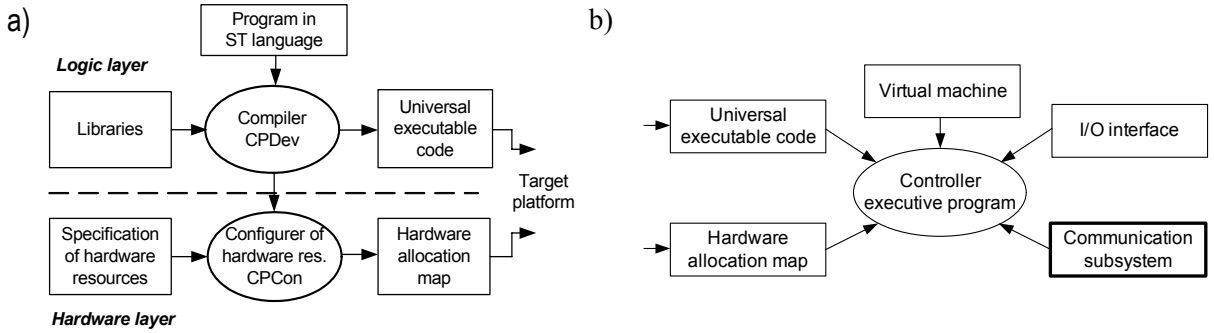

Fig. 2. a) Logic and hardware layers of CPDev environment; b) Deployment of executable software in the controller

Rys. 2. a) Logiczna i sprzętowa warstwa środowiska CPDev; b) Uruchamianie kodu wykonywalnego w sterowniku

The CPDev ST compiler [7] is an essential part of the logic layer. It translates ST programs into the universal executable code interpreted later by virtual machine on the target platform (SMC here). Configuration of hardware resources at the lower layer involves memory, input/output and communication interfaces. This includes memory types and areas, numbers and types of inputs, outputs and communication channels, physical addresses,

validity flags, etc. The CPCon configurer which collect all this into hardware allocation map will be described later.

The CPDev user interface is shown in Fig. 3. Here the main window contains the program PRG_START_STOP written according to ST language rules [5]. MOTOR is turned on immediately after pressing a button START and the PUMP five seconds later. Pressing STOP or activation of an ALARM sensor triggers similar turn off sequence.

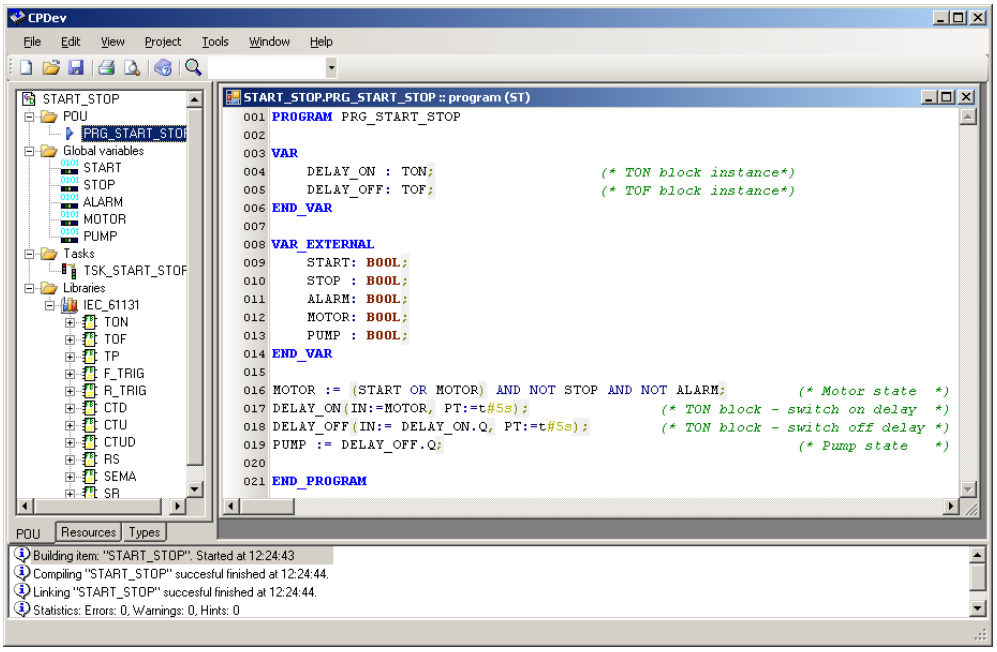

Fig. 3. User interface of the CPDev system
Rys. 3. Interfejs użytkownika systemu CPDev

2.2. Virtual machine

The universal executable code together with the hardware allocation map is transferred to SMC controller where it is executed by the virtual machine. The virtual machine consists of universal and platform-dependent modules (Fig. 4a). The latter modules interface the machine to particular hardware, executing requests to low-level procedures. For example the real-time clock can trigger events that occur at regular, long intervals (e.g., every Sunday midnight). The communication subsystem, which provides an interface to plant input/output fieldbus or PC is described in Sec. 4.

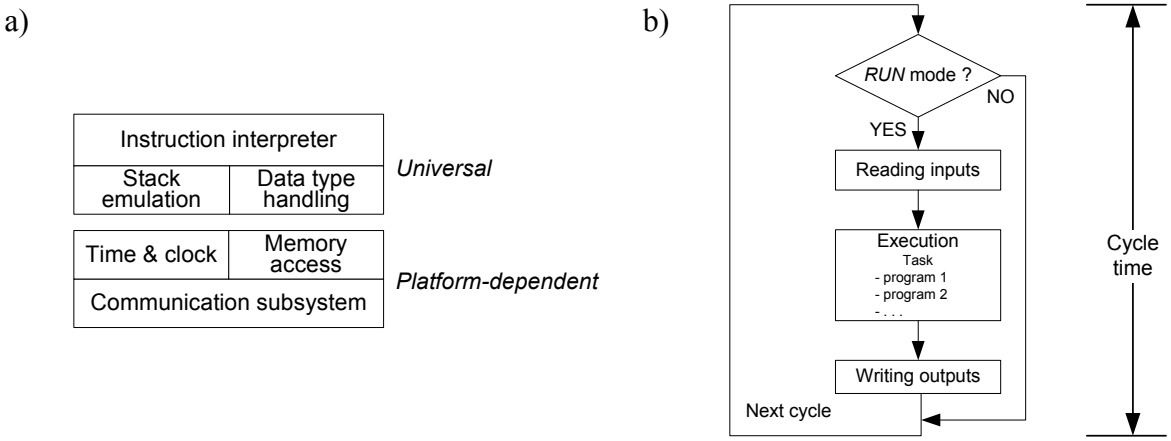

Fig. 4. a) Internal structure of the SMC virtual machine; b) Phases of the machine cycle
Rys. 4. a) Struktura wewnętrzna maszyny wirtualnej SMC; b) Fazy cyklu maszyny

The virtual machine is an automaton operating according to Fig. 4b. While executing the program, the machine fetches successive instruction, decodes it, fetches the operands, and finally executes the instruction. It also triggers communication subsystem procedures responsible for input/output of external variables.

3. Model of SMC communication subsystem

Distributed mini-systems for control and monitoring are able to process only limited number of data transactions. So, the SMC communication subsystem must be flexible enough to overcome potential problems, like bottleneck or blockage.

Design model in the form of hierarchical timed coloured Petri net (HCTPN) [4] is created to examine the subsystem properties. Essential part of the model is shown in Fig. 5. Transitions “Net 1” and “Net 2” represent transmission nets between SMC and slave devices. Here they denote substitution transitions which introduce low level subnets. The hierarchy simplifies modelling and development for nets with multi-layered details. The low-level subnet, corresponding to these transitions, i.e. transmission nets, is shown on Fig. 6.

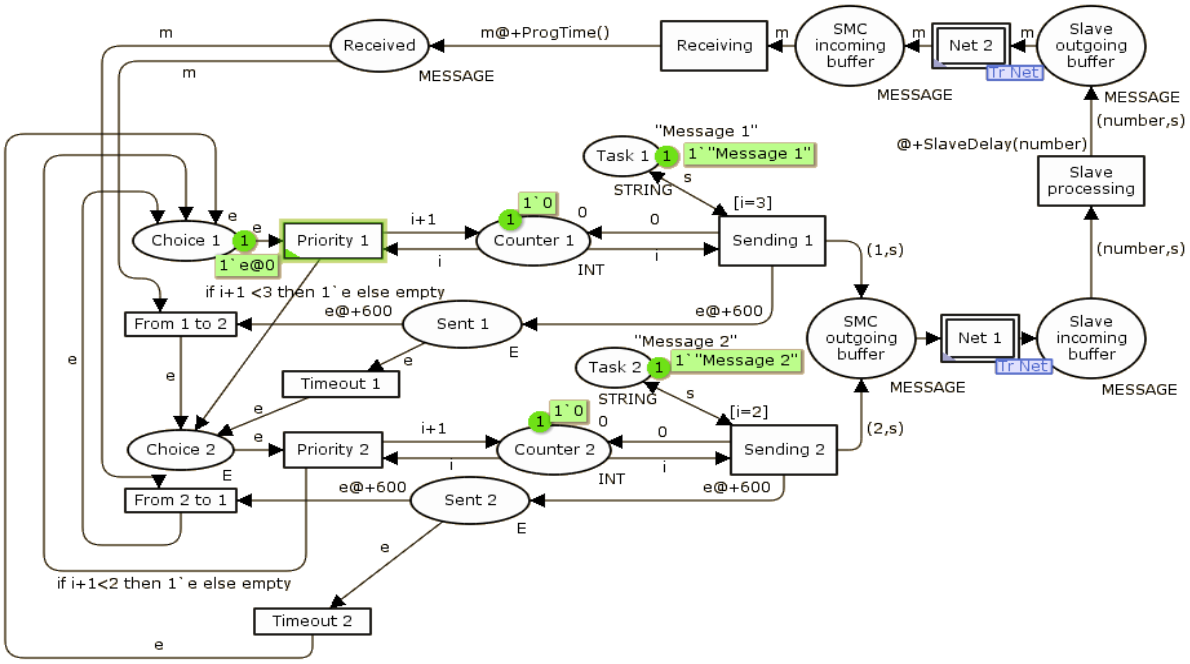

Fig. 5. Coloured Petri net representation of the SMC communication subsystem
Rys. 5. Kolorowana sieć Petriego przedstawiająca podsystem komunikacyjny SMC

Two places, input port “Incoming buffer” and output port “Outgoing buffer”, connect the subnet with the main model. Packets ready to be transmitted by the net are represented by tokens in the “Incoming buffer”. Transition “Net” modifies token timestamp according to the NetDelay function.

```
fun NetDelay(no:INT)=case no of
1 => 4 | 2 => 7 | _ => 0;
```

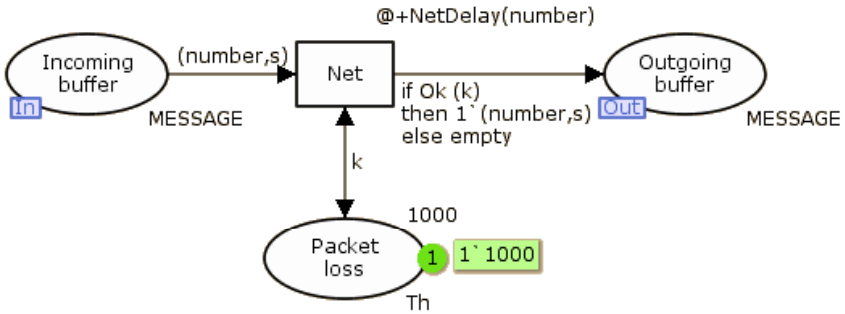

Fig. 6. Subnet modelling the transmission between SMC and slave
Rys. 6. Podsieć modelująca transmisję pomiędzy SMC a urządzeniem podrzędnym

This represents different delays during transmission, depending on the packet. Conditional inscription at the arc from “Net” to “Outgoing buffer” models packet loss with specified probability, similarly as in [9]. Other models of transmission net can also be applied if necessary, however without change of the main model.

The HTCPN model of Fig. 5 represents the SMC communication subsystem. The designer can declare a number of communication tasks, each one represented directly in the

model. Every communication task implements a single transaction with particular slave device (question–answer or command–acknowledgement), which reads or sets a group of registers. For simplification, two communication tasks are presented in Fig. 5. Before the packet stored in "Task 1" or "Task 2" will be sent, one of the task must be chosen. The tasks are executed sequentially, taking into account priorities. Communication subsystem examines priorities until the highest is found. In the example, token being at the place "Choice 1" means that priority of task 1 will be checked. The lower number specified in the guard expression at the transitions "Sending 1" and "Sending 2", the higher priority the task has. Places "Counter 1", "Counter 2" are counters. In the example, the token in the "Counter 1" has value of 0. When transition "Priority 1" will be fired, this value will be increased by 1. Arc from "Priority 1" to "Choice 2" has conditional inscription if i+1<3 then 1`e else empty, so the token will be put in "Choice 2" if value of "Counter 1" is too low to fire "Sending 1". This means, that priority of this task is too low to start execution, so priority of next task will be checked. As it is seen, every attempt to execute a task increases appropriate counter. Eventually one task will be chosen and executed. In our example, after several steps the transition "Sending 2" will be fired, appropriate message put in "SMC outgoing buffer", "Counter 2" zeroed, and appropriate token put in the place "Sent 2". Inscription at the arc between "Sending 2" and "Sent 2" increases timestamp of the token to model the timeout. This means that the transition "Timeout 2" will be active after specified time. Inscription at the arc from "Sending 2" to "From 2 to 1" allows to fire transition "From 2 to 1" before timeout, as soon as reply appears in the place "Received". If packet is lost then response will never be received, so transition "Timeout 2" will be fired after specified timeout time. Packet put in the "SMC outgoing buffer" will be transmitted through the net. Transition "Slave processing" delays response by additional random time. This depends on processing time and is specified below in the SlaveDelay function

```
fun SlaveDelay(no:INT)=case no of
1 => 3+SlaveAddDelay.ran()
|2 => 5+SlaveAddDelay.ran()
|_ => 0;
```

Similarly, the response will be transmitted through the net to the "SMC incoming buffer". Inscription at the arc between "Receiving" and "Received" can delay token, according to the ProgTime function

```
fun ProgTime()=if CycleTime()<100 then 0 else 200-CycleTime()
```

Function CycleTime represents current time of the virtual machine cycle

```
fun CycleTime()=IntInf.toInt (time()) mod 200
```

Communications take place during virtual machine cycles, in the time windows between program executions. In the example, the ProgTime function assumes that single virtual

machine task cycle takes 200 ms, and execution of the program requires 100 ms. Packets received during communication window will be served immediately, while these received during program execution will be delayed, and handled in the next cycle. Eventually the transition “From 2 to 1” will be fired, what means that second communication task has been executed successfully. Then, in the same way next task for execution is chosen.

The HTCPN model has been examined using CPN Tools [2]. Simulations have revealed that the concept of the tasks and priorities can be used to manage numerous communication requests. Up to several dozen of communication tasks can be implemented into the model and analysed, which is more than enough in typical, small SMC-based distributed system.

Reading two or more separated variables from the same slave device can be achieved in several transactions (one transaction per each register) or in a single transaction related to the continuous set of registers (where possibly few excessive registers have to be read in order to make this set also continuous, and will be discarded later). Which of the two ways is more efficient depends on location of the variables in the slave memory (number of excessive registers), baud rate (additional transmission time), slave processing delay, program cycle time, etc. Simulation of the model helps to choose more convenient way by estimating behaviour of the SMC controller in particular case.

Simulation results can be used to calculate average and maximum response times. They depend on task count, priorities, baud rate, probability of packet loss, register grouping, etc. In this way one can predict possible bottlenecks in the early stages. Such simulation has been performed for the START_STOP example (Sec. 2.1). The results have shown e.g. that average response time is around 25 ms for grouped register readout (2 tasks with equal priorities). Reading all variables separately would take much longer. Response time of real system for typical conditions turned out to be similar.

4. Implementing the SMC communications

The mini-DCS considered here involves SMC programmable controller, I/O modules of SM series or other devices from LUMEL, and a PC with CPDev or SCADA-type packages (Fig.1b). Modbus RTU protocol is employed on both sides of the SMC.

The Petri net model of communication subsystem has become a basis for development of software modules responsible for communication. CPCon is a configurer of hardware resources, adapted for the SMC communication. Its tasks are: configuration of communication between SMC and SM I/O modules, setting transmission parameters for communication with PC, creation of file with hardware resource allocation map and downloading the files with executable code and map to the SMC.

Communication between SMC and SM slaves is defined by creating suitable communication tasks in CPCon window (Fig. 7).

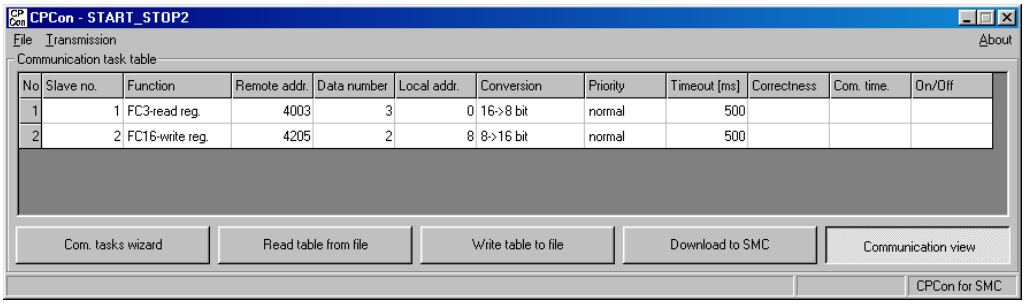

Fig. 7. Communication configuration using CPCon
Rys. 7. Konfiguracja komunikacji w programie CPCon

Communication task table determines communication tasks, i.e. what question–answer and command–acknowledgement transactions take place between SMC and SMs. Tasks are represented by the rows in the table and can be defined manually, half-automatically by the wizard, or automatically through instances of communication blocks declared in ST program. Every task is associated with specified variables (registers) in particular slave.

The first row specifies communication between SMC and SM5 binary input module. SM5 is connected to pushbuttons which, in case of the START_STOP project, set the variables START, STOP and ALARM (Fig. 3). In SMC these variables have successive addresses beginning from 0000. SM5 places the inputs in consecutive 16-bit registers starting from 4003. So all the variables can be read in a single Modbus transaction with the code FC3 (read group of registers [6]). However, since BOOL occupies single byte in CPDev, the interface of the SMC virtual machine has to perform 16- to 8-bit conversion.

Second row of the task table defines communication with the SM4 binary output module. Two consecutive variables, MOTOR and PUMP, the first one with the local address 0008, are sent to SM4 in single message with the code FC16 to remote addresses beginning from 4205 (write group of registers). This time 8- to 16-bit conversion is needed.

Communication tasks are carried out sequentially, according to their slave numbers and priorities. Programs executed on the virtual machine can also enable or disable each task selectively, and check communication status (correctness, communication time). Tasks are handled by SMC during pauses that remain before end of each cycle, after execution of the program. Single transaction takes 10 to 30 ms, depending on the speed (max. 115.2 kb/s). If the pause is large, the task can be executed a few times. It has been assumed that the task with NORMAL priority is executed twice slower than the task with HIGH priority, and the task with LOW priority three times slower. As seen in Fig. 7, the communication with both modules has NORMAL priority. The timeout is set to 500 ms.

5. Conclusions

Formal models can be used during design phase to examine controller software behaviour [10]. Such model for SMC-based small distributed control-and-measurement system, involving hierarchical timed coloured Petri net has been presented. The model takes into account delays and timeouts. The concept of communication tasks and priorities has been introduced. The simulation of the model helps to predict possible bottleneck problem in the early stages of development.

The model has been examined and simulated. The results have supported development of the SMC software, i.e. configurer CPCon and communication module of the virtual machine.

REFERENCES

1. ATmega 128 Datasheet. Atmel, 2007.http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf.
2. CPN Tools: Computer Tool for Coloured Petri Nets. http://www.daimi.au.dk/cpntools
3. IEC 61131-3 standard: Programmable Controllers Part 3, Programming Languages. IEC, 2003.
4. Jensen K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Springer –Verlag, 1997.
5. John K.H., Tiegelkamp M.: IEC 61131-3: Programming Industrial Automation Systems. Springer-Verlag, Berlin - Heidelberg 2001.
6. Modicon MODBUS Protocol Reference Guide. MODICON, Inc., Industrial Automation Systems, Massachusetts 1996 http://www.modbus.org/docs/ PI_MBUS_300.pdf.
7. Rzońca D., Sadolewski J., Trybus B.: IEC 61131-3 standard ST compiler into universal executable code. In: Real-Time Systems. Methods and Applications, WKŁ, Warsaw 2007, p. 189÷198 (in Polish).
8. Rzońca D., Sadolewski J., Trybus B.: Prototype environment for controller programming in the IEC 61131-3 ST language, Computer Science and Information Systems, 12/2007.
9. Rzońca D., Trybus B.: Modelling of safe communication link initialization in master-slave protocols by using coloured Petri nets, in: S. Węgrzyn, J. Klamka, S. Kozielski, T. Czachórski, L. Znamirowski (Red.): Sieci komputerowe, Nowe technologie (in Polish), WKŁ, Warszawa 2007, p. 247÷254.
10. Trybus B.: Introduction to Conversion of Control Software Structured Models into Coloured Petri Nets, Theoretical and Applied Informatics, Vol. 19 2007, no.1, p. 57÷70.

11. Rzońca D., Trybus B.: Timed CPN model of SMC controller communication subsystem, in: S. Węgrzyn, T. Czachórski, A. Kwiecień (Eds.): Contemporary Aspects of Computer Networks, WKŁ, Warszawa 2008, p. 203÷212.

Recenzent: Prof. dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 25 września 2008 r.

Omówienie

Sterownik SMC jest produkowany przez LUMEL Zielona Góra (rys. 1). Jego oprogramowanie zostało opracowane na Politechnice Rzeszowskiej. Urządzenie jest programowane w środowisku CPDev (rys. 3) za pomocą języka ST normy IEC 61131-3 [3]. Maszyna wirtualna uruchomiona na SMC umożliwia wykonywanie kodu, będącego rezultatem kompilacji programu w ST (rys. 2, 4).

SMC nie posiada własnych wejść i wyjść obiektowych, jest natomiast używany jako centralny moduł (*master*) w niewielkich rozproszonych systemach sterowania i monitorowania (mini-DCS). W typowej konfiguracji współpracuje z modułami wejść/wyjść typu SM, lecz standard MODBUS [6] umożliwia dołączenie innych urządzeń *slave*.

W niektórych przypadkach taka struktura systemu rozproszonego może wiązać się z istnieniem „wąskiego gardła", gdy zbyt wiele komunikatów wysyłanych jest w tym samym czasie. W celu zminimalizowania tego problemu zdecydowano się opracować odpowiednio elastyczny podsystem komunikacyjny. W pierwszej fazie opracowano model w postaci hierarchicznej czasowej kolorowanej sieci Petriego (HCTPN) [4]. Rys. 5 przedstawia główną sieć modelującą podsystem komunikacyjny SMC, w której przejścia "Net 1", "Net 2" reprezentują komunikację z urządzeniami podrzędnymi. Transmisja ta została uszczegółowiona w podsieci niższego rzędu (rys. 6), która uwzględnia opóźnienie transmisji oraz prawdopodobieństwo utraty pakietu. Projektant może zadeklarować odpowiednią ilość zadań komunikacyjnych o różnych priorytetach, z których każde odpowiada pojedynczej transakcji z urządzeniem *slave*. Model uwzględnia priorytety zadań, przekroczenie czasu (*timeout*) oraz czas cyklu programu. Wprowadzono także zróżnicowanie opóźnienia odpowiedzi urządzenia pośredniego.

Utworzony model poddano analizie pakietem CPN Tools [2]. Wyniki symulacji pokazały, że koncepcja zadań i priorytetów może służyć do zarządzania wieloma żądaniami komunikacyjnymi w niewielkich systemach DCS. Na etapie projektowania symulacja

modelu pozwala estymować zachowanie podsystemu komunikacyjnego, m.in. szacować średni i maksymalny czas odpowiedzi dla różnych konfiguracji.

Model formalny stał się podstawą opracowania oprogramowania podsystemu komunikacyjnego sterownika, a także narzędzia konfiguracyjnego CPCon (rys. 7), które realizują koncepcję zadań komunikacyjnych i priorytetów.

Addresses

Dariusz RZOŃCA: Rzeszow University of Technology, Department of Computer and Control Engineering, ul. W. Pola 2, 35-959 Rzeszów, Poland, drzonca@prz-rzeszow.pl
Bartosz TRYBUS: Rzeszow University of Technology, Department of Computer and Control Engineering, ul. W. Pola 2, 35-959 Rzeszów, Poland, btrybus@prz-rzeszow.pl