NETWORK DRIVER FOR COMMUNICATION IN SPV INDUSTRIAL NETWORK

Georgy Slavchev Mihov[1], Stanimir Damyanov Mollov[1], Ratcho Marinov Ivanov[1], Stoyan Nikolov Jilov[2]

[1] Faculty of Electronic Engineering and Technologies, TU – Sofia, 1797, Sofia, Bulgaria, e-mail: gsm@tu-sofia.bg, smollov@abv.bg, rmi@tu-sofia.bg

[2] SPV Ltd. Rakovsky 135, 6000, St. Zagora, Bulgaria, e-mail: spv@abv.bg

Keywords: network driver, industrial network, queue, buffer, message stream

In the present paper the development of a network driver for communication in SPV industrial controllers has been discussed. The hardware/software approach for realisation of the network driver is offered. The network driver realises all functions of sending and receiving, protecting times in industrial network, the repeating of the message when an error occurs and the solving of the conflict situations. In additional, the organisation of the receiving and the transmitting of the stream of messages are discussed. Possibilities for existing conflict situations have been analysed. The variants for their overcoming are offered. The proposed network driver could be used as a universal solution for industrial communication driver, which is moved outside of a personal computer.

1. INTRODUCTION

A driver for local array networking for industrial purposes have to provide all functions of sending and receiving of messages, supporting of protected times, repeating of messages in case of errors occur etc. [1]. Two ways for network driver realisation exists: almost **software** and combined **hardware/software**. In the software way of the realisation, the network driver is fully built in the controller environment, and the operating system of the controller serves it. In the hardware/software way of the realisation an additional processor for the communication supporting is included (communication processor).

The connection between the main processor and the communication processor could be realised as followed:

– Via a common memory or a common system bus;
– Via parallel interfaces (ISA, PCI);
– Via serial interfaces (RS232, USB);

In the present paper a variant of network driver realised by using of a serial interface RS232 is offered. The network driver is used for connecting the personal computer to the SPV industrial network. The physical environment for communication in this network is based on interface RS485. Due to this solution the network driver have to be included in the structure of a bridge converter RS232 – RS485. The existing solutions of such converters are strictly specialized, depending of the specific protocol used for communication. The proposed solution of the network driver provides the connection of the personal computer to the industrial

network to be based on the SPV protocol [2].

The messages in the network may be received one after another with a faster speed than the system is able to process them. That requires the network driver to provide special buffers for messages. Usually FIFO buffers, organized as queues, are applied for message buffers. In the present paper, besides the basic actions with the queue, so-called procedure for **holding the new message** when the queue is full is included.

Each message is written in additional (input) buffer before to be entering into the queue. The next new message is received after the old message from this buffer has been transferred to the queue. In this situation a silent time before receiving a new message is inserted. A new solution, which decreases this silent time, is offered.

2. ENTRY POINTS OF THE NETWORK DRIVER

The network driver has the following entry points that are shown on Fig. 1.

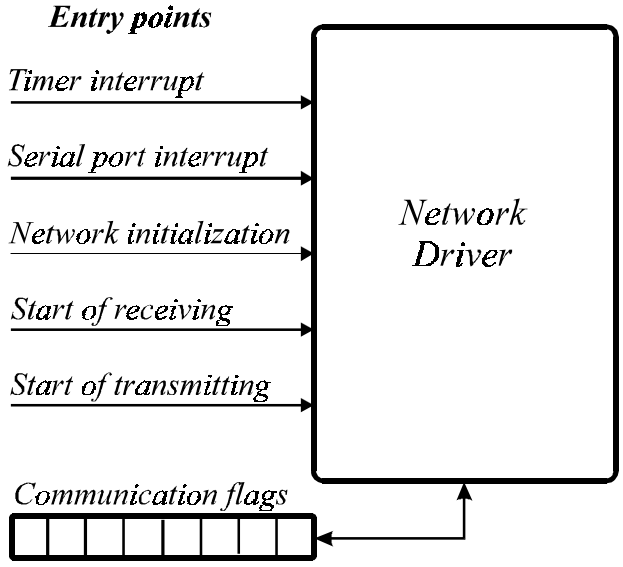

Fig. 1. Entry points of the network driver.

Interrupt from the system timer. The control is given in this point in case of an interrupt from the system timer is received. This entry point just modifies the counters of protected times and checks whether protected times are passed or not.

Interrupt from the serial port. This is the interrupt from the serial port, which submits request when a symbol is received by the network or when a symbol is sent via the network.

Initialisation of the network – This entry point is being used for initial initialisation of the serial port. The receiver and the transmitter are set in inactive states.

Start of receiving – This entry point sets the address of the input buffer and its maximum length and also sets the receiver in an active mode. After that the receiver starts to watch the line and to receive the messages for the station. After some message is received the receiver marks this fact by setting appropriate flags in the state's word and goes into end of receiving mode.

Start of transmission – This entry point is activated only in case of inactive transmitter (the transmission of the last message is finished). The address of the output buffer, the length of the message and the recipient are initialised. The transmitter checks whether the line is busy or not and when the line is being free for some time it starts the transmission of the message. After the transmission of the message is finished the transmitter sets appropriate flags in the state's word and goes into inactive mode.

3. HARDWARE REALIZATION OF THE NETWORK DRIVER

The necessary parameters of the microprocessing system for realisation of the offered network driver are **microcontroller, two serials port, RAM with size about 16KB (for input and output queues)** and **ROM (for the program memory).**

In the present solution a microprocessing system based on microcontroller MC68HC11 is used (Fig. 2). The used microcontroller has only one embedded serial communication interface (SCI). The solution needs two SCI. There exists two ways for the realisation of a secondary SCI:

– **Software emulation of the SCI;**

– **Hardware/software approach by using a secondary microcontroller**.

The software emulation of the serial port requests a lot of time resource of the processor. This is the reason that in the present solution a hardware/software approach has been used. Microcontroller PIC16F876 is chosen for secondary CSI realising. The connection between two microcontrollers is realised via SPI interface. A galvanic isolation between personal computer and industrial network is applied with appropriate opto-couples. The power supply of the galvanic isolated part from the PC side is supported by an external source. The industrial network supplies another part of the network device.

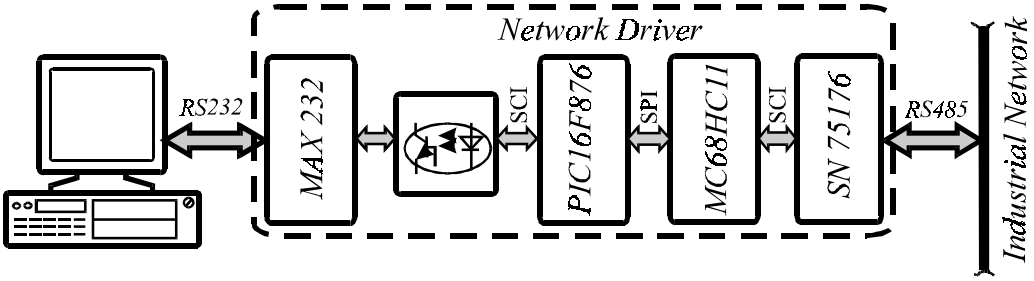

Fig. 2. Hardware/software building of the network driver device.

4. SOFTWARE REALIZATION OF THE NETWORK DRIVER

Multitask system controls the driver for the local array network. Basic program units of the network driver are **actions**. Action is an independent program that works under management of the network driver. Basic communication units of the network driver are **events**. Each action waits a specific event or generates an event.

4.1 Data structures for communications between actions

Used data structures for communications between actions are **flags** and **queues.** Flags are realised as states of bits of a byte in the data memory. Queues are used for message exchange between actions. Queues are buffers between actions. They prevent the data lost in case of message received before the previous is treated.

A queue is a data structure that consists of memory array and two pointers: input pointer and output pointer (Fig. 3). The movement of pointers is accomplished just in one direction. When the pointer goes to the last address of memory array it comes back to the beginning of the memory array. Basic actions of the queue are:

– Entering a message into the queue;

– Bringing out a message from the queue;

– Checking-up for the empty queue;
– Cleaning the first (oldest) message from the queue.

A critical situation may occur when the free space in the queue is smaller than the message length. In this case the action that sends the message may proceed in two ways – refusing the processor time and waiting for freely space into the queue or cleaning the oldest messages and entering the new message into the queue.

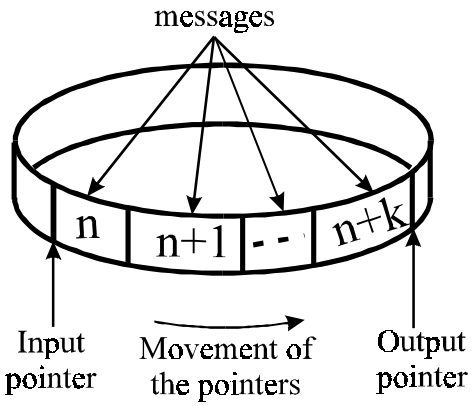

Fig. 3. Queue functioning.

The network driver of a slave station is allowed to clean the first message in the queue in case of full queue. Otherwise the network driver of the master station is not allowed to reject the oldest message. In this case so-called **holding the new message** is applied. After receiving the new message the action-receiver checks its length and if the free space in queue is enough for the message. If the space is enough the message is put into the queue and a positive acknowledge is sent to the transmitted station. Otherwise, in case of space deficiency negatives acknowledge is sent to require the transmitted station for the last message repeating.

4.2. Transmitting the messages via network

The developed network driver is applied for the SPV industrial network [1]. If the sending message is big and does not fit in one transmitted block its transition is done in several blocks. The separation in small-transmitted blocks has an advantage if a transmission error occurs only the wrong block has to be transmitted again.

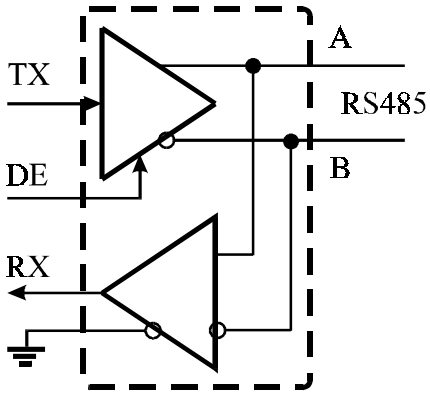

Fig. 4. Always enabled receiver.

All actions of the network driver have their protected times. If the sender does not receive a positive acknowledge in a definite time it sends again the current block. The driver processes each received symbol. In the receiving mode it watches whether the line is busy and whether the message is destined for this station. In the transmission mode the driver watches also whether the received symbol is the same to the sent one. A conflict situation may occur when more than one station starts transmission at the same time. To get out this conflict each sender listens and compares for a right transmission (Fig. 4). In case of difference the sender interrupts the transmission and sends a BREAK signal (supports the conflict). Each sender restores the transmission after a protected time that depends of the station address.

To increase the effectiveness of the data exchange speed the input buffer of the network driver is realised as shown on Fig. 5. After each received message, the network driver changes alternatively input buffers. So, a new message is allowed to arrive before the previous message is translated from input buffers to the input queue.

5. STREAM MESSAGES ORGANISING

The processing of the stream messages is done coincidentally with the system's normal work. For receiving, buffering, transmitting and processing of messages several actions are carried, as shown in Fig. 6. Queues A and B are used for the communication between the actions.

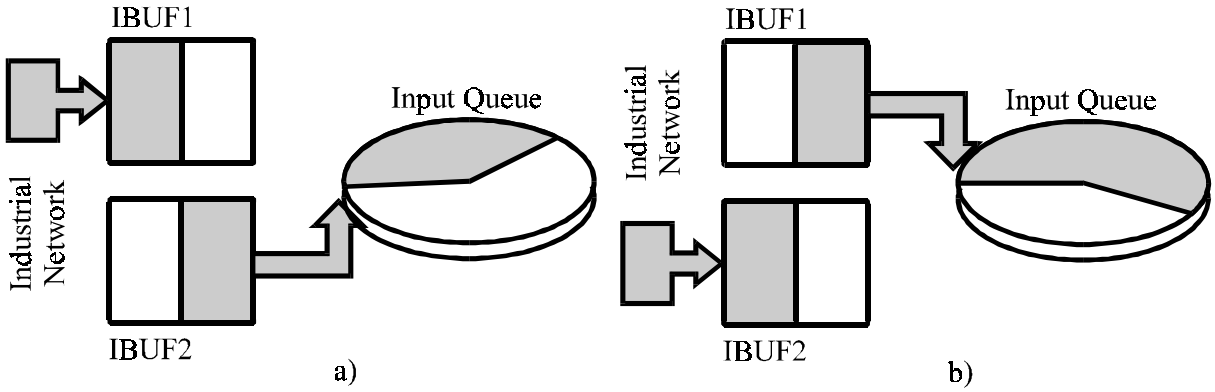

Fig. 5. Input buffers organising.

5.1. Working actions under network driver management

Receiver – This action activates the receiver of the network driver and checks the message after its receiving. If there is no message the action-receiver gives up its processor time to the other actions. If a message presents in the input buffer IBUF the action transfers it into the queue A.

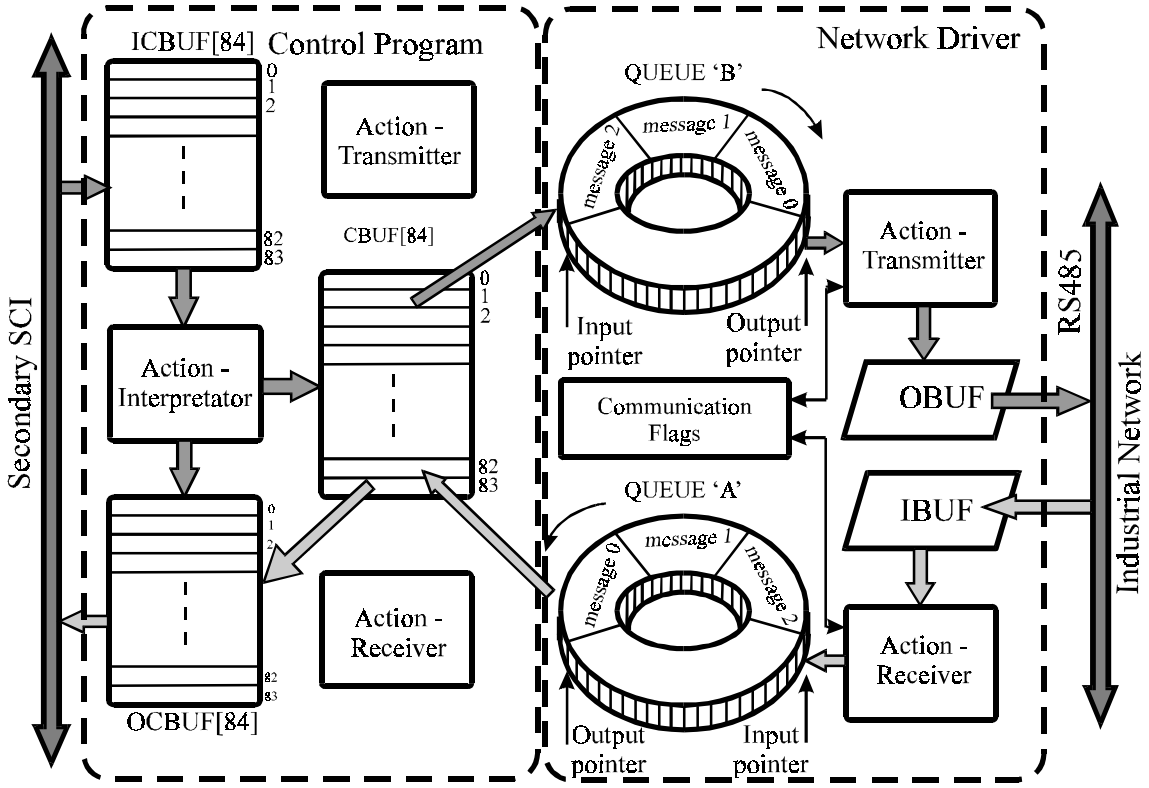

Fig. 6. Treating of messages in the industrial network.

Transmitter – This action checks whether the queue B is empty or not. If empty the action gives up its processor time to the other actions. If the queue B is not empty

a check is done whether the transmitter of the driver is active. If the transmitter is active the action-transmitter waits while the transmission of the last message will finish. After the finish the message is translated to the output buffer OBUF and the transmitter of the driver comes to the active mode.

5.2. Working actions under controlled program management

Interpreter – This action checks whether the input buffer ICBUF is empty. If it is not empty the action-interpreter gets the message and processes it. The message may be addressed to the main station or to a slave station. If the message is addressed to the main station the action-interpreter prepares a reply message and sends it into OCBUF. Otherwise this action translates the message to the CBUF.

Receiver – This action checks whether a message is presented in CBUF. If no message presents the action-receiver checks whether in the queue B has enough free space. If there is not enough free space for the message, a negative acknowledges is replied. If the space is enough the action extracts the message and put it into the queue B and a positive acknowledge is replied.

Transmitter – This action checks whether the queue A is empty or not. If not empty the CBUF is checked whether it is full and if it is not full the action extracts the oldest message from the queue A and put it into the CBUF.

6. CONCLUSIONS

The proposed network driver realises all sending and receiving functions, protected times, repeating when an error occurs. It could be used as universal solution for industrial communication driver that is moved outside of a personal computer.

The developed driver has been used as equipment for the master station of SPV industrial network. Therefore, the driver does not allow cleaning the oldest message from the queue, when the queue is full. That requires supporting holding mode for the new message.

To increases the effectiveness of the data exchange speed, the receiver of the network driver is activated before message to be transferring from input buffer IBUF to the input queue A.

7. REFERENCES

[1] Tashev, I. *Methods, devices and systems for collecting and transformation of an information.* Book for remote education. Teshnical University - Sofia, 1997 (in Bulgarian).

[2] Dimitrov, E., G. Mihov, I. Tashev, M. Mitev, *Local array network for industrial controller*, Proceedings of the Int. Scientific Conference ENERGY AND INFORMATION SYSTEM AND TEHNOLOGIES, . vol. 3 pp. 608-613, June 7- 8, 2001, Bitola, Macedonia.

[3] Herbert, F. *Implementing Network Protocols and Drivers – with STREAM.* Embedded Systems Programming, p.28, April, 1997.

[4] Herhandez, E., Chidester, and A. George, "*Adaptive Sampling for Network Menagement"*, Journal of Network and System Management, Journal Publ. from the HCS Research Lab, 1999.

[5] Ovcharov, St. *Automation of the electronic industry*. Teshnical University – Sofia, 2004.

The paper has been reviewed by Associate Professor Ph.D. Stefan Ovcharov from department of Electronics, Technical University – Sofia.