[Dulger* 6(7): July, 2019] ISSN 2349-4506
Impact Factor: 3.799

Global Journal of Engineering Science and Research Management

DESIGN AND IMPLEMENTATION OF A NEW DATA TRANSMISSION PROTOCOL, *SPI-RING PROTOCOL*, CONNECTING TWO SINGLE MD100-PLC PROGRAMMABLE LOGIC CONTROLLERS VIA SPI INTERFACE TO FORM MODULAR MD100-PLC SYSTEM.

Mustafa Dülger
Mechanical Engineering Department, Faculty of Engineering, University of Istanbul Cerrahpasa, Istanbul 34320, Turkey

DOI: 10.5281/zenodo.3345109

KEYWORDS: Data, Transmission, Protocol, SPI-Ring Protocol, PLC-Machine, KUMANDA, MD100-PLC

ABSTRACT

In this paper, bi-directional data transmission protocol between two micro-controller boards connected via SPI-Interface is presented. SPI-Ring Protocol is developed as a new transmission protocol for this purpose. In this connection, one of the controller acts as a master device while the other acts as a slave device.

SPI-Ring protocol is mainly developed in order to connect two MD100-PLCs [MD100 Programmable Logic Controller]. Total I/O lines of the MD100-PLC become sometimes insufficient for the process under control. Connection of two MD100-PLCs forms modular MD100-PLC system having doubled I/O lines and suggests a reliably solution for the cases where the I/O lines of a single MD100-PLC are not adequate.

The SPI-Ring Protocol is fully implemented in software for both master and server side devices in assembler language. Two test cases are created to test the SPI-Ring Protocol. Snapshots from the KUMANDA (The complete Program Development Environment for MD100 Programmable Logic Controllers developed in C++) showing the test programs are given. The programs are executed on the modular MD100-PLC system. States of the IO lines of the modular PLC-System are indicated in separate photos. The results have proven full satisfaction.

INTRODUCTION

PLC is the abbreviation for *Programmable Logic Controller*. Among others, number of input/output channels, volume of program memory, volume of static RAM and micro-controller tact frequency determine resources of the PLC. It is quite clear that the more the resources, the powerful the PLC is.

Conventionally a PLC device has two base modules. They are the central processing unit (CPU) and I/O modules. For small PLC devices, micro-controller input/output lines cover the PLC I/O channels. I/O modules are not designed separately but put on the periphery of the micro-controller. For such PLC devices, micro-controller acts as CPU and micro-controller port lines act as I/O module so that base modules are embedded together. Such a system is called as a minimum PLC system.

A practical way of doubling resources of minimum PLC system is to connect two identical minimum PLC systems over a serial bus with a definite protocol. MD100-PLC controller is developed as a minimum PLC device.

In the next section, two MD100-PLC modules, connected through SPI interface in order to form modular MD100-PLC system, is presented.

MODULAR MD100-PLC SYSTEM

MD100-PLC controller is developed as a minimum PLC device. Resources of MD100-PLC controller are adequate for many applications in industry. There are however plenty of cases where a minimum MD100-PLC controller is not capable of covering all needs. Especially I/O lines may not match for the requirements of the application. In those cases there is no choice other than replacing the PLC with that having more I/O lines.

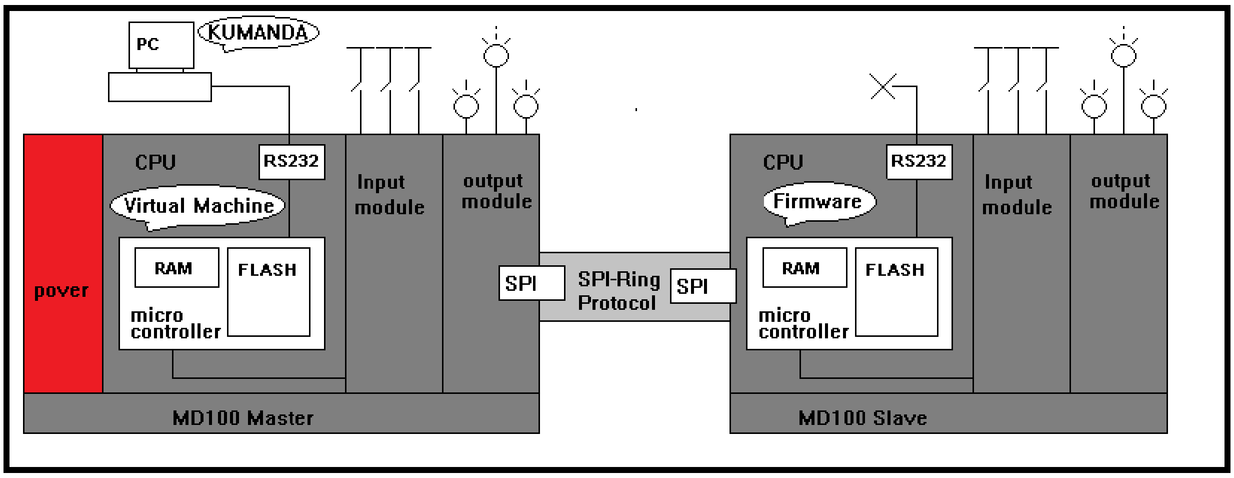


Fig.1: Schematic of the modular MD100-PLC System

A practical and economical solution to the problem is presented in which two MD100-PLCs are connected via SPI interface forming a modular PLC-System with increased I/O lines. Schematic layout of the two MD100-PLCs connected via SPI interface is given in Fig.1. They form together master-slave modular PLC-System with increased I/O lines.

In this system, one of the controller runs as a master and the other as a slave. KUMANDA is the *Integrated Development Environment* for creating PLC program [1]. It runs on the programming device (PC). The PLC program governing the process under control is loaded into the master device over RS232.

The *SPI-Ring Protocol* is developed as the data transmission protocol between master and slave devices. *Virtual PLC-Machine* is pre-installed on the master device [2]. It runs the master-side implementation of the *SPI-Ring Protocol*. A permanent *firmware* program running on the slave device runs the slave side implementation of the *SPI-Ring Protocol*.

Scan Cycle of the Modular MD100-PLC System

PLC-Scan cycle for the modular PLC system differs from that for the minimum PLC system. The PLC-Scan cycle for the modular PLC-System is illustrated in Fig.2.

PLC-Scan cycle for the modular PLC-System consists of two separate scans; one is the *MASTER-SCAN* running on the master device and the other is the *SLAVE-SCAN* running on the slave device. *MASTER-SCAN* does the same as in the PLC-Scan cycle in a minimum MD100-PLC except that it builds connection in *SYSTEM-SCAN* with the slave device by using the *SPI-Ring Protocol*. Master device uses this connection to perform the data transmission in order to exchange system variables among master and slave devices.

The output variables OR00 and OR01 of the slave device are assigned to the output variables OR02 and OR03 of the master device. Similarly the input variables IR00 and IR01 of the slave device are assigned to the input variables IR02 and IR03 of the master device. This implies that from within the application program, the slave system variable IR00 is accessed through master system variable IR02 and the slave system variable IR01 is accessed through the master system variable IR03. In a similar manner, the slave output variables OR00 and OR01 are accessed through the master output variables OR02 and OR03 respectively.

The analog output variables AO00 and AO01 of the slave device are assigned to the analog output variables AO02 and AO03 of the master device. The analog input variables AI00 and AI01 of the slave device are similarly assigned to the analog inputs variables AI02 and AI03 of the master device. All analog variables are two-byte long and they are not shown in Fig.2.

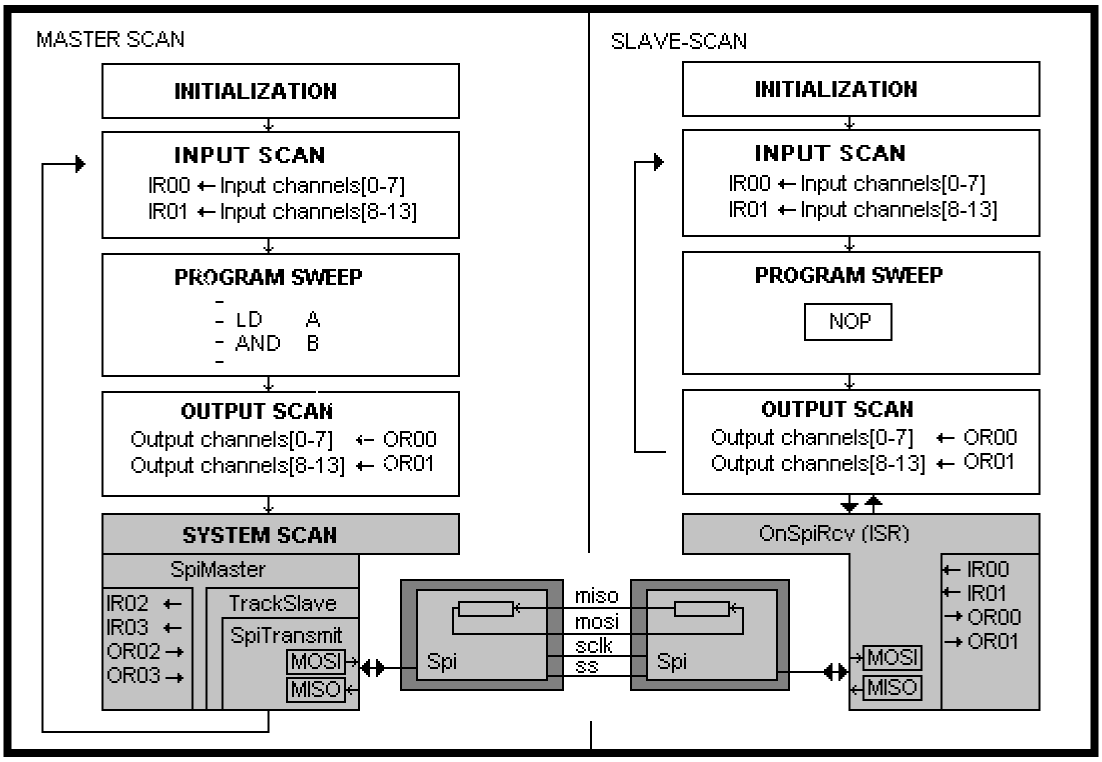


Fig.2: PLC-Scan Cycle for the modular MD100-PLC System

Latest values of the output channels of the slave device are recalculated in the *PROGRAM-SWEEP* of the *MASTER-SCAN* and stored into the respective master system variables. The content of master system variables OR02 and OR03 are transmitted to the slave device over the SPI-interface using *SPI-Ring Protocol* in order to update the slave system variables OR00 and OR01. Slave output channels are updated through the slave system variables OR00 and OR01 in the *OUTPUT-SCAN* of the slave device.

The current values of the input channels of the slave device are read in the input scan of the slave device and stored into the slave system variables IR00 and IR01. The content of slave system variables IR00 and IR01 are simultaneously transferred back to the master device as the slave receives master output variables OR02 and OR03. These returned variables are used by the master device to update master input variables IR02 and IR03 respectively. The *PROGRAM-SWEEP* of the *SLAVE-SCAN* incorporates only NOP (no operation) plc-command which does no real operation.

Register File of the MD100-PLC

Because *SPI-Ring Protocol* operates on the register file of the MD100-PLC device, a little bit more explanation of the register file is needed. The register file is nothing else than the collection of the system variables. The system is actually a virtual PLC machine running on the master and slave device. Each variable is either one or two byte long RAM-cell or working register of the micro controller [3] [4] [5].
Description of the registers, relevant to the *SPI-Ring Protocol* in the Register File of the MD100-PLC device, is given in Table.1.

Table.1: Register File of MD100-PLC

Register File						
Register name		**Register index**	**Working register**	**Ram address**	**Length**	**Description**
IR00		00	-	0x0060	1 byte	Digital input register–00. Keeps signal level of digital input channels 0-7
IR01		01	-	0x0061	1 byte	Digital input register–01. Keeps signal level of digital input channels 8-15
IR02		02	-	0x0062	1 byte	Digital input register–02. Keeps signal level of digital input channels 16-23
IR03		03	-	0x0063	1 byte	Digital input register–03. Keeps signal level of digital input channels 24-31
OR00		00	-	0x0064	1 byte	Digital output register–00. Keeps signal level of digital output channels 0-7
OR01		01	-	0x0065	1 byte	Digital output register–01. Keeps signal level of digital output channels 8-15
OR02		02	-	0x0066	1 byte	Digital output register–02. Keeps signal level of digital output channels 16-23
OR03		03	-	0x0067	1 byte	Digital output register–03. Keeps signal level of digital output channels 24-31
DATR		-	R01	-	1 byte	Work register of micro controller. It is used as a temporary variable to assist execution of commands in slave device
MOSI		-	R14	-	1 byte	Work register of micro controller. It is used to keep MOSI byte.
MISO		-	R15	-	1 byte	Work register of micro controller. It is used to keep MISO byte.
X	XL	-	R26		1 byte	Internally used. It has low (XL) and high (YH) parts each of which is one byte long. It is used as a pointer to address RAM and/or flash memory.
	XH	-	R27	-	1 byte	
Y	YL	-	R28	-	1 byte	Internally used. It has low (YL) and high (YH) parts each of which is one byte long. It is used as a pointer to address RAM and/or flash memory.
	YH	-	R29	-	1 byte	
VALU		-	-	0x024E	1 byte	Keep value to be transmitted to the slave
TARE		-	-	0x024F	1 byte	Keep target register code which is to be updated in slave device by the value kept in register TARE
RTVA		-	-	0x0250	1 byte	Keep the value returned from the slave device
AI00		00	-	0x023D	2 byte	Analog input register low byte
		01				Analog input register high byte

AI01	02	-	0x023F	2 byte	Analog input register low byte
	03				Analog input register high byte
AI02	04	-	0x0241	2 byte	Analog input register low byte
	05				Analog input register high byte
AI03	06	-	0x0243	2 byte	Analog input register low byte
	07				Analog input register high byte
AO00	00	-	0x0235	2 byte	Analog output register low byte
	01				Analog output register high byte
AO01	02	-	0x0237	2 byte	Analog output register low byte
	03				Analog output register high byte
AO02	04	-	0x0239	2 byte	Analog output register low byte
	05				Analog output register high byte
AO03	06	-	0x023B	2 byte	Analog output register low byte
	07				Analog output register high byte
SLTY	-	-	-	1 byte	slave identification (type)
SLOF	-	-	-	1 byte	slave identification (offset)

Each variable has a symbolic register name and unique RAM address. Input and output registers have register index which enumerates them internally. They are spoken over their register index by the *SPI-Ring Protocol*. Registers, which are directly assigned to micro controller working registers R0 to R31, are accessed directly by assembler commands.

SPI-RING PROTOCOL

The *SPI-Ring protocol* is so designed that it encodes and decodes the data transmitted over the SPI-interface through the *SPI-Ring protocol* into the protocol commands. The master device encodes the single *SPI-Ring protocol* command and sends them to the slave device. The slave device decodes the received command and then executes it.

Master device initiates the transmission by sending the content of the *MOSI* register to the slave device over the *mosi (master out slave in)* line. Before transmission, the content of the *MOSI* register is to be updated by the master device *(Virtual-Machine)*. As the content of the *MOSI* register is being transmitted over the *mosi* line, the master device receives data from the *miso (master in slave out)* line. The received data is placed into the *MISO* register of the master device at the end of the transmission.

Slave device is requested by setting /SS signal low by the master device. As the slave device receives data over the *mosi* line, it sends back the content of its *MISO* register over the *miso* line simultaneously. The content of the *MISO* register must therefore be updated by the slave device *(firmware)* before it is transmitted.

SPI-RING COMMANDS

Table.2 lists valid *SPI-Ring Protocol* commands. Each *SPI-Ring Protocol* command is one byte long. It is stored in the MOSI system variable of the master device before it is transmitted. After reception of the command, slave device stores it in the MISO system variables and executes it. Slave device should be given an adequate time in between following commands so that it can execute the preceding command before receiving the next.

Table.2: SPI-Ring Protocol commands

SPI-Ring Commands (Master)											**Operation in Slave**
Name		Opcode	Command Bits								
			C7	C6	C5	C4	C3	C2	C1	C0	
SBC [Sub Commands]	S0	0x60	0	1	1	-	0	0	0	0	reserved
	S1	0x61	0	1	1	-	0	0	0	1	MISO = SLTY
	S2	0x62	0	1	1	-	0	0	1	0	MISO = SLOF
	S3	0x63	0	1	1	-	0	0	1	1	SLOF = DATR
	S4	0x64	0	1	1	-	0	1	0	0	SLTY = DATR
	S5	0x65	0	1	1	-	0	1	0	1	reserved
	S6	0x66	0	1	1	-	0	1	1	0	XL = DATR
	S7	0x67	0	1	1	-	0	1	1	1	XH = DATR
	S8	0x68	0	1	1	-	1	0	0	0	(X) = DATR, X+
	S9	0x69	0	1	1	-	1	0	0	1	(X) = DATR
	SA	0x6A	0	1	1	-	1	0	1	0	MISO = (X)
	SB	0x6B	0	1	1	-	1	0	1	1	MISO = (Y), Y+
	SC	0x6C	0	1	1	-	1	1	0	0	YH = 0 YL = (SLTB) MISO =(Y), Y+
	SD	0x6D	0	1	1	-	1	1	0	1	reserved
	SE	0x6E	0	1	1	-	1	1	1	0	reserved
	SF	0x6F	0	1	1	-	1	1	1	1	reserved
GM [Get Miso]			1	0	d/a	i/o	r	r	r	r	MISO = register(r r r r)
DT [DaTa]			0	0	0	h/l	d	d	d	d	DATR(h/l) =(d d d d)
LD [LoadData]			1	1	d/a	i/o	r	r	r	r	register(r r r r) = DATR

The abbreviation used in Table.2 for the individual bits in SPI-Ring command is as follows,

(d/a)	: digital/analogue, 0 digital, 1 analogue.
(i/o)	: input/output, 0 input, 1 output.
(h/l)	: high/low, 0 high, 1 low.
(d d d d)	: data bits (0 – 15 decimal value).
(r r r r)	: register index (0 – 15 decimal value).
(-)	: don’t care bits. The position is not check and filled with 0.

As can be observed from Table.2, each command is given a symbolic name. The *Opcode* is the hexadecimal reading of the command byte. Although it is theoretically possible to encode 256 different variations, only four main commands are defined by using most significant bits (C7 C6). Data (d) and register (r) bits are the parts of related commands.

Each bit in the SPI-Ring command has a definite meaning. Most significant bits C7 and C6 determine main commands. There are four main commands defined in the *SPI-Ring Protocol*. They are SBC (S0-SF), GM, DT and LD commands. The explanations is as follows,

- SBC (S0 – SF) (Sub-Commands) are operative commands. Bits (C7 C6) are set to (0 1) and bit C5 is set to (1). Individual sub command is determined by least significant four bits of the command (C3 C2 C1 C0). There are total of $2^4=16$ sub commands. They are named through S0 – SF. The sub commands have fixed *Opcode*s. Operation dictated by these commands on the slave device is given in the last row of the Table.2.

- GM (Get-Miso) command in which bits (C7 C6) are set to (1 0) gives instruction to slave device which register's content should be returned over MISO register for the next SPI cycle. Slave device assign the asked register value to the MISO register. Hereby low nibble of the command specify register number. Bit (i/o) indicates if the register to be returned is of type input or output (input if (i/o) bit is equal to 0, output otherwise). Bit (d/a) determines that if the register is of type digital or analogue. If, for example, digital input register IR00 of the slave device is to be returned to the master device for the next SPI cycle, master device should issue the command

$$(1000\ 0000)_2 = 0x80.$$

If the low byte of the analog input register AI01, (register index equal to 2) should be returned then the corresponding command appears as

$$(1010\ 0010)_2 = 0xA2.$$

- DT (DaTa) command transmits 4-bit information (data) from master device to slave device. Bits (C7 C6) are set to (0 0). The data is going to be assigned to the high or low nibble of the data register (DATR) in slave device depending upon the value of the (h/l) bit of the command. Data is carried by the low nibbles of the command (C3 C2 C1 C0). If for example high nibble of the data register DATR is to be replaced by the bit muster 1111 then the master device should issue the command

$$(0000\ 1111)_2 = 0x0F$$

If the low nibble of the data register should be replaced by the bit muster then the corresponding command looks like

$$(0001\ 1111)_2 = 0x1F.$$

- LD (Load-Data) command instructs the slave device to transport content of the data register DATR to the register whose register index is specified in the commands low nibble. Bits (C7 C6) are set to (1 1). For example, if the value of DATR register is to be assigned to the digital output register OR03, the master device should issue the command

$$(1101\ 0011)_2 = 0xD3.$$

If, for instance, IR01 is replaced with the DATR, the master should issue the command

$$(1100\ 0001)_2 = 0xC1.$$

Implementation of SPI-Ring Protocol in Assembler Language

Because MD100-PLC device uses AVR-Mega Series of micro controller as the central processing unit, the *Virtual PLC-Machine* building the PLC system on the micro controller is totally developed in AVR- Assembler language [6]. The *SPI-Ring Protocol* driver for master side devices are parts of the *Virtual PLC-Machine* and implemented likewise in AVR- Assembler language. Likewise the permanent firmware system program running on the slave device implements the slave side driver for the *SPI-Ring Protocol.*

Three cascaded assembler routines are developed for the master side driver implementation of the *SPI-Ring Protocol.* These routines are *SpiTranmit*, *TrackSlave* and *SpiMaster*. For the slave side driver implementation, an interrupt service routine, *OnSpiRcv*, is developed. All these routines are explained in the following sections in detail.

SpiTransmit

SpiTransmit is the low level kern routine which sends *MOSI* register by using the SPI-Interface. The initialization of the SPI-Interface is done automatically as the system is started. MOSI register is passed to the *SpiTransmit* routine as an argument. The incoming byte is stored in the MISO register. The host function (*TrackSlave*) which calls the *SpiTransmit*, must update MOSI register before calling the *SpiTransmit* routine.

TrackSlave

To put the transmission in a more compact form, *TrackSlave* routine is designed. It is the host function calling the *SpiTransmit*. The objective is that, once,

i. the value to be transmitted,
ii. the slave target register to which the transmitted value to be assigned
iii. and the slave register whose content is to be returned are specified

The routine should manage the rest. Three system variables are defined for this purpose. They are namely VALU, TARE and RTVA registers. These registers are passed to the *TrackSlave* routine as static arguments. The upper level function (*SpiMaster*) which calls *TrackSlave*, must update these registers before calling the *TrackSlave* routine.

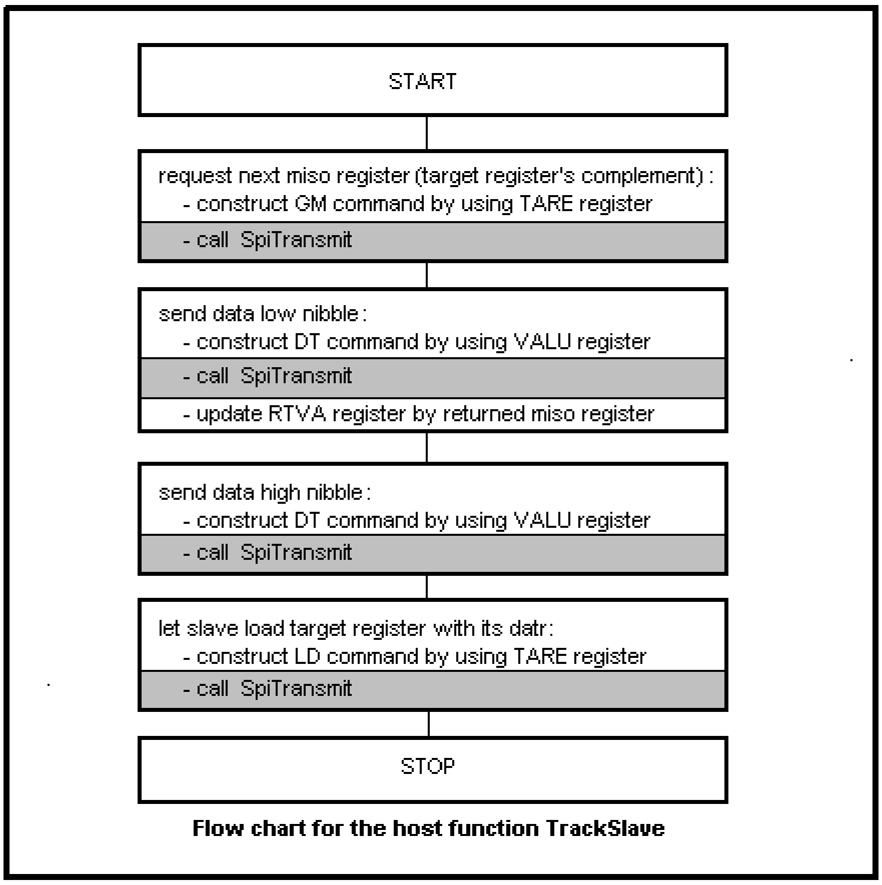

START
request next miso register (target register's complement) :
- construct GM command by using TARE register
- call SpiTransmit
send data low nibble :
- construct DT command by using VALU register
- call SpiTransmit
- update RTVA register by returned miso register
send data high nibble :
- construct DT command by using VALU register
- call SpiTransmit
let slave load target register with its datr:
- construct LD command by using TARE register
- call SpiTransmit
STOP
Flow chart for the host function TrackSlave

Fig.3: Flowchart for the function TrackSlave

VALU register keeps the value which is going to be transmitted to the slave device. TARE register keeps (d/a) and (i/o) information bits together with the register index of the target slave register to which the transmitted value should be assigned by the slave device. *TracksSlave* function does not allow input registers as target register simply because input registers should only be updated by corresponding input channels in both devices. Therefore it sets (i/o) bit of the TARE register always high (=1). Individual bits of the TARE register are seen as follows,

- - (d/a) (i/o=1) r r r r

where (d/a) indicates if target register is of type digital or analogue, (i/o) indicates if target register is of type input or output and (r r r r) indicates the target register index.

RTVA register is updated by the content of the target register's complement. Because the target register is always the output register, its complement is defined as the input register in the slave device whose register number (r r r r) and (d/a) bits are defined in TARE.

In brief, *TrackSlave* function is called with current VALU and TARE registers in the *SYSTEM-SCAN* by the master device. The target register in the slave device is to be updated with the content of the VALU register after a successful call to the *TrackSlave* function. At the same time, the content of the target register's complement is returned from the slave device and stored into the RTVA register in the master device.

The flow chart of the host function *TrackSlave* is given in Fig.3. As seen from the flow chart, the host function calls four times the low level transmission function *SpiTranmit* in order to establish a single task in which the target register in the slave device is updated and its complement is simultaneously drawn back to the master device.

SpiMaster

SpiMaster is the upper level function calling the host function *TrackSlave*. *SpiMaster* updates VALU and TARE registers and past them together with RTVA registers to the *TrackSlave* function. *SpiMaster* calls *TrackSlave* six times in order to update output channels in the slave device and retrieves input channels from the slave device.

Table.3 lists the tasks performed by the upper level function *SpiMaster*. Each task is a single call to the *TrackSlave* function. Contents of passed arguments and returned value from the slave device are indicated for each task.

Superscript *m* indicates that the variables belong to master device. Similarly superscript *s* shows that the variable belongs to slave device. As can be observed from the Table.3, task-1 and task-2 do operation on digital variables. Remaining tasks do operation on analog variables.

Table.3: Tasks performed by the SpiMaster function

SpiMaster Function				
Task: TrackSlave-Call	**Arguments for TrackSlave function**			**Operation after TrackSlave-Call**
Task Number	**VALU**	**TARE**	**RTVA**	**Update target register's complement**
1	$[OR02]^m$	$[OR00]^s$	$[IR00]^s$	$[IR02]^m$ =RTVAL
2	$[OR03]^m$	$[OR01]^s$	$[IR01]^s$	$(IR03)^m$ =RTVAL
3	$[(AO02)low]^m$	$[(AO00)low]^s$	$[(AI00)low]^s$	$[(AI02)low]^m$ = RTVAL
4	$[(AO02)high]^m$	$[(AO00)high]^s$	$[(AI00)high]^s$	$[(AI02)high]^m$ = RTVAL
5	$[(AO03)low]^m$	$[(AO01)low]^s$	$[(AI01)low]^s$	$[(AI03)low]^m$ = RTVAL
6	$[(AO03)high]^m$	$[(AO01)high]^s$	$[(AI01)high]^s$	$[(AI03)high]^m$ = RTVAL

OnSpiRcv

OnSpiRcv subroutine is implemented on slave device. It is an interrupt service routine (ISR). It is called each time when a one-byte long data is received over SPI-Interface. The received data is put into the MOSI register. It is then checked for valid *SPI-Ring Protocol* command. If it is a valid command then it is executed. The flow chart of the subroutine *OnSpiRcv* is shown in Fig.4.

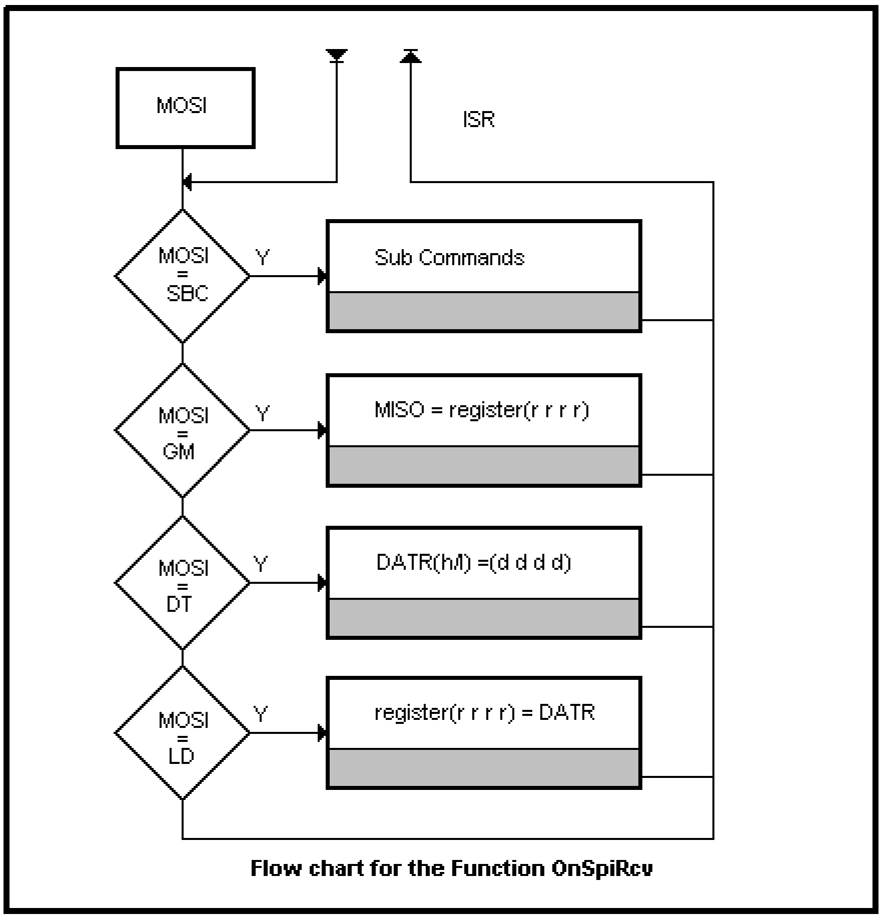


Fig.4: Flow Chart for the Interrupt Service Routine OnSpiRcv

TESTING THE SPI-Ring PROTOCOL

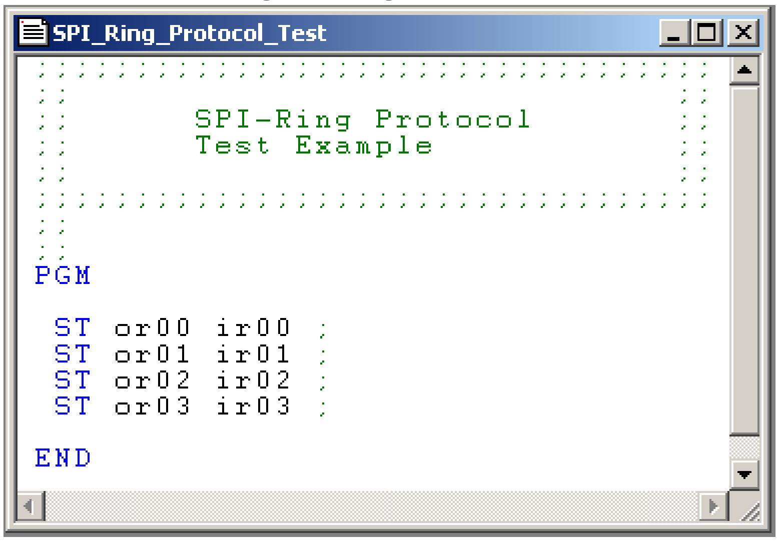
In order to test the performance of the *SPI-Ring Protocol*, two test cases are developed. In test case-1, a simple instruction-list PLC program, *InToOut*, assigning the input channels to the output channels is written. In test case-2, a block-diagram PLC program, *flashor,* simulating some output channels toggling at a specific frequency is developed. Both programs visualize the transmitted data and indicate the success of the *SPI-Ring Protocol*

Test case-1

In test case-1, the test program *InToOut* assigns the digital input channels to the corresponding digital output channels. That is,

- IR00 is assigned to OR00 (OR00 ← IR00).
- IR01 is assigned to OR01 (OR01 ← IR01).
- IR02 is assigned to OR02 (OR02 ← IR02).
- IR03 is assigned to OR03 (OR03 ← IR03).

A snapshot from the KUMANDA showing the instruction list of the example program *InToOut* is given in Fig.5.


SPI_Ring_Protocol_Test

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                              ;;
;;       SPI-Ring Protocol      ;;
;;       Test Example           ;;
;;                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;
PGM

 ST or00 ir00 ;
 ST or01 ir01 ;
 ST or02 ir02 ;
 ST or03 ir03 ;

END
```

Fig.5: SPI-Ring Protocol Test Example-1

Channels on the master device represented by master variables IR00, IR01, OR00 and OR01 lie physically in the master device. They are therefore not relevant in testing *SPI-Ring Protocol*. Channels on the master device represented by master variables IR02 and IR03 must be updated from slave channels represented by slave variables IR00 and IR01. That means slave input channels are read into the slave input variables IR00 and IR01 by the slave device. They are then passed through the *SPI-Ring Protocol* and assigned to master input variables IR02 andIR03 respectively.

Similarly output variables IR02 and IR03 on the master device are updated in the PLC program and passed to slave output variables OR00 and OR01 through the *SPI-Ring Protocol*. Then the slave device must update slave output channels from slave output variables respectively. Therefore channels represented by the master variables IR02, IR03, OR02 and OR03 are critical in Testing SPI-Ring Protocol simply because these channels lie on the slave device.

When any of the input channels is set to high, the corresponding output channel is set to high, or vice versa by the program. The states of channels are observable from channel's led built on the MD100-PLC as the program runs. Input channels have green led and output channels have red led.

The photo illustrating the results of the test program *InToOut* is indicated in Fig.6. Variation of the input channel's states and corresponding output channel's states are illustrated through the sub sections 1 to 4 in Fig.6. Because input channels are directly assigned to output channels, the states of the output channels must be equal to the states of the corresponding input channels. In variation 1, IR02 is set arbitrarily to the binary values $(100000010)_2$ and IR03 is set arbitrarily to the binary values $(00000010)_2$ by applying 12 volt input voltage to the respective input channel's pins. It is observed that output channels have the same state as the input channels states.

In variation 2, 3 and 4 input channel's states are changed. Corresponding state changes in output channels are observed. All output channels shoved the same states as the input channels as expected.

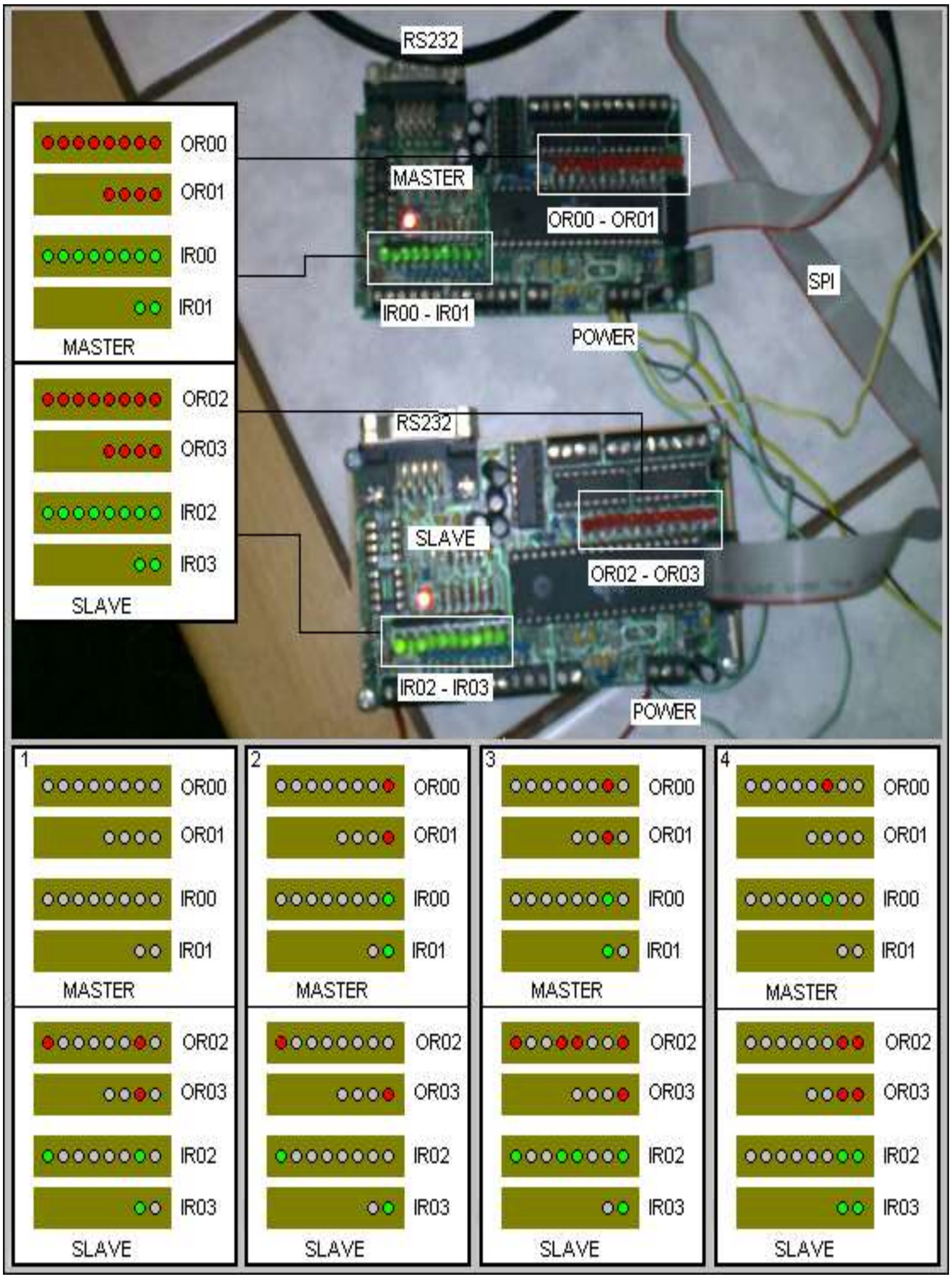
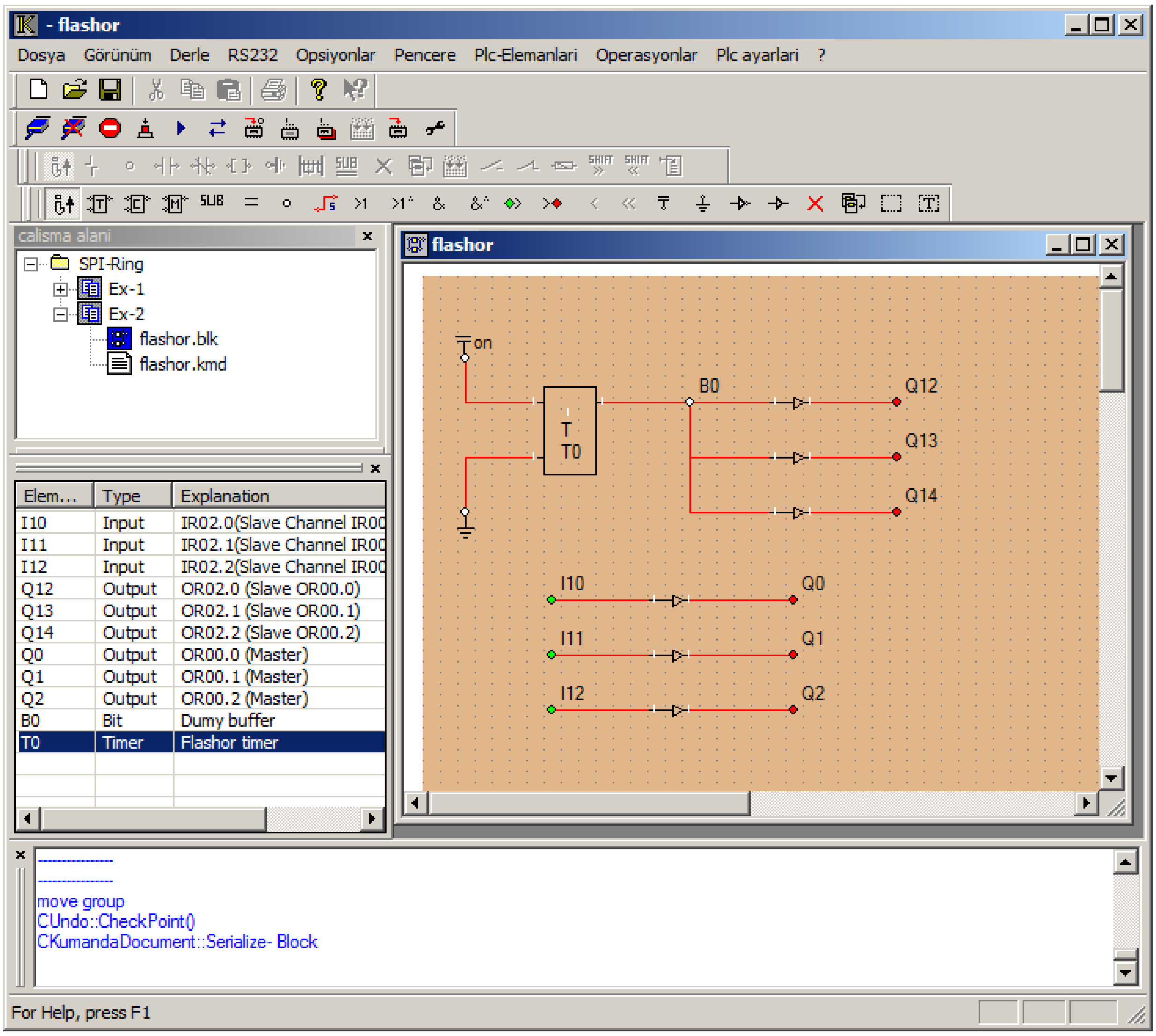


Fig.6: Result of the test example InToOut running on the Modular MD100-PLC system

Test Case -2

In test case-2, the block-shalt PLC program, *flashor*, is developed. The snapshot from the KUMANDA showing the block-shalt program, *flashor* is indicated in Fig.7.



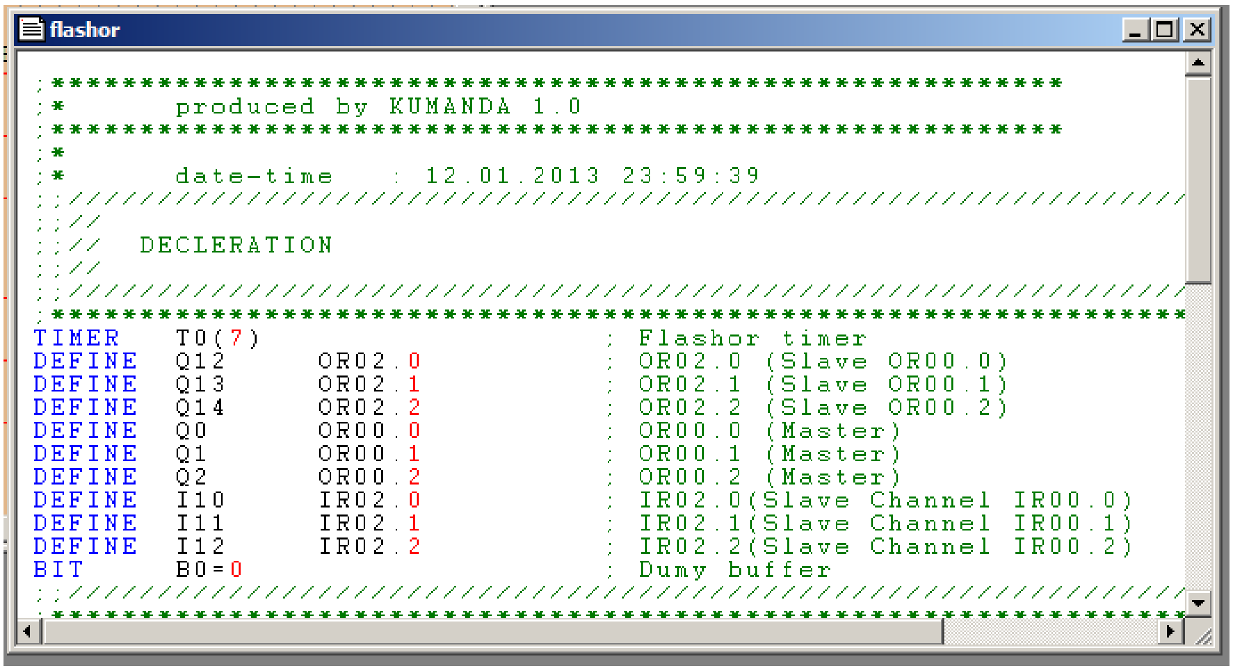
Elem...	Type	Explanation
I10	Input	IR02.0(Slave Channel IR00
I11	Input	IR02.1(Slave Channel IR00
I12	Input	IR02.2(Slave Channel IR00
Q12	Output	OR02.0 (Slave OR00.0)
Q13	Output	OR02.1 (Slave OR00.1)
Q14	Output	OR02.2 (Slave OR00.2)
Q0	Output	OR00.0 (Master)
Q1	Output	OR00.1 (Master)
Q2	Output	OR00.2 (Master)
B0	Bit	Dumy buffer
T0	Timer	Flashor timer

```
---------------
---------------
move group
CUndo::CheckPoint()
CKumandaDocument::Serialize- Block
```

Fig.7: Individual channels assignments by the KUMANDA for the program flashor
SPI-Ring Protocol Test Example

After the compilation of the *flashor*, KUMANDA produce the instruction list of the block-shalt program automatically. The snapshot showing the definition segment of the automatically produced instruction list for the block-shalt program *flashor* is given in Fig.8.

In the *flashor*, flashing timer T0 set the internal buffer B0 on and off for the period of two seconds continuously (one second on / one second off). State of the buffer B0 is transmitted to the slave channels Q12, Q13 and Q14 over the *SPI-Ring Protocol* at the end of the each PLC-SCAN.



```
;*******************************************************
;*       produced by KUMANDA 1.0
;*******************************************************
;*
;*       date-time    : 12.01.2013 23:59:39
;;///////////////////////////////////////////////////////////////////////
;;//
;;//   DECLERATION
;;//
;;///////////////////////////////////////////////////////////////////////
;*******************************************************************
TIMER    T0(7)                     ; Flashor timer
DEFINE   Q12      OR02.0           ; OR02.0 (Slave OR00.0)
DEFINE   Q13      OR02.1           ; OR02.1 (Slave OR00.1)
DEFINE   Q14      OR02.2           ; OR02.2 (Slave OR00.2)
DEFINE   Q0       OR00.0           ; OR00.0 (Master)
DEFINE   Q1       OR00.1           ; OR00.1 (Master)
DEFINE   Q2       OR00.2           ; OR00.2 (Master)
DEFINE   I10      IR02.0           ; IR02.0(Slave Channel IR00.0)
DEFINE   I11      IR02.1           ; IR02.1(Slave Channel IR00.1)
DEFINE   I12      IR02.2           ; IR02.2(Slave Channel IR00.2)
BIT      B0=0                      ; Dumy buffer
;;///////////////////////////////////////////////////////////////////////
;*******************************************************************
```

Fig.8: Individual channels assignments by the KUMANDA for the program flashor

In order to control the data transmission in the direction of slave-to-master, slave input channels I10, I11and I12 are connected to the master output channels Q0, Q1 and Q2. The values of the input channels on the slave device should be transmitted back to the master output channels over the *SPI-Ring protocol* at the end of the each PLC-SCAN.

When the program is let run, it is observed that slave output channels Q12, Q13 and Q14 are flashing with a frequency of 1/2 hertz. Similarly input values on the slave input channels I10, I11 and I12 are seen at the master output channels Q0, Q1 and Q2 respectively. This proofs the true data transmission in both directions over the *SPI-Ring Protocol.*

DISCUSSION

The Time interval for the slave device in successive *SpiTransmit* function calls is 30 microseconds at the tact frequency of 8 MHz. Because *SpiTransmit* is called four 4 times in a single transaction (*TrackSlave*) and there is a total of 6 transactions in the *SpiMaster* function, the total amount of the delay for the master is equal to 30x4x6=720 microseconds. Increasing the tact frequency of the micro controller to 32 MHz shortens this delay down to 180 microseconds. In most cases analog channels are redundant in the slave device. When Analog channels are inactivated then only 2 transactions stay in the *SpiMaster* function. This reduced the total delay to 60 microseconds.

All of these calculations are based upon pre-declared SPI-Interface frequency (SCLK) of 250 kHz (SCLK is set to f/32 where f denotes µC's tack frequency and set to 8 MHz for the test MD100-PLC board). It is quite possible to increase SCLK frequency to f/8. In such cases, SCLK frequency increases to 1 MHz which in turn brings the total delay to 15 microseconds.

This is the time interval by which the PLC-cycle is delayed in modular MD100- PLC system as compared to single MD100-PLC system. 15 microseconds delay in PLC-Scan time is not always critical for a plenty of applications in industry. Such amount of delay can surely be tolerated for gaining doubled I/O lines.

CONCLUSION

The developed *SPI-Ring Protocol* provides a flexible and economic data transmission between master and slave side devices. Although it is specifically developed for the MD100 Programmable Logic Controllers, it can surely be adopted for custom controllers performing a specific job and need communication among each other. The protocol is extremely tested. It is quick enough for most of industrial applications. Reliable and satisfactory results are obtained. Many devices employing the *SPI-Ring Protocol* are already installed in industry. They proved success in all related aspects.

PROPOSEL FOR FUTURE WORK

The followings studies are proposed for future work.

- Expansion of *SPI-Ring Protocol* for multi-slave system. This can be realized by extending the master side of implementation of system functions *SpiMaster* and *TrackSlave*. The existing routines can easily be passed for multi slave system. For this purpose, the MD100-PLC must slightly be modified to incorporate additional */SS* lines.
- Enlarging *SPI-Ring Protocol* command to two bytes. This provides more application specific commands to be categorized. This of course necessities reimplementation of master and slave side transmission functions.
- A different physical interface. Instead of SPI interface in the physical layer, other transmission channels such as RS232 or I2C can be implemented.

SYMBOLS & ABBREVIATIONS

PLC	: Programmable Logic Controller
µC	: Micro Controller
MD100-PLC	: MD100 Programmable Logic Controller
AVR-MEGA	: µC series from producer ATMEL
AVR-MAGA32	: µC from producer ATMEL
SPI	: Serial Peripheral Interface
RS232	: Recommended Standards for serial Transmission 232
Virtual PLC-Machine	: Firmware, building PLC system on micro controller
IO	: Input / Output
SCLK	: Serial Clock in SPI-Interface
MOSI	: Master Out Slave In Register
MISO	: Master In Slave Out Register
/SS	: Slave Select in SPI-Interface
INPUT-SCAN	: Input Channel’s Scan
PROGRA-SWEEP	: Time Interval to run application program once
OUTPUT-SCAN	: Output Channel’s Update
SYSTEM-SCAN	: System Interface
SPI-Ring Protocol	: Data Exchange Protocol over SPI-Interface
OR00	: Digital Output System Register-00 [Output Channels 0 to 7]
OR01	: Digital Output System Register-01 [Output Channels 8 to 15]
OR02	: Digital Output System Register-02 [Output Channels 16 to 23]
OR03	: Digital Output System Register-03 [Output Channels 24 to 31]
IR00	: Digital Input System Register-00 [Input Channels 0 to 7]
IR01	: Digital Input System Register-01 [Input Channels 8 to 15]
IR02	: Digital Input System Register-02 [Input Channels 16 to 23]
IR03	: Digital Input System Register-03 [Input Channels 24 to 31]
AI00	: Analog Input System Register-00[Analog input channel 0]
AI01	: Analog Input System Register-01[Analog input channel 1]
AI02	: Analog Input System Register-02[Analog input channel 2]
AI03	: Analog Input System Register-03[Analog input channel 3]
AO00	: Analog Output System Register-00[Analog output channel 0]
AO01	: Analog Output System Register-01[Analog output channel 1]

AO02	: Analog Output System Register-02[Analog output channel 2]
AO03	: Analog Output System Register-03[Analog output channel 3]
NOP	: No-Operation Command
XL	: Low nibble of X Register in AVR-MEGA µC architects
XH	: High nibble of X Register in AVR-MEGA µC architects
YL	: Low nibble of Y Register in AVR-MEGA µC architects
YH	: High nibble of Y Register in AVR-MEGA µC architects
X	: X Register in AVR-MEGA µC architects
Y	: Y Register in AVR-MEGA µC architects
DATR	: Data Register in MD100-Register file
VALU	: Value Register in MD100-Register file
TARE	: Target Register in MD100-Register file
RTVA	: Return-Value Register in MD100-Register file
S0	: Sub-Command 0 of SPI-Ring Protocol Command SBC
S1	: Sub-Command 1 of SPI-Ring Protocol Command SBC
S2	: Sub-Command 2 of SPI-Ring Protocol Command SBC
S3	: Sub-Command 3 of SPI-Ring Protocol Command SBC
S4	: Sub-Command 4 of SPI-Ring Protocol Command SBC
S5	: Sub-Command 5 of SPI-Ring Protocol Command SBC
S6	: Sub-Command 6 of SPI-Ring Protocol Command SBC
S7	: Sub-Command 7 of SPI-Ring Protocol Command SBC
S8	: Sub-Command 8 of SPI-Ring Protocol Command SBC
S9	: Sub-Command 9 of SPI-Ring Protocol Command SBC
SA	: Sub-Command A of SPI-Ring Protocol Command SBC
SB	: Sub-Command B of SPI-Ring Protocol Command SBC
SC	: Sub-Command C of SPI-Ring Protocol Command SBC
SD	: Sub-Command D of SPI-Ring Protocol Command SBC
SE	: Sub-Command E of SPI-Ring Protocol Command SBC
SF	: Sub-Command F of SPI-Ring Protocol Command SBC
GM	: SPI-Ring Protocol Command Get-Miso
DT	: SPI-Ring Protocol Command DaTa
LD	: SPI-Ring Protocol Command Load-Data
d/a	: digital/analog, 0 digital, 1 analog.
i/o	: input/output, 0 input, 1 output.
h/l	: high/low, 0 high, 1 low.
(d d d d)	: data bits, in *MOSI* byte (0 – 15 decimal value).
(r r r r)	: register index in *MOSI* byte (0 – 15 decimal value).
(-)	: don’t care bit, position not checked and filled with 0.
Opcode	: Operation Code for SPI-Ring Protocol command
C0	: Bit-0 (Least Significant Bit)
C1	: Bit-1
C2	: Bit-2
C3	: Bit-3
C4	: Bit-4
C5	: Bit-5
C6	: Bit-6
C7	: Bit-7 (Most Significant Bit)
KUMANDA	: Developing Environment for the MD100-PLC on PC.
f	: Tact frequency of the µC

REFERENCES

1. Internet: KUMANDA, "Integrated Program Development Environment for MD100-PLC Devices", Online. http://www.plcturk.com/webserver/kumanda/IDELadder.gif, 2019.
2. Internet: Virtual Machine, "Firmware building PLC System on Avr Microcontroller", Online. http://www.plcturk.com/webserver/kumanda/VirtualMachine-01.htm, 2019.
3. Internet: AVR Working-Register, "8-bit Avr Microcontroller", Online. http://academy.cba.mit.edu/classes/embedded_programming/doc7766.pdf, 2019.
4. Internet: AVR-MEGA16, "8-bit Avr Microcontroller with 16 KBytes Flash Memory", Online. http://www.atmel.com/Images/doc2466.pdf, 2019.
5. Internet: AVR-MEGA32, "8-bit Avr Microcontroller with 32 KBytes Flash Memory", Online. http://www.atmel.com/Images/doc2503.pdf, 2013.
6. Internet: AVR-Assembler language, "Avr Assembler User Guide" http://www.atmel.com/Images/doc1022.pdf, 2013.