RESEARCHERID
THOMSON REUTERS

[Dülger * 6(5): May, 2019] ISSN 2349-4506
Impact Factor: 3.799

Global Journal of Engineering Science and Research Management

DETEL SERIAL DATA TRANSMISSION PROTOCOL FOR THE COMMUNICATION OF AN IBM-PC WITH THE MDxx SERIES OF PROGRAMMABLE LOGIC CONTROLLERS

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

DOI: 10.5281/zenodo.3235361

KEYWORDS: Transmission Protocol, PC, Telegram, DETEL, ISR, Slave, Master.

ABSTRACT

In this paper a new serial transmission protocol, *DETEL*, is designed and presented. *DETEL* protocol is developed to provide a reliable communication between an *IBM-PC* and a *Programmable Logic Controller*, *PLC*, of type *MDxx* series. Here the *IBM-PC* acts as a master (programmer) device and the *PLC* acts as a slave device. The integrity of the data to be transferred is discussed. OSI model of the *DETEL* protocol is presented. Link-layer and application-layer protocols are implemented. *DETEL* Protocol is implemented on both master and server side devices. A software driver is developed for the master device in C++. A micro-controller board, on which a virtual *PLC* machine of type *MDxx* series is installed, is used as a slave device. A driver for the slave device is developed in *avr-assembler* language.

INTRODUCTION

The serial data transmission between two communicating electronic devices needs a certain data transmission protocol so that both agents of the transmission clearly understand each other. A typical master-slave system connected by a serial cable (i.e. *RS232*) is shown in Fig.1. The master device is an *IBM-PC* and the slave device is a *Programmable Logic Controller*, *PLC*, of type *MDxx* series [1]. The master sends a telegram to the slave. The slave should interpret the received telegram and operate accordingly.

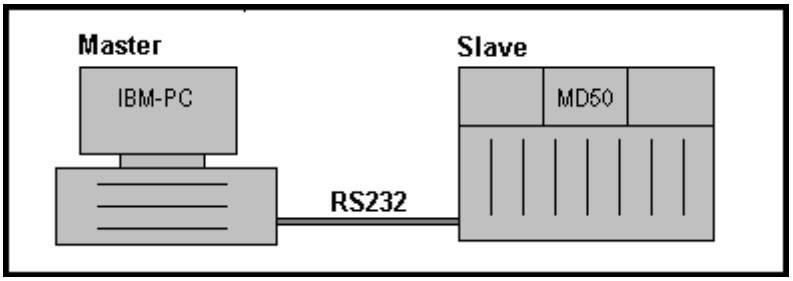

Fig.1: Master Slave Type System

OSI (*Open Systems Interconnection*) is a reference model for how network agents communicate [2]. Seven abstract layers are defined in *OSI* reference model. The first layer in *OSI* model, physical-layer, defines physical medium by which the bit stream is transmitted. The second layer in *OSI* model, data-link-layer, defines the protocol according to which data telegram is encoded/decoded into/from byte stream. The last layer in *OSI* model, application-layer, defines the protocol how the application data should be interpreted. It is then assured that both master and slave devices interpret the application data uniquely.

The minimum requirement for the communication is that first, second and last layers in OSI model must be implemented. In this work, a new serial data transmission protocol, *DETEL*, is designed obeying *OSI* reference model.

ISSN 2349-4506
Impact Factor: 3.799
Global Journal of Engineering Science and Research Management

DETEL SERIAL DATA TRANSMISSION PROTOCOL

DETEL is a well-defined data transmission protocol to transmit data in a master-slave system. It is developed primarily to achieve data transmission between the host *PC* and *MDxx* series of *PLCs*. *OSI* Model of the *DETEL* protocol in comparison with that of Modbus is given in Fig.2.

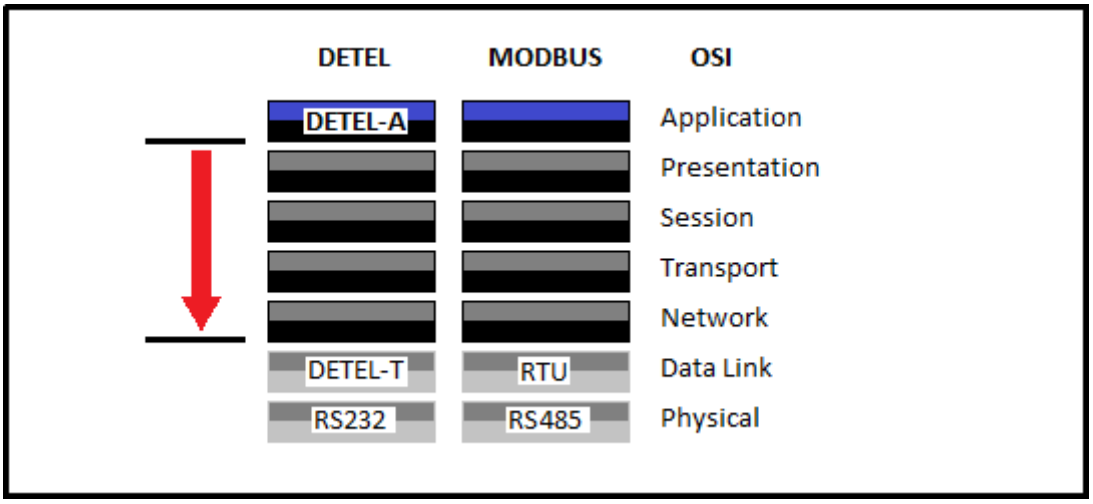

Fig.2: OSI Reference Model of the DETEL Serial Data Transmission Protocol

As it is clear from Fig.2, physical layer of the *DETEL* is implemented by using *RS232* serial connection. It can be further extended to other hardware connections (RS485 for instance).

DETEL-T telegram is developed to implement data link layer of the *DETEL* transmission protocol in OSI reference model. It is analogous to the RTU and/or ASCII transmission in the OSI reference model of the corresponding *Modbus* protocol. *DETEL-A* protocol implements the application layer of the *DETEL* transmission protocol. DETEL-A is developed for the programming software *KUMANDA* which runs on the *PC* in order to program *MDxx* series of *PLCs* [3].

DETEL-T SERIAL DATA TRANSMISSION TELEGRAM

Fig.3 gives the schematic of the *DETEL-T* serial data transmission telegram. As seen from Fig.3, the telegram is made up of four segments. They are start segment, control segment, data segment and end segment.

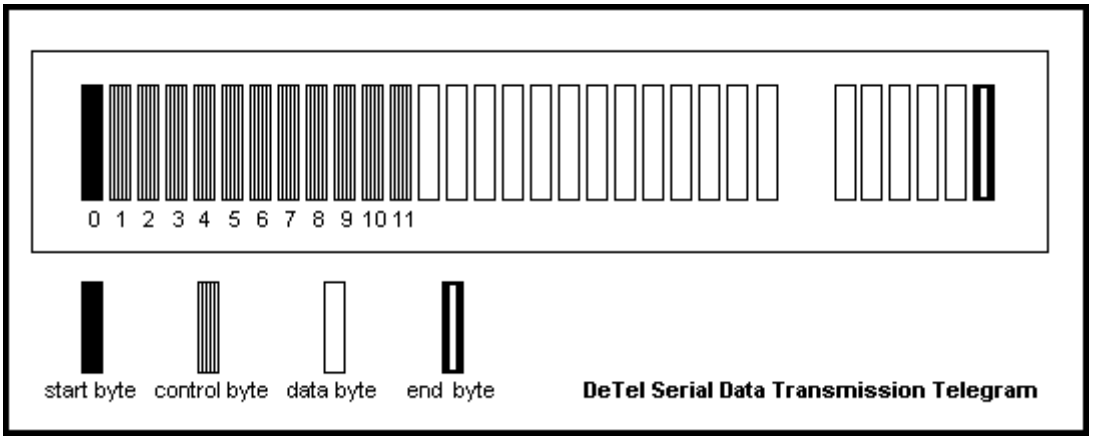

Fig.3: Schematic of the DETEL-T Serial Data Transmission Telegram

The start segment contains the *Start-Byte* signalising the start of the telegram. It is indexed by 0 in the telegram. Its value is fixed and equal to 0xFD.

Control segment is located next to the *Start-Byte* and contains control bytes. There are fixed number of control bytes (11 control bytes). They are indexed through digits 1 to 11 (*CTRL-1, CTRL-2. …., CTRL-11*). Last control byte, *CTRL-11* keeps the number of data bytes in data segment. The rest of the control bytes are application specific and left as unimplemented.

Next to the control segment is the data segment. It contains data bytes. The data segment stores row information to be passed. Maximum of 255 (0xFF) data bytes can be allocated in this segment. The length of data segment is variable and denoted by *CNT*. *CNT* is always given by the control byte *CTRL-11* in the telegram.

The last segment is the end segment and contains only the *End-Byte*. The *End-Byte* signalises the end of the telegram.

Minimum telegram size occurs when the length of the data segment is null. In this case, there are *Start-Byte*, eleven control bytes and *End-Byte*. They sum altogether to 13 bytes. Maximum telegram size is 268 and it occurs when the length of the data segment is 255. In this case the data segment contains 255 data bytes.

Detailed description of each byte in the *DETEL-T* telegram is given in table Table.1.

Table.1: Illustration of DETEL-T Telegram

Index	segment	member	Explanation
0	start	XON	start of the telegram
1	control	CTRL-1	
2	control	CTRL-2	
3	control	CTRL-3	
4	control	CTRL-4	
5	control	CTRL-5	
6	control	CTRL-6	
7		CTRL-7	.
8	control	CTRL-8	
9	control	CTRL-9	
10	control	CTRL-10	
11	control	CTRL-11	CTRL-11 = CNT. CNT is the number of data bytes in data segment
12	data	DAT0	Data (Byte 0)
13	data	DAT1	Data (Byte 2)
14	..	DAT2	Data (Byte 3)
..	..	..	..
..	..	..	..
..	..	..	..
CNT+10	data	DAT[CNT-2]	Data (Byte CNT-2)
CNT+11	data	DAT[CNT-1]	Data (Byte CNT-1)
	crc	CRC-L	Cyclic Redundancy Check. Not yet implemented
	crc	CRC-H	Cyclic Redundancy Check. Not yet implemented
CNT+12	end	XOF	end of the telegram

DETEL-T TELEGRAM INTEGRITY

Because *DETEL-T* telegram is a well-structured telegram, following requirements must be fulfilled in the implementation of the telegram.

1. *Start-Byte* and *End-Byte* must have unique values
   While the stream of bytes is being received by a slave device, the start and end of the telegram in the stream must be recognized by the slave device. It is made possible by defining unique bytes for start and end of the telegram. *Start-Byte*, *XON*, is equal to 0xFD and *End-Byte*, *XOF*, is equal to 0xFE per definition.

2. Control segment must be fixed in size.
   For the sake of the integrity of the whole telegram all control bytes must exist. This requires that the set of control bytes must appear completely at the beginning of the telegram. Therefore first eleven bytes next to the *Start-Byte* in the telegram are always regarded as control bytes by the slave device.

3. *Start-Byte* and *End-Byte* must not be repeated in telegram
   If the *Start-Byte* is repeated in data or control segment, the integrity of the telegram is broken down. In this case the telegram start is shifted to the position where the *Start-Byte* is repeated. From here on following eleven bytes are regarded as control bytes. This in turn leads to wrong telegram construction. Therefore the *Start-Byte* must neither in control nor in data segment be repeated.

   Similarly if the *End-Byte* is repeated in control or data segment, the integrity of the telegram is again broken down. In this case early end of the telegram occurs causing the rest of the bytes to fail. Occurrence of *End-Byte* in control or data segment must hence be avoided.

4. Transmission of *DETEL-T* Telegram should obey the half-mode transmission.
   One can raise the question how a telegram can be constructed in them a data byte has a value equal to either *Start-Byte* or *End-Byte*. To solve this problem a telegram transmission scheme, referred to as *half-mode transmission*, is developed. In this mode of telegram transmission, occurrence of *Start-Byte* or *End-Byte* in data segment does not break down the integrity of the telegram.

These requirements are taken into account in the design of *DETEL-T* telegram. Details of half mode telegram transmission are going to be discussed in the next section.

HALF MODE TRANSMISSION OF DETEL-T TELEGRAM (MASTER SIDE)

DETEL-T telegram is transmitted in two different modes. They are full-mode and half-mode. The full mode implementation is out of the scope this article. The half-mode implementation will be discussed in this section.

In the half-mode, master-device has a *FIFO* (*First In First Out*) transmission *Queue*. Fig.4. shows schematic representation how *DETEL-T* telegram is allocated in the transmission *Queue* of the master device just before the transmission. The first byte entering the *Queue* will be transmitted first.

Master-device does not allocate the *DETEL-T* telegram byte by byte in the transmissions *Queue*. Instead, the allocation of *DETEL-T* telegram in the *Queue* is performed as follows;

Start-Byte, *XON*, signalising the start of the telegram is first allocated in the transmission *Queue* and hence transmitted first. Following the *Start-Byte*, control bytes are allocated in the transmission *Queue* and then they are transmitted in the order they are put in the *Queue*. Data bytes, on contrary to start and control bytes, are not directly allocated in the transmission queue. Each data byte is first expanded into two bytes and resulting two bytes are then allocated into the transmission *Queue*. The resulting bytes keep low and high nibbles of the original data byte respectively.

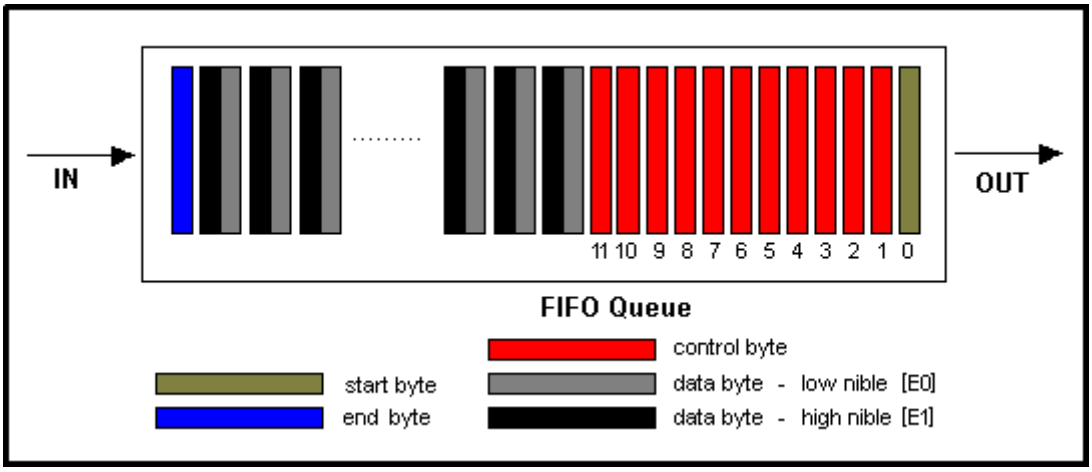

Fig.4: Allocation of DETEL Telegram in the Transmission Queue of the Master Device

For example the original data byte D in the binary form

D: d7 d6 d5 d4 d3 d2 d1 d0

is expanded to following bytes

E0: **0 0 0 0 d3 d2 d1 d0**

E1: **d7 d6 d5 d4 0 0 0 0**

E0 contains low nibble of the original data byte D. E1 contains high nibble of the original data byte D respectively. E0 is first allocated in the transmission *Queue* and transmitted first. E1 is then allocated in the transmission *Queue* and transmitted next. That means low nibble byte of the original data byte is transmitted first and high nibble byte of the original data byte is transmitted next.

Low nibble of the original data byte lies in the range of [0x00 – 0x0F]. High nibble of the original data byte lies in the range of [0x00, 0x10 – 0xF0]. *Start-Byte*- and *End-Byte* are defined so that they lie out of above ranges. Therefore *Start-Byte* is defined as 0xFD and *End-Byte* is defined as 0xFE. As per definition both *Start-Byte* and *End-Byte* have high and low nibbles different than zero.

Let us assume original data byte in the data segment is equal to *Start-Byte* in value. In this case low nibble E0 will be equal to 0x0D and high nibble E1 will be equal to 0xF0. Both nibbles are transmitted separately and will not destroy integrity of the telegram because each differs from the *Start-Byte*. Similarly if data byte is equal to *End-Byte* in value, integrity of the telegram is also preserved.

If we put any control byte in the control segment equal to *Start-Byte* or *End-Byte* in value, integrity of the telegram is however broken down because control bytes are allocated in the transmission *Queue* without expansion. It cannot hence be defined any control byte equal to either *Start-Byte* or *End*-Byte in value. Therefore a special attention is to be given to the definition of the control bytes. Especially control bytes keeping address values are to be carefully defined so that they are not equal to *Star*- or *End-Byte* in value. Otherwise the telegram gets corrupted.

End-Byte, *XOF*, indicating the end of the telegram is lastly put in the transmission queue and transmitted at the end. By the transmission of the *End-Byte*, transmission of the entire telegram is completed.

In short, *Start-Byte*, *End-Byte* and control bytes are allocated directly in the transmission *Queue*. Data bytes however are first expanded into two bytes keeping low and high nibbles separately and then put into the transmission *Queue*. With this scheme of allocation in the transmission *Queue*, the integrity of the telegram is granted provided that all control bytes differ from *Star-Byte* and *End-Byte* in value.

HALF MODE RECEPTION OF DETEL-T TELEGRAM (SLAVE SIDE)

The decoder machine (the component of the driver software running on the slave device) is responsible for building *DETEL-T* telegram on the slave device. A Special care must be paid to the reception of the *DETEL-T* telegram by the slave device.

As the slave device receives serial data, the decoder machine runs simultaneously in order to construct the *DETEL-T* telegram in the RAM of the slave device. Each byte received is completely processed by the decoder machine before the next byte appears on the serial port. The logic diagram of the decoder machine is indicated in Fig.5.

The decoder machine filters control and data bytes. The decoder machine is actually a subroutine called by a serial data reception *Interrupt Service Routine (ISR)* [4]. Remember that *ISR*s are called by the system when a certain event occurs. Serial data reception *ISR* is called when a character (data in form of one byte) appears at the serial port of the slave device.

Fig.4 indicates logic diagram of the decoder machine in *ISR*. As it is seen from the logic diagram, the decoder machine defines a memory pointer *X*, a strobe signal *strobe*, temporary byte variable *temp*, and a nibble signal *nible*. When a character is caught by the hardware, it is stored in a serial port receive register, *UDR* [4]. The content of the receive register is transferred to a temporary variable *temp*. *X* pointer is used to point current RAM address to which the received byte in variable *temp* is going to be stored. *X* pointer is set to the telegram base address of the slave device, *iTBAS* as the *XON* control signal is received. It is then increased by one for each time as the control byte or high nibble of the data byte is received.

Strobe signal *strobe* is used to denote the fact that the reception of the *DETEL-T* telegram is in action. It is set to high as *XON* signal is received and set to low as *XOF* signal is received. Nibble signal *nible* is an indicator of whether the currently received data is of high nibble (E1 byte) or not. If *nible* signal is equal to null, the received data is the low nibble of the original data else it is the high nibble of the original data. If the *nible* is equal to one, the original data is constructed from the previous data stored in variable *nkeep* and current data stored in the variable *temp*. So constructed original data is then located to the RAM position pointed by *X* pointer. After allocation of original data, *X* pointer is increased by one pointing to the next position in the RAM.

For the programmers, it is worth here to state that the sequence of logic comparisons in the logic diagram is important and should not be changed.

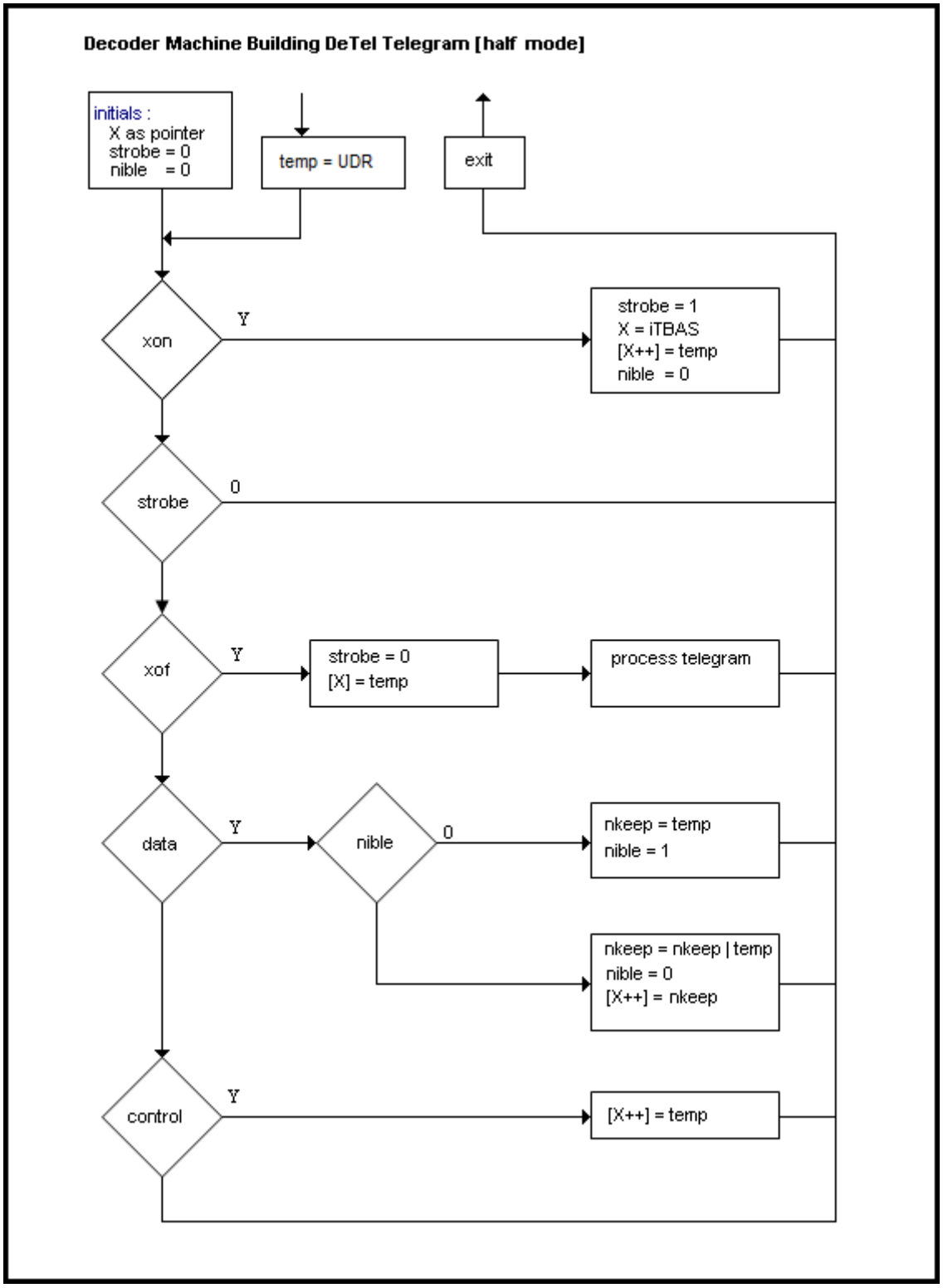

Fig.5: Decoder Machine Building DETEL-T Telegram in Slave Device

ISSN 2349-4506
Impact Factor: 3.799
Global Journal of Engineering Science and Research Management

DETEL-A HIGH LEVEL APPLICATION PROTOCOL

DETAL-A is a high level application protocol. Control bytes are implemented as indicated in Table 1.

Table.1: Implementation of DETEL-A High Level Application Protocol.

Index	segment	member	Explanation
0	Start	XON	START BYTE (0xFD): It indicates the start of the telegram
1	control	CMD0 (CTRL-1)	LOW COMMAND BYTE: It is the low byte of the command. It is implemented for MDxx series of PLCs.
2	control	CMD1 (CTRL-2)	HIGH COMMAND BYTE: It is the high byte of the command. It is reserved to increase number of commands for different applications.
3	control	ADD0 (CTRL-3)	ADRESS BYTE 0: Address byte 0. First address byte. Address bytes are used for the addressing purpose in the slave device. There are total of four bytes for addressing space and are adequate for addressing 4 Giga byte address space.
4	control	ADD1 (CTRL-4)	ADRESS BYTE 1: Address byte 1. Second address byte.
5	control	ADD2 (CTRL-5)	ADRESS BYTE 2: Address byte 2. Third address byte.
6	control	ADD3 (CTRL-6)	ADRESS BYTE 3: Address byte 3. Fourth address byte.
7		NILL (CTRL-7)	ZERO BYTE (0xFC): Zero byte. Nill is the value which should be interpreted as zero.
8	control	RSV0 (CTRL-8)	Reserved : Reserved for future use.
9	control	RSV1 (CTRL-9)	Reserved: Reserved for future use.
10	control	RSV2 (CTRL-10)	Reserved: Reserved for future use.
11	control	CNT (CTRL-11)	DATA COUNT BYTE: Count of data byte. It indicates number of data bytes in the telegram. It cannot exceed 255.
12	data	DAT0	DATA: First byte of data.
13	data	DAT1	DATA: Second byte of data.
..	..	..	..
..	..	..	..
CNT+11	data	DAT[CNT-1]	DATA: Last byte of data.
	crc	CRC-L	Cyclic Redundancy Check. Not yet implemented
	crc	CRC-H	Cyclic Redundancy Check. Not yet implemented
CNT+12	end	XOF	END BYTE (0xFE): It indicates the end of the telegram.

Table.2: DETEL commands

command	cmd1	cmd0	explanation
SP_HALT	00	0x81	enter system into halt mode
SP_WRFL	00	0x82	write data in program memory
SP_RESET	00	0x83	reset
SP_CLEAR	00	0x85	clear pages in user section
SP_ECHO	00	0x86	echo
SP_WREP	00	0x87	write EEPROM

The *DETEL* protocol is integrated into the *Program Developing Environment (KUMANDA)* of the *MDxx* series of PLCs. Here the PC, on which the *KUMANDA* runs, acts as master device and the PLC acts as a slave device. The PLC device has a reduced instruction set controller of ATMEGA16 with multiple IO channels as a centre processing unit [4]. A permanent firmware *Virtual PLC Machine*, *VPM*, which builds a PLC machine on the controller, is preinstalled.

Master driver of the *DETEL* protocol is done in C++ and integrated into the *KUMANDA*. Slave type implementation of the *DETEL* protocol is done in avr-assembler language and embedded into the *VPM*. For the sake of completeness, primary commands which are implemented in *DETEL-A* protocol are listed in Table.2.

TEST AND RESULT

The echo command is executed on the *KUMANDA* in order to indicate that the *DETEL Serial Data Transmission Protocol* is running as expected. Fig.6 shows the screen snippet of the KUMANDA after execution of the *echo* command.

The echo command keeps 128 (0x80) data bytes. Data bytes start from 0 (0x00) and enumerates till 127 (0xFD). When the slave device detects an echo command in the received telegram, it reflects the entire telegram back to the master device by using the *DETEL-T* protocol. The reflected telegram is caught by *KUMANDA* and displayed on the output screen after formatting. What is sent by the master is received back entirely as expected.

DISCUSSION & CONCLUSION

Although *DETEL Serial Data Transmission Protocol* is principally designed for communicating electronic devices forming a single master-slave system, multi slave systems forming ring or star type connection in topology, can also adopt *DETEL Protocol* to build communication among their members. In those systems, however, *DETEL Telegram* should be slightly modified for addressing slave devices with which the master wants to communicate because each device in the network (connection) has to have unique device identification throughout the network. Only the slave with which the master wants to speak must react to the coming telegram. It is possible to implement one the of reserved control bytes as a slave address in *DETEL Telegram*. In cases where one byte does not cover the address range in network, combination of reserved bytes can be used

DETEL Protocol can further be used among intelligent electronic elements lying on a single printed board. Two micro controllers which are serially connected to each other, either through I²C or TWI interfaces, for instance, can adopt *DETEL Protocol* for mutual communication purposes.

The transmission speed of the *DETEL* Protocol is dependent upon the physical connection among devices. It is obvious that the effective data transmission speed is reduced because of the nature of the half mode transmission of *DETEL Telegram*.

Although maximum data count is limited by the maximum value of one byte, (0xFF), it can easily be extended by defining a new command in which length of data is redefined by making use of reserved control bytes.

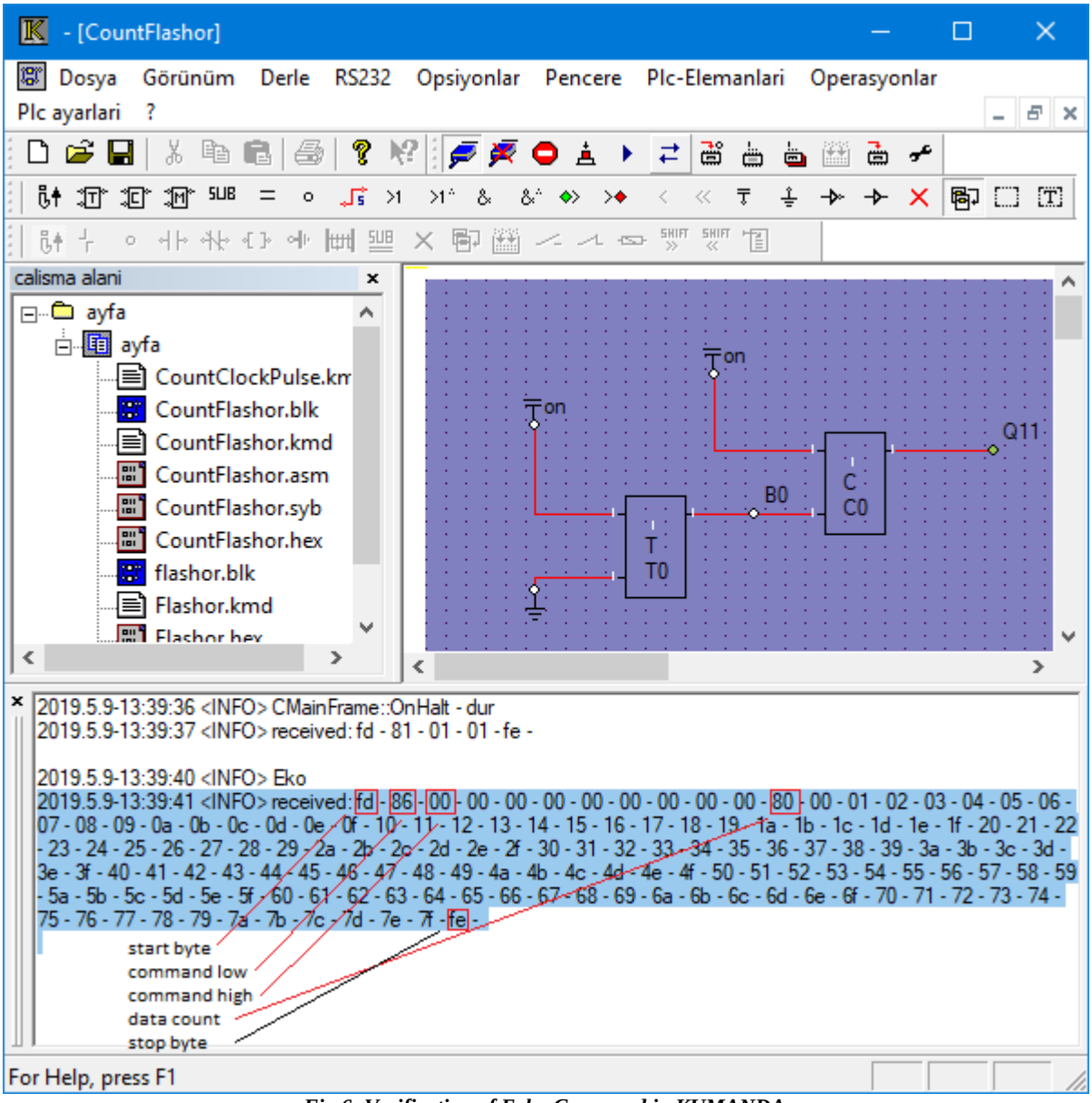

Fig.6: Verification of Echo Command in KUMANDA

On the light of aforementioned discussion it can be conclude that the *DETEL Serial Data Transmission Protocol* can be engaged in following communicating system(s) without yielding any problem.

- System where a new set of command is to be defined.
- System where slave device is expected to send back an acknowledgment on the reception of a command. In those cases it makes sense defining a set of acknowledgments in form of *DETEL Telegram*.
- System where interactive communication is preferred rather than dummy slaves.

ISSN 2349-4506
Impact Factor: 3.799
Global Journal of Engineering Science and Research Management

PROPOSAL

The main short coming of the *DETEL Data Transmission Protocol* lies in the *half mode transmission* of the *DETEL Telegram*. This short coming can be eliminated by implementing *full mode transmission* of the *DETEL Telegram*.

In *full mode transmission*, duplication of the data byte would be avoided by defining a new sequence of *Special Bytes* (eighth specially defined bytes in sequence, for instance). This sequence of bytes is pre hanged to the data byte if the data byte is equal to either *Start*- or *End-Byte* in value. So modified data segment will be transmitted by the master device directly without further expansion. On the slave side, the decoder machine should recognize the sequence of *Special Bytes* and interpret there after coming byte as the data byte although it is equal to either *Start*- or *End-Byte* in value. *Start*- and *End-Byte* will only be valid control bytes if they do not appear after the sequence of *Special Bytes*. Otherwise they will be accepted as a data byte.

Such an algorithm would reduce duplication of the data byte remarkably and proposed by the author as the next study.

ABBREVIATION

RS232:	RS323 serial interface
PLC:	Programmable Logic Control.
PC:	IBM Personal Computer.
KUMANDA:	Integrated Development Environment for the MDxx series of PLCs.
XON:	Start byte in *DETEL* Telegram.
NILL:	Null byte in *DETEL* Telegram.
XOF:	End byte in *DETEL* Telegram.
iTBAS:	Telegram Base.
ISR:	Interrupt Service Routine.

REFERENCES

1. MDxx Series of PLCs, ' http://www.plcturk.com'
2. International Organization for Standardization, "*Open System Interconnection (OSI)"* 'https://www.iso.org/ics/35.100/x/'
3. KUMANDA, '*Integrated Developing Environment*', IDE, for Programming MDxx Series of PLCs, 'http://www.plcturk.com/webserver/kumanda/IDEList_01.htm '
4. ATMEL, ATmega16 Data Sheet, Serial Port Data Receive Register, ' http://www.atmel.com '.