BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 62 (66), Numărul 2, 2016
Secţia
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ

MANAGING SERIAL EEPROM MEMORY THROUGH I^{2}C WITH ATMEL FAMILY MICROCONTROLLER

BY

PETRUŢ DUMA[*]

“Gheorghe Asachi” Technical University of Iaşi
Faculty of Electronics, Telecommunications and Information Technology

Received: April 22, 2016
Accepted for publication: May 30, 2016

Abstract. The work describes the structure of the interface used for saving data in a serial EEPROM memory. A microcontroller equipped development system is used for managing the memory, the real-time clock, various sensors and transducers. The acquired data is concatenated and stored in an EEPROM memory chip similar to a database, as records. The program that was written includes commands for displaying the database, substituting data in the database records, transferring the database from the EEPROM to a personal computer using an asynchronous serial interface, deleting records from the database, defining the database structure. Other commands are also available in order to perform basic EEPROM operations, such as: displaying, substituting, loading, moving, deleting, receiving/transmitting data blocks and so on.

Key words: electrically erasable programmable read only memory; non-volatile serial memory; I^{2}C protocol; database; microcontroller; command program.

1. Introduction

The user application processes manage various parameters that alter in time, collected from digital or analogue inputs and outputs, but also from the signals acquired from sensors, transducers and other circuits of the system. All this data is usually loaded into a volatile memory and therefore is lost when the

[*]Corresponding author: *e-mail*: pduma@etti.tuiasi.ro

power supply of the system is shut down. In order to eliminate this downside, but also in order to maintain and access the history of the data acquired, the use of a non-volatile memory is recommended, able to store all the process parameters values that change in time. Serial EEPROM memory chips are preferable, since they have a medium storage capacity, they can be written and deleted while connected in the system's circuitry, they have a reduced number of external connections and a simple interface. However, they do have a slightly longer command cycle.

An application system equipped with an ATMEL family micro-controller is used for the implementation of an user process that monitors in real-time various parameters. The basic structure of this system is shown in Fig. 1; the notations used have the following meanings: AS_µC – application system equipped with microcontroller; SEM – serial EEPROM memory; RTC – real time clock; DDC – data display console; DI – digital inputs; DO – digital outputs; AI – analogue inputs; AO – analogue output; ADC – analogue-digital converter; DAC – digital-analogue converter; STC – sensors, transducers and other circuits; RS232_SI – RS232 serial interface; PC – personal computer.

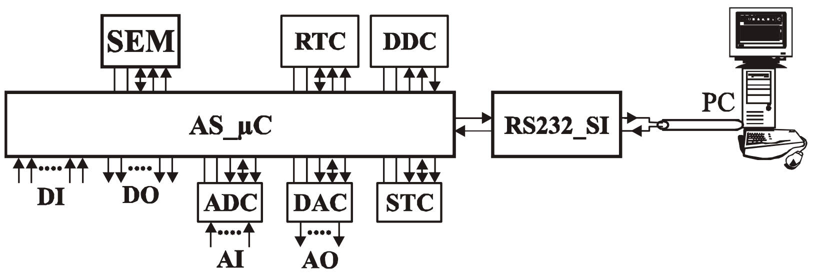

Fig. 1

This work covers the serial EEPROM memory management. Subsequent papers will deal with sensors and real-time clock monitoring, various digital and analogue signals acquisition etc.

2. EEPROM Memory

The EEPROM circuits (Electrically Erasable Programmable Read Only Memory) are in fact non-volatile memory chips used for storing data in computers, electronic equipment, microprocessor and microcontroller systems. The most important feature of these memory chips is that the data stored is not lost when the power shuts down. Basically, these memory chips consist of a cell matrix, each cell consisting of a pair of transistors separated by a thin layer of insulating oxide. These memory chips can be repeatedly erased and written while connected in the actual circuit, by applying a higher voltage internally generated.

The command of the EEPROM memory circuits is made through a serial or a parallel interface. The most used serial interfaces are: I^2C, SPI, Microwire, UNI/0 and 1-Wire. They use between one and four command and control signals, with distinct sets of instructions, in order to perform various functions. This work analyses the EEPROM memory circuits with I^2C serial interface.

Their main features are: powered from a single source of continuous voltage ranging from 1.7 V to 5.5 V for AA and FC series and 2.5 V to 5.5 V for LC series respectively; built in low power CMOS technology; I^2C compatible serial interface; up to three address lines allowing to cascade up to 8 circuits on a bus; noise reduction based on trigger Schmitt inputs; output slope control to eliminate ground balance; automatically timed data write cycle, including memory locations erasure; operation modes: normal, fast and fast plus, using clock frequencies of 100 KHz, 400 KHz and 1 MHz respectively, thus allowing a data transfer rate of 100 Kbps, 400 Kbps and 1 Mbps respectively; data writing buffers in memory pages of up to 128 bytes, depending on the memory circuits capacity; writing a memory page takes up to 5 ms; hardware write protected; more than 1,000,000 write/erase cycles supported; over 200 years data retention period without power; factory programming available; packaged in 8-terminal capsules PDIP, SOIC, TSSOP, MSOP, DFN and in 5-terminal capsules SOT-23; operating temperature ranges for commercial, industrial and automotive standards.

Serial EEPROM memory chips are manufactured, for example, by Microchip Technology in a capacity range from 128 bits to 1 MBits (Table 1) and are organized in 8-bit words.

Table 1

Memory circuit	Capacity	Organization
24xx*00	128 bits	16 bytes
24xx*01	1 Kbit	128 bytes
24xx*02	2 Kbits	256 bytes
24xx*04	4 Kbits	512 bytes
24xx*08	8 Kbits	1 Kbyte
24xx*16	16 Kbits	2 Kbytes
24xx*32	32 Kbits	4 Kbytes
24xx*64	64 Kbits	8 Kbytes
24xx*128	128 Kbits	16 Kbytes
24xx*256	256 Kbits	32 Kbytes
24xx*512	512 Kbits	64 Kbytes
24xx*1025/1026	1 Mbit	128 Kbytes

* where xx is AA/LC/C/FC

The internal block schematics of a serial EEPROM memory chip is shown in Fig. 2, the notations having the following meanings: LI/OC – logical input/output control; LMC – logical memory control; AC – address counter;

XDEC, YDEC – row and column decoder; HVG – high voltage generator; EEPROM MMA – EEPROM memory matrix area; PR – page register (for writing); R/WBC – read/write buffers control; PSC – power supply circuit; POR – Power On Reset.

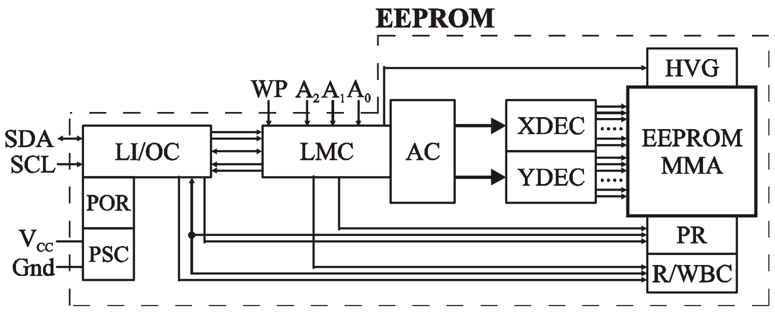

Fig. 2

The logical control of the inputs/outputs manages the bi-directional data line (SDA) and the clock input (SCL) in order to set the reception of the START and STOP conditions, to receive the device selection and the memory read/write commands, to receive the address and data bytes, to send data bytes, to receive or to send acknowledge bits.

The logical memory control checks the device addressing based on the address lines $A_2A_1A_0$ and on the command sent, writes the internal address counter, selects the row and column decoder, reads the EEPROM memory, sends the read data, writes the page register, commands the high voltage generator, writes the EEPROM memory, provides all the required command and control signals for the memory area.

The power on reset blocks the execution of the memory commands when the power voltage drops below a certain minimal threshold.

3. Interfacing the Serial EEPROM Memory

The command and control of the serial EEPROM memory used initially in testing, checking and developing the application is built using a development system equipped with an ATMEL family microcontroller. Fig. 3 shows the interface used for managing the memory using port P1 of the microcontroller.

The EEPROM memory is powered from a DC supply of 3.3V, also used for several sensors and transducers used in the application. The hardware interface supplies the power from the three-point positive voltage stabilizer TS1117_3V3, that only requires decoupling capacitors (100 nF and 22 μF). The power on command ($\overline{\mathrm{CA}}$) is sent through an open collector buffer and a transistor *T* (BC 556); the power on signal is active on the logical level 0 and is provided through software on the line *P*1.5 of the microcontroller. After powering on the memory, a short time is necessary for the V_{CC} voltage to stabilize, after which the device passes in standby mode. The consumed current does not exceed 0.4,...,1 mA for read operations, 3,...,5 mA for write operations and 1,...,5 μA when in standby, the range depending on the capacities of the different memory chips.

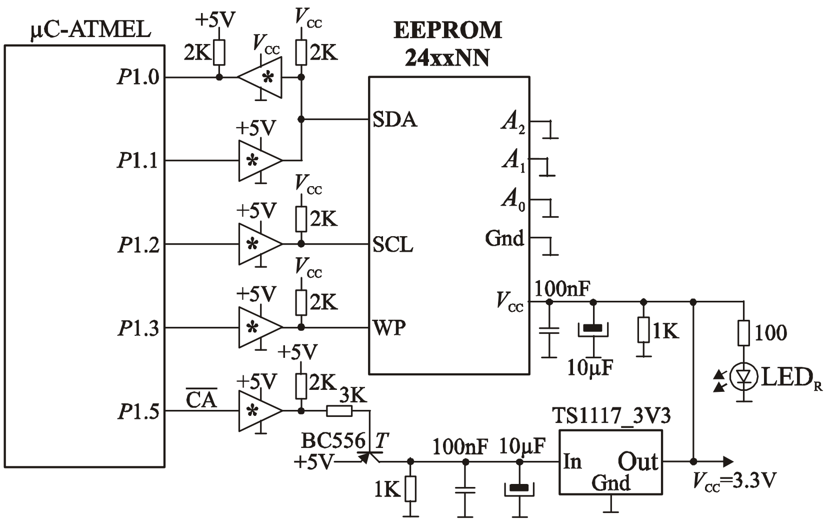

Fig. 3

The bidirectional serial data line SDA, the serial clock input SCL and the write protect input WP are connected through open collector buffers to the lines of port *P*1 of the ATMEL microcontroller (Fig. 3). These buffers perform the logical levels translation between the microcontroller of the development system powered at +5 V and the serial EEPROM memory chip that, along with the application's sensors and transducers, are powered at 3.3 V. The final application is managed by an application system equipped with an ATMEL microcontroller powered at 3.3 V, these buffers no longer being necessary.

4. Monitoring the EEPROM Memory

The EEPROM memory supports a bidirectional data transmission/ reception protocol on a serial I^{2}C (Inter-Integrated Circuit) with two lines. The microcontroller that manages the communication is master and provides the clock used for synchronisation, while the serial EEPROM memory is always the slave. In order to perform a write or a read operation from the memory, the microcontroller generates a command that sends the START condition, the control byte, the memory location address, then the data bytes are sent for write operations or receives the data bytes for read operations and, finally, the STOP condition is sent. A data transfer can be initiated only if the I^{2}C bus is free, that is the data line SDA and the clock line SCL are in logical 1.

The START condition (*S*) consists of a falling edge on the data line SDA, while the clock line SCL is stable in logical level 1 (Fig. 4). Any data transfer command starts with this condition. The slave devices must receive and identify any START condition in order to perform a data transfer, and must not execute any command in the absence of this condition.

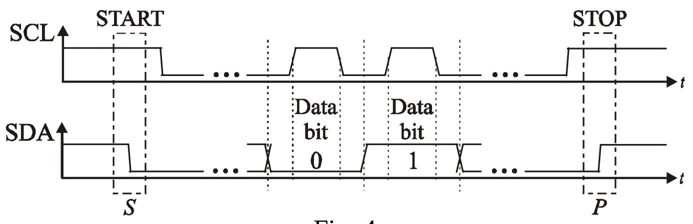

Fig. 4

The STOP condition (P) consists of a rising edge on the data line SDA, while clock line SCL is stable in logical level 1 (Fig. 4). This condition ends the communication between the slave memory and the master microcontroller. The condition determines the slave device to enter the standby mode after a read command, and, respectively, to initiate the internal write cycle after a write command. The data bits are modified only during logical 0 level of the clock signal and must remain stable during logical 1 level of the clock signal. The changes on the data line while the clock line is in logical 1 provide the control signals of the protocol I^2C.

The data bits of a byte are transferred bit by bit between the communicating circuits (Fig. 5) and are transmitted on SDA starting with the most significant bit (b_7) and ending with the least significant bit (b_0). After sending the eight data bits of a byte, the receiving circuit acknowledges the received byte using an acknowledgement bit during the ninth clock period provided by the master. The acknowledgement bit has logical level 0.

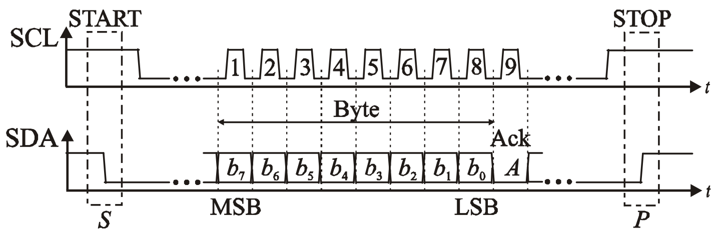

Fig. 5

The slave memory confirms the reception using an acknowledgement bit (A) after each received byte. The master microcontroller acknowledges using a dedicated bit (A) after each data byte read from the memory, but this mode allows reading of the memory location with the consecutive address. As the master microcontroller no longer requires reading the next location, it concludes the read command with a not acknowledge bit $\overline{A}$, defined as logical 1 (Fig. 6).

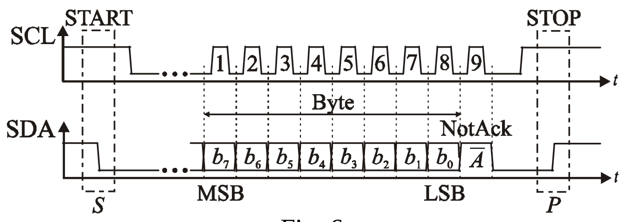

Fig. 6

Any command includes the following byte types: control, address and data (Table 2).

Table 2

Type of Bytes	Byte								Obs.
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
Control Byte	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	–
	1	0	1	0	A_2	A_1	A_0	I	
Address Byte	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Low address
	a_{15}	a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	High address
Data Byte	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	–

The control byte consists of four bits ($c_7c_6c_5c_4$) used to identify the specific circuit class, three address bits ($c_3c_2c_1$) and a bit (c_0) for the read/write indicator I= R/$\overline{\mathrm{W}}$.

The serial EEPROM memory circuits with I^{2}C protocol are identified with the following bit combination: 1010.

The address bits $c_3c_2c_1$ = $A_2A_1A_0$, for some of the serial EEPROM memory chips with I^{2}C protocol, manufactured by Microchip have no significance. For some circuits they are used to select a memory block of a certain capacity from the EEPROM memory area, while for other chips, they have the role to select a specific device from several connected to the I^{2}C bus.

The read/write indicator selects the operating mode of the slave circuit: if I = 0, the slave receiver mode is selected, therefore data being written into the memory chip; if I = 1, the slave transmitter mode is selected, data being read from memory.

There is only a single address byte sent by the microcontroller for serial EEPROM memory chips with the capacity less or equal with 16 Kbits, while for memory chips with capacities equal or higher than 32 Kbits, there are two address bytes. In this latter situation, after the control byte, the address byte containing the most significant bits of the address is sent, followed by the address byte containing the least significant eight bits.

The data byte is sent by the master microcontroller for a memory write command (I = 0) or by the slave memory chip for a read command (I = 1). After any EEPROM memory location read or write operation, an automatic increment of the internal address counter is performed.

The EEPROM memory data byte write operation is performed according to these steps: the microcontroller sends the START condition, then the control byte with I = 0, the address byte, the data byte (after each sent byte, the microcontroller receives the acknowledge bit) and, in the end, the STOP condition (Fig. 7).

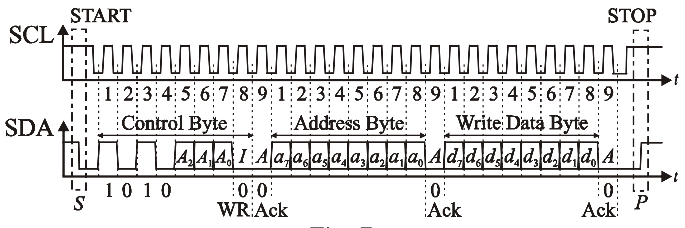

Fig. 7

The command for writing a memory page with data bytes is executed in the same manner as the previously described command, except for the fact that, after the data byte, the consecutive data bytes are sent, then in the end, the STOP condition (Fig. 8). The number of bytes written into a page must not exceed the capacity of the page register.

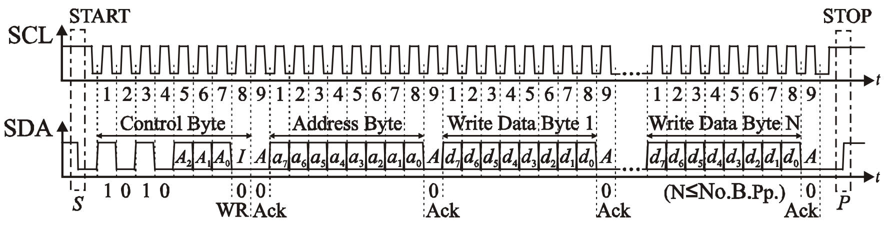

Fig. 8

The command for reading a data byte from the EEPROM memory is performed as follows: the microcontroller transmits the START condition, the control byte with I = 1 (after sending this byte, the microcontroller receives the acknowledgement bit), then the microcontroller receives the data byte sent by the memory chip from the location defined by the internal address counter. Afterwards, the microcontroller sends a not acknowledge bit and concludes the executed command by the condition STOP (Fig. 9). This last command has the drawback of not allowing the reading of an EEPROM memory location from any address.

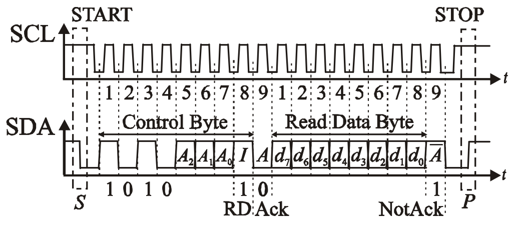

Fig. 9

The command used to eliminate this drawback, allowing the random read for a data byte from any memory location is performed using the following steps: the microcontroller sends the START condition, the control byte with I = 0 and the address byte. These steps write the internal address counter with the required value. Then the same steps are performed as in the reading a data byte previous command, starting with the retransmission of the START condition, followed by all the others (Fig. 10).

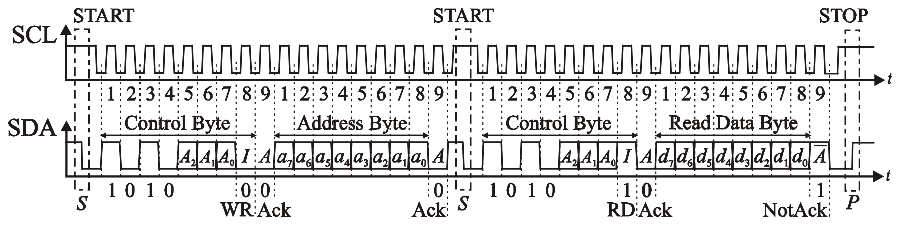
START
START
STOP
SCL
SDA
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
Control Byte
Address Byte
Control Byte
Read Data Byte
$A_2 A_1 A_0$ I A $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$ A
$A_2 A_1 A_0$ I A $d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0$ $\bar{A}$
t
S 1 0 1 0 0 0 WR Ack 0 Ack S 1 0 1 0 1 0 RD Ack 1 NotAck P

Fig. 10

The command for sequential reading of a data block from the EEPROM memory is performed similarly as the last two commands, except the fact that, after the microcontroller receives the data byte, it sends the acknowledgement bit (A = 0), then it receives the next data bytes from consecutive memory addresses. In the end, the microcontroller sends a not acknowledgement bit ($\bar{A}$ = 1) and closes the command with a STOP condition (Fig. 11).

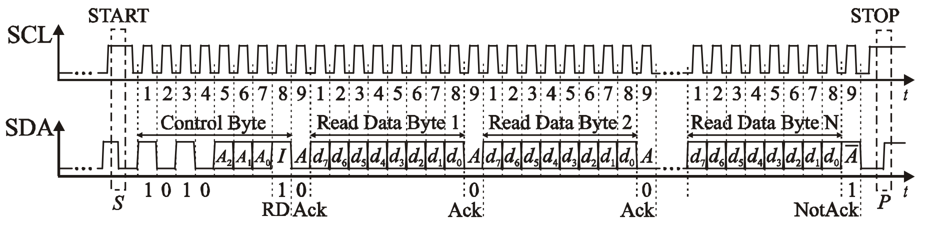
START
STOP
SCL
SDA
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
Control Byte
Read Data Byte 1
Read Data Byte 2
Read Data Byte N
$A_2 A_1 A_0$ I A $d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0$ A $d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0$ A $d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0$ $\bar{A}$
t
S 1 0 1 0 1 0 RD Ack 0 Ack 0 Ack 1 NotAck P

Fig. 11

The serial EEPROM memory management program is based on a series of elementary program sequences performing the following functions: **TxS** – transmits the START condition; **TxP** - transmits the STOP condition; **TxCBN0** – transmits the control byte with address $A_2A_1A_0 = N$ and indicator bit I = 0, for memory write; **TxCBN1** – transmits the control byte with address $A_2A_1A_0 = N$ and indicator bit I = 1, for memory read; **RxA** – receives the acknowledgement bit to the memory chip; **TxA** – transmits the acknowledgement bit to the memory chip; **TxNA** – transmits the not acknowledgement bit to the memory chip; **TxAB** – transmits the byte with the memory location address; for some memory types, $A_2A_1A_0$ is loaded with the most significant bits of the address; for memories with capacity higher or equal to 32 Kbits, two address bytes are sent; **TxDB** – sends the data byte for memory write operation; **RxDB** – receives the data byte read from the memory.

Using these basic program sequences, a few essential subroutines were written, that are dealing with: **WRB** – write a data byte into the memory, at a certain address; **WRP** – write a page of data into the memory, at a certain address; **WRBD** – write a data block into the memory, at a certain address (the block consists of several pages); **RDB** – read from the memory a data byte from the current address of the internal address counter; **RDBR** – read from the memory a data byte from any random address; **RDBD** – read from the memory a data block starting from the current address of the internal address counter; **RDBDR** – read from the memory a data block starting from any random address.

These subroutines are used in the user application program for reading and writing data from/to the EEPROM memory.

A data table is defined in the program memory, including all the specifications of the EEPROM memory chips from the family (capacity, number of address bytes, number of address bits, number of bytes in the write buffer, clock frequency, write time etc.).

The data acquired from sensors, digital or analogue inputs/outputs, including the real time clock, are concatenated and stored unprocessed into the memory, as records.

If the application system is connected to a console or a personal computer through the asynchronous serial interface, then user commands are available, offering more features.

The commands implemented through software for managing the serial EEPROM memory are described concisely below. Any command is concluded using terminator RETURN (**.**), while between the parameters, the separator BLANK (**_**) is used. Some of the commands use the same BLANK separator in order to enumerate the following byte, data block, record etc. while separator COMMA (**,**) is used to enumerate the previous element.

A.
Display the list of available commands.

B_
Set the baud rate of the serial asynchronous communication of the UART interface (one of the standard values between 150 bps and 19,200 bps).

C.
Delete all the numeric values from the all database stored into the EEPROM memory; all the corresponding memory locations are filled with 0FFH.

C p1_p2.
Delete all the numeric values of the variables from the database, starting from record p1 until record p2; the command also compacts the database if necessary.

DB p1_
Display on the console the contents of an EEPROM memory block from address p1 until address p1 + 80H; on each console row it is displayed in brackets the address of the memory location, followed by the contents in hexadecimal of the 16 consecutive memory locations.

DI p1_

Display on the console eight records from the database stored in the EEPROM memory, starting with record p1; each record is displayed on a new row and consists of printing all the names and hexadecimal values of variables in the record.

DT p1_

Display on eight rows on the console a text consisting of the characters corresponding to the ASCII codes stored in the EEPROM memory starting from address p1; the displays continues, waits or is stopped by pressing the keys C, A and O respectively on the system's keyboard.

E_

Set the EEPROM memory type used in the application.

F p1_p2_p3.

Fill the EEPROM memory area starting from address p1 until address p2 with the hexadecimal data p3.

L.

Load the database from the EEPROM memory, converted into a TAB delimited text file, into a personal computer; the hexadecimal values are converted into corresponding ASCII characters, TAB delimiters (09H) are inserted between the numeric values of the variables and, after each record, there are inserted the codes CR and LF (0DH and 0AH).

L p1_p2.

Load the database converted into a text file, from record p1 to record p2, into a personal computer.

M p1_p2_p3.

Move the EEPROM memory area starting from address p1 until address p2 into the memory area starting with address p3.

P.

Sets the write protect of the EEPROM memory.

R p1_p2.

Receive from a personal computer a data block that is loaded into the EEPROM memory starting from address p1 until address p2.

S_

Define the structure of the database; an information message is inserted, that includes: the number of variables and, for each variable, the number of bits and the name of the variable, consisting of maximum four ASCII codes.

SB p1_

Display and/or substitute the contents of the EEPROM memory locations starting from address p1; the address of the memory location is displayed in brackets, then its contents; the user can input a new value or can just continue the command with no alteration to the memory contents.

SI p1_

Display and/or substitute the numeric values of the variables starting from record p1; the name of the variable is displayed, then its contents; the user can input a new value or can just continue the command with no alteration to the variable value.

ST p1_

Substitute the contents of the EEPROM memory locations starting from address p1 with the ASCII codes of the keys pressed; the command ends when CTRL+P is pressed.

T p1_p2.

Transmit a data block from the EEPROM memory starting from address p1 until address p2 to a personal computer.

V.

Connect/disconnect the power supply V_{CC} used to power on the serial EEPROM memory and the sensors of the application.

5. Conclusions

The hardware structure described was implemented in practice. It consists of the serial EEPROM memory requiring a minimal amount of external components, along with digital sensors (temperature, pressure, humidity etc.), a hardware real-time clock and a development system equipped with AT89S8253 microcontroller.

The command program is based on a few subroutines that allow writing and reading a byte or a data block starting from any address in the memory. These subroutines access a data table allowing them to be used for the management of any serial EEPROM memory chip from the same manufacturer, with various capacities.

The EEPROM memory is used as storage for a database-like structure consisting of records including the numeric values corresponding to the parameters acquired at pre-programmed time intervals, their extreme and accidental values, along with the value of the real-time clock and other analogue or digital inputs/outputs of the user process.

There were also implemented through software, commands that allow defining a database, to display and edit its contents, to load it from the EEPROM memory into a personal computer as a text file.

Other commands perform memory data display, data substitution, loading a memory area with a value, moving the contents of a memory area, sending and receiving memory data blocks to/from a personal computer etc.

The program was written in machine code and it uses a program memory area of 3.7 KB. It stands out by the small amount of memory that it uses, compared to the features and functions offered.

REFERENCES

Aghion C., Ursaru O., *Aplicații practice ale microcontrolerelor*, Edit. PIM, Iaşi, 2009.
Balan R., *Microcontrolere. Structură şi aplicații*, Edit. Todescu, Cluj-Napoca, 2002.

Blixt B., *Serial EEPROM Overview*, MICROCHIP, 2008.
Duma P., *Microcontrolere în telecomunicații*. Edit. TEHNOPRESS, Iaşi, 2007.
Duma P., *Monitoring the DS1307 Real Time Clock Using an ATMEL Family Microcontroller*, Bul. Inst. Politehnic, Iaşi, **LIX(LXIII)**, *3*, s. Electrotehnică, Energetică, Electronică, 83-93 (2013).
Duma P., *Monitoring the SHT75 Relative Humidity Sensor Using an ATMEL Family Microcontroller*, Bul. Inst. Politehnic, Iaşi, **LX(LXIV)**, *1*, s. Electrotehnică, Energetică, Electronică, 35-46 (2014).
Duma P., Petac E., Alzoubaidi A.R., *Monitoring the MPL115A1 Pressure Sensor Using a ATMEL Family Microcontroller*, Bul. Inst. Politehnic, Iaşi, **LIX(LXIII)**, *1*, s. Electrotehnică, Energetică, Electronică, 73-84 (2013).
Hintz J.K., Tabak D., *Microcontrollers. Arhitecture, Implementation and Programming*, McGraw Hill, New York, 1993.
Leventhal L.A., *Programmation en langage assembleur*, Edit. Radio, Paris, 1998.
Peatmann B.J., *Design with Microcontrollers*, McGraw Hill, New York, 1998.
Petreuş D., Muntean G., Juhos Z., Palaghița N., *Aplicații cu Microcontrolere din Familia 8051*, Edit. Mediamira, Cluj-Napoca, 2005.
Valeanu A., *Interfacing 8051 MCUs with I^2C^{TM} Serial EEPROMs*, AN1147, MICROCHIP, 2008.
Valeanu A., *Using ASM and a Hardware Module to Interface 8051 MCUs with I^2C^{TM} Serial EEPROMs*, AN1190, MICROCHIP, 2008.
* * * ATMEL, *Family Microcontroller*, Data Book, 1998.
* * * ATMEL, *CMOS Memory*, Data Book, 2004.
* * * ATMEL, *AT89S8253 Microcontroller*, Data Sheet, 2005.
* * * MAXIM, *Multichannel RS232 Drivers/Receivers*, MAX232A Data Sheet, 2006.
* * * MICROCHIP, *CMOS Memory Products*, Serial EEPROM, 2015.
* * * MICROCHIP, *128 Bits/ 1 Kbit/ 2 Kbits/ 4 Kbits/ 8 Kbits/ 16 Kbits/ 32 Kbits/ 64 Kbits/ 128 Kbits/ 256 Kbits/ 512 Kbits/ 1024 Kbits I^2C^{TM} CMOS Serial EEPROM*, 24AA/LC/C00, 24AA/LC/C01, 24AA/LC/C02, 24AA/LC04, 24AA/LC08, 24AA/LC16, 24AA/LC32, 24AA/LC/FC64, 24AA/LC/FC128, 24AA/LC/FC256, 24AA/LC/FC512/515, 24AA/LC/FC1025/1026 Data Sheet, 2007-2015.
* * * MICROCHIP, *PICmicro^{TM} Mid-Range MCU Family*, Reference Manual, 1997.
* * * Taiwan Semiconductor, *Low Dropout Positive Voltage Regulator*, TS1117 Data Sheet, 2003.
* * * Texas Instruments, *Digital Logic*, Data Book, 2007.

GESTIONAREA MEMORIILOR SERIALE EEPROM PRIN I^2C CU MICROCONTROLER DIN FAMILIA ATMEL

(Rezumat)

Se descrie structura interfeței pentru salvarea datelor într-o memorie serială EEPROM. Un sistem de dezvoltare echipat cu microcontroler gestionează memoria, ceasul de timp real, diferiți senzori şi traductoare. Datele achiziționate sunt concatenate şi salvate în memoria EEPROM ca într-o bază de date sub formă de înregistrări. Programul scris are o serie de comenzi care afişează baza de date, substituie date din înregistrările bazei de date, transferă baza de date din memoria EEPROM într-un

calculator personal prin intermediul interfeței seriale asincrone, şterge înregistrări din baza de date, defineşte structura bazei de date. Sunt disponibile şi alte comenzi pentru a realiza funcții de bază cu memoria EEPROM, cum ar fi: afişarea, substituirea, încărcarea , mutarea, ştergerea, recepționarea/ transmiterea blocurilor de date şi altele.