

Multiprocessor Communication using 8051 Microcontroller and RS-485 Line Driver

Kashif Altaf, Javaid Iqbal
Department of Mechatronics, College of E&ME
National University of Science and Technology
Rawalpindi, Pakistan
kashifaltaf1@gmail.com, jiqbal-EME@nust.edu.pk

Abstract---Embedded systems are the brains of today's most digital and industrial control systems. In systems where more than one processor is incorporated, the need for multiprocessor communication often arises. Products that might exploit this feature include data terminals which receive updates from a central control system, sensor systems which are interrogated by a monitoring station and multi-axis robots with each axis under the control of its own processor, to name a few.

In this paper, the author has discussed a simplified prototype of industrial multiprocessor communication systems using ATMEL 89C51 (8051) microcontroller, via serial communication RS-485 protocol. He has constructed and simulated such a model for experimental purposes. The experimental results and the problems faced by the author in the way to get the task done are discussed. The interfacing of Hitachi Liquid Crystal Display (LCD) 44780 and Hex Keypad to this microcontroller, and the operation of three microcontrollers in master-slave configuration is elaborated.

The communication protocol developed in this paper may be extended to meet the data transfer requirements in an industrial setting.

I. INTRODUCTION

There has been a great shift of trend towards fabrication and industrial use of embedded systems during the last three decades owing to manifold increase in their applications. In an industrial setting where there are to be two or more inter-related processing systems, the need for multiprocessor communications is indispensable. The microprocessors or microcontrollers may communicate via serial or parallel mode. Serial communication can be accomplished via many protocols e.g. RS-232, RS-422 and RS-485. Among them, RS-485 is the most superior in terms of its distance range and the maximum number of communicating processors supported.

8-bit microcontrollers are widely used mainly because they are simpler and easier to use than 16-bit or 32-bit microcontrollers, and also due to their low cost. Among them, Motorola 68HC11, Microchip PIC family, and the industry standard 8051 are prevalent [1]. To interface these devices to the outside world, input and output devices, like Keypad and LCDs respectively, are generally used [2].

In this paper the serial RS-485 mode of communication using 8051 is the main concern, with an emphasis on interfacing it with LCD Hitachi 44780. An approach towards

interfacing hex keypad to 8051, and scanning it without use of any driver IC is also discussed.

Even though many simulators are available for the 8051 family [3] [4] [5], most of them do not support multi-microcontroller operation, nor do they simulate all the embedded peripherals, and only a few can simulate external peripherals. So, programming, simulation and debugging have been done in UVI51, a software tool equipped with all these provisions [6].

We have used two slaves (1 and 2) and a master controller, with an LCD screen and a hex keypad attached to the master, a hex keypad and a seven-segment-display (SSD) board with four SSD displays on it to Slave 1, and a similar SSD board to Slave 2. The serial communication is performed via RS-485 standard.

II. RS-485 PROTOCOL

The RS485 is a serial communication protocol that uses balanced/ differential voltage signals for transmission, rather than ground-referenced signals as is in case of RS-232 [7]. RS-485 is similar to RS-422 with the difference in that unlike RS-422, it supports multiple drivers and receivers [8]. It is used for multipoint communications: more devices may be connected to a single signal cable [9]. Most RS-485 systems use Master/ Slave architecture, where each slave unit has its unique address and responds only to packets addressed to this unit [10]. The common Master generates these packets.

An RS-485 link can extend as far as 1200 (meters) and can transfer data at up to 10 (Mbps), but not both at the same time. Additional devices in form of slaves may be accessed. Two wires (usually a twisted pair) carry the signal voltage and its inverse. The receiver detects the difference between the two. Because most noise that couples into the wires is common to both wires, it cancels out.

III. CONTROLLER BOARDS

We have used three controller circuits, each containing an 8051 with I/O port terminals supplied with buffers, to avoid unnecessary load on the controller in terms of current. Both RS-232 and RS-485 are available on the board for providing the user with options of switching from one communication protocol to the other. To provide visual information about

the status of each I/O pin, it is connected through a buffer LM 245 to ground via Light-emitting-diodes (LEDs), whose current requirements being met by the buffer. An on-board DB9 female connector is also supplied on each board, as shown in Fig. 1.

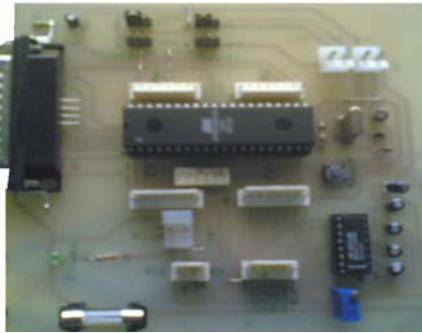


Fig. 1. A sample controller board

IV. SEVEN SEGMENT DISPLAY (SSD) BOARDS

We have used two SSD boards, each connected to a slave. Each SSD board has a set of four common anode SSDs, and to reduce the number of SSD control pins, a Binary-coded-decimal (BCD) to seven segment decoder LM 47 is used with each SSD, in combination with an LM 573 latch. Controller sends the BCD number (to be displayed on the SSD) to latch, and at the same time toggles its LE pin high temporarily, to feed it with new data, which otherwise is kept low at all times. The latch receives this data, passes it onto decoder and latches it. Decoder converts this BCD number to its 7-segment code and transmits it to the SSD. LE is then pulled low. Fig. 2 shows connections of SSDs on the board.

LE is playing a very important role here. If it is high, latch accepts the BCD number at its input and latches it. Then, if LE is pulled low, latch will not accept any new input to its pin, and will keep on latching the previous output that was a result of the input to it when LE was pulled up previously. In simple words, it is analogous to a gate. If LE is 1, gate is open and vice versa.

The LE pins of the four latches (one connected to each SSD) are connected to each bit of a 4-bit terminal on the board, and this 4-bit nibble serves as an “SSD control nibble” when separate data are to be sent to each of 4 SSDs.

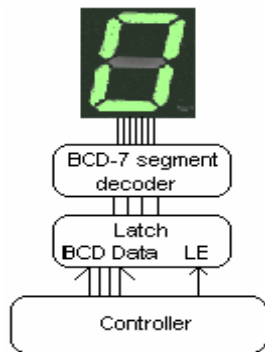


Fig. 2. SSD board connections

All the four SSDs have access to the BCD input, but only one with its LE pin enabled can accept it. At one time, data can be sent to only one SSD, by setting its LE and resetting other three SSDs' LE pins, which, thereby maintain their previous status. For a four-digit data, the four SSDs are enabled one by one, and in other words, a “select and send data” approach is used.

V. GOALS

To establish a multiprocessor communication system in a master-slave configuration, three 8051 controllers in master-slave configuration are connected. The aim is to perform master to slave communication and vice versa. A set of rules have been set, and 2-way communication is established while abiding by these rules. A visual display of the communication scheme is shown in Fig. 3.

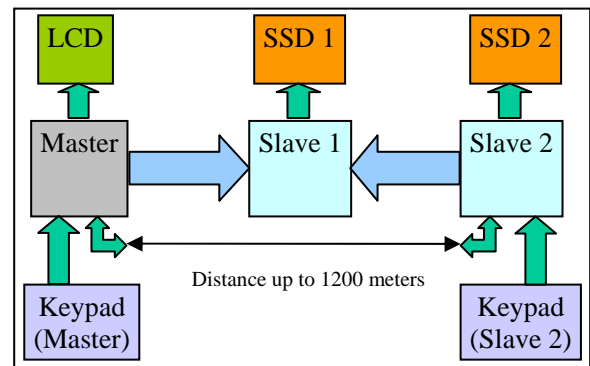


Fig. 3. Signal flow diagram of system

The two slaves are assigned addresses 1 and 2. In Master to Slave communication, the master 8051 has to wait for the address of the slave entered by the user through master keypad, and on pressing any one of the above mentioned keys, the selected slave should display its address on its SSD board, which should remain there until data comes in to take its place. The master LCD should simultaneously display the slave address selected by the user.

Once address has been selected, the user enters 4-bit data via same keypad, which should be displayed both on LCD and on the SSD board of addressed slave, with each SSD showing one digit, after refreshing the previously displayed address. The non-selected slave should remain idle. The master to slave procedure could be repeated any number of times without reset.

In the reverse communication mode, the address entered by the user through slave 2 keypad (not slave 1, since it always acts as slave), should be displayed on its SSD board and the display (SSD/ LCD, whichever applicable) of the addressed master/ slave controller. After this, data entered should be displayed on slave-2 SSD and the display of the addressed device after refreshing the previous information.

The system has to distinguish itself whether to perform communication from master to any selected slave, or in reverse, depending upon circumstances, i.e., which keypad is

pressed first, meaning thereby that if key press happens to be on master keypad, the mode will be master to slave, and if it is on slave 2 keypad, it will be either slave to master or slave to slave mode.

VI. INTER BOARD CONNECTIONS

Since RS-485 works using differential signals rather than a ground-reference, the only need to be fulfilled is to connect the non-inverted “A” terminals of all the three RS-485s together, and same for the inverted “B” terminals of RS-485. These pins are the lines on which differential signals generated by RS-485 get transferred from one terminal to the other. This means that same signals are accessible to all the micros’ 485s, but the only one which is selected at runtime by sending its address (through resetting its DE and \overline{RE} pins, commonly called 485-direction pins), can receive that data.

VII. THE HEX-KEYPAD

Hex keypad is basically a matrix of many rows and columns. The number of pinouts of keypad equals the sum of number of rows and columns in the keypad, with each pin attached to a particular, unique, entire row or entire column via a naked tiny conducting wire, as shown in Fig. 4. All the wires are properly insulated from each other, however, at the places where a wire for a column crosses a wire representing a row, there is a short air-spacing between the two, instead of some insulated material, so that the two wires do not touch each other at their own, but may be made to do so on key press. That is where we press the keys on the keypad. As we press a particular key on the keypad, the spacing between two wires reduces until the two wires, of one particular row and one particular column, touch each other. Different scanning techniques may be developed to determine which key is pressed (generally termed as keypad scanning).

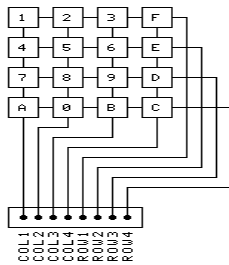


Fig. 4. Internal wiring layout of keypad

VIII. KEYPAD SCANNING

The wires from four columns and four rows on the keypad form a full byte, and a particular port on the master 8051 may be dedicated for interfacing with keypad and scanning it [11]. In our configuration, the more significant nibble on this port is attached to columns portion of keypad, and is selected as the input nibble. The less significant nibble is for the four

rows, which, normally are pulled up to logic “1”, and if any key is pressed on keypad, a particular pin in this nibble captures the data sent to a particular column. Our goal is to search that pin, and also to find out as to data of which pin has been captured there. The scanning scheme is elaborated below, and keypad connections are shown in Fig. 5.

The start is taken from column 0, logic 1 is sent on all other three columns and a 0 on this column 0 through the input nibble 0111.

The output nibble coming from the rows is checked for a zero on anyone pin, which are normally pulled up to a 1, and can read a 0 only if any key in the column 0 is pressed. If no pin in the nibble read is 0, then the input nibble 0111 is shifted to right by one place to yield 1011, and is re-sent on columns, and rows are again checked for any zero. This idle process is continued till user gives some input to the keypad.

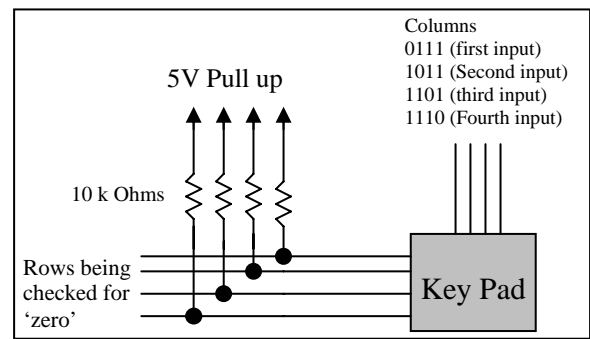


Fig. 5. Keypad connections

Each time we shift the input nibble, a particular variable already set for this purpose, is incremented, starting from zero for first column, to a maximum of three for the fourth column. It represents the column number in which logic 0 is detected, and is saved in some register, say R0, for future reference. At the moment any zero is received in any row, the value of the nibble coming from the rows is also stored in some another register, say R1. Now to find out that which the key pressed, shift R0 to the right through carry, using RRC instruction, and keep a check on carry, incrementing another variable by 4, every time the RRC instruction is used, until a 0 in the carry appears. At this moment, the variable incremented by 4 every time, would be reading the offset starting from the very first key on keypad to the start of this particular row. Now once the row has been detected and the variable incremented, it is incremented again by the value of the column number, R1 [12]. The ultimate answer we have at the end is the final value of the variable, which is the offset starting from zero, having a maximum possible value of 15 for a 4*4 matrix. Once the offset is obtained, the corresponding ASCII code is selected from the lookup table formulated at the end of the software of master controller. It is selected using the already calculated offset from the starting location of the lookup table. This ASCII code is sent to the data byte of LCD to be displayed, and to other controllers via serial port to be displayed on their SSDs.

IX. INITIALIZING THE LCD 44780

One of the major challenges students face in using LCDs is to initialize them for the very first time [13]. Actually most of the LCDs demand a high level of precision and care in handling, requiring particular amount of time, called the idle time (IT), between two consecutive instructions, which is usually not taken care of, and thus keeps LCDs from being initialized. The initializing instructions include selecting the 2-line or 1 line mode, selecting the font size, options for cursor blinking, cursor position, 8-bit or 4-bit mode etc.

Three LCD pins of the selected LCD panel, namely RS, R/\overline{W} and E are crucial in this regard. The functions of these pins are [14]:

- To send an instruction, RS is always zero, and to send some data instead, it is logic 1.
- For reading data R/\overline{W} is 1 and for writing, it will be 0.
- E is set to 1 as some instruction or data is to be sent, that data or instruction is written into LCD pins, and E is toggled to zero which automatically sends that stuff into the LCD processor.

This is summed up in the chronograph in Fig. 6.

The LCD panel is initialized repeatedly on experimental basis, and it has been observed that if these three pins are properly toggled and corresponding rules are strictly adhered to, one has not to face much difficulty in initializing the panel.

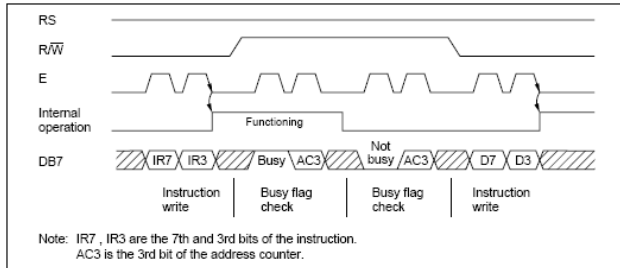


Fig. 6. Chronographs for RS, RW and E

As the LCD is busy, it polls the Busy Flag high, and when free, resets it to 0. Either we have to keep a check on this, or we can wait for maximum amount of time LCD can take to execute an instruction, which is almost 40 (microseconds) at the maximum for this particular model. To simplify the initializing process, if we use a delay of this much duration between any two consecutive instructions, the instruction delay requirements of LCD controller are met successfully, and initialization can be accomplished pretty easily.

X. ROLE OF 485-DIRECTION PINS (\overline{RE} AND DE)

\overline{RE} is 2nd pin and DE is 3rd pin in RS485 pin configuration. These two pins are very important with a

multiprocessor perspective. If these pins of an RS-485 are set, this makes it a line driver, and then it can transmit, but can not receive data. On the other hand, resetting both these pins makes it a receiver, so that it can only receive data and can not transmit it [16]. We have made use of this provision. Each of master and slave-2 has been given access to these pins of all the three controllers.

In master-to-slave mode, these two pins of the addressed slave are reset, so that only it could receive data (sent by master obviously), while those of the master and non-addressed slave are set, leading to master as a sending body, addressed slave as a receiving body and non-addressed slave in idle mode, neither receiving nor transmitting.

Exactly similar situation is there while in slave-to-master and slave-to-slave mode, with the only difference being that here, slave-2 temporarily makes it an “acting master” and actual master as an “acting slave”. In this case, these two pins of slave-2 and the non-addressed controller are set, and those of the addressed body are reset for reasons already described in the upper paragraph.

XI. THE SOFTWARE PROTOCOL

The embedded software for the master microcontroller is organized in such a way so as to first of all initialize the LCD in 2-line and 8-bit mode. All the necessary instructions are sent to initialize it properly, and once it has been done, the display is now ready to display whatever master sends it. At this stage, if the user wants master to slave communication, user, using master keypad, enters “Slave address” (1 or 2), where he intends to transmit data. Otherwise, for slave-to-master and slave-to-slave communication, he enters the address of master or slave 1 respectively, through Slave-2-keypad. In these two different cases, different subroutines are to be followed, as shown in the flowchart diagram in Fig. 7.

In the master-to-slave communication, the signal from master keypad (the slave address entered by user) precedes any serial signal coming from either of the slaves. Once the user enters the slave address, the address is transmitted to and displayed on the LCD and simultaneously, a notification signal is sent serially to the respective slave to be ready to receive data. 485-Direction pins are accordingly set or reset, as described previously. Then on entering data, the LCD displays it in the format “Data= XXXX”. That 4-digit data is simultaneously sent to, and latched into the SSD board of respective slave. The other slave remains idle. Now the master controller ignores any incoming serial signal and is only a transmitting body.

If however, after initialization of LCD, the master first detects a serial signal sent from a slave (obviously slave-2 since slave-1 can not send anything, it can only receive in either case) before receiving a signal from master keypad, this means that user wants a slave to master communication.

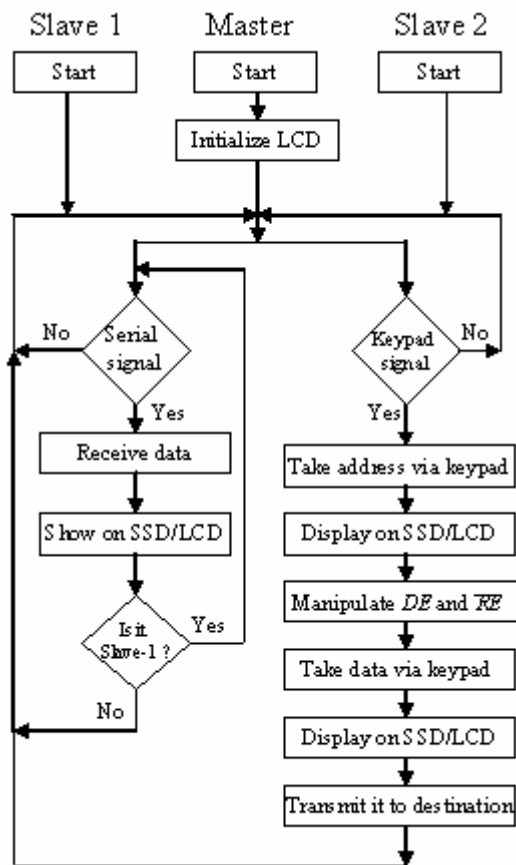


Fig. 7. Combined Flowchart of algorithm

In this case, another subroutine is executed, whereby if master's address is entered by the user, data to be displayed on LCD is received from slave-2 (entered by user through slave-2 keypad) through serial port, rather than from the master keypad.

XII. COMPARISON OF MASTER AND SLAVE-2

It can easily be noticed that the subroutines in master and slave-2 are working opposite to each other. In case of master, if a signal from its keypad is detected before any serial signal from slave-2 (user presses master keypad first, to start master-to-slave mode), its role as a "real master" is ascertained. So it manipulates 485-Direction pins of the three controllers accordingly. Conversely, if slave-2 detects a signal from its keypad before any serial signal from master (user presses slave-2 keypad first, to start slave-2-to-master mode), it starts as an acting-master and converts the actual master to an acting-slave, by sending suitable logic levels to 485-Direction pins of the three controllers. Whatever the case maybe, slave 1 is slave all the time.

XIII. SOFTWARE ASSEMBLER

The software assembler we have used is UVI51. It is suitable for applications like multiprocessor communication,

in addition to having many other handy tools which other such assemblers lack. A few of them are:

- It has an in-built Logic Analyzer, for having a visual check on the data. It provides the user with a better tool to debug errors in coding.
- UVI51 has the ability to simulate upto four microcontroller or microprocessor programs simultaneously, while keeping them synchronized, a unique tool which proves useful in particular situations like we have here.
- It provides the user to simulate different peripherals by having a Graphical User Interface (GUI), as there is an option to open a peripherals-window showing the status of all peripherals visually.
- There is a built-in assembler, and C-compiler.
- Above all, it is absolutely free.

XIV. PROBLEMS FACED AND THEIR SOLUTIONS

The whole assignment was completed successfully, and the devised communication protocol proved fairly reliable. Two problems faced at different stages of the experiment were:

- First, whenever a key was pressed on the keypad, it often led to an unintended repeated transmission of data, e.g. if you pressed the digit 5 on the keypad once, it would appear many times on the LCD. This problem at first caused much confusion, since no end of the keypad terminals was set free and was either connected to the controller pins or pulled up, so such unreliability was perplexing. However, after much a struggle, it became obvious that it was due to the irregularity in the keypad terminal surfaces at microscopic level, which is a common imperfection. So, when a key was pressed once, it would cause corresponding keypad wires to touch and get separated many times, producing the effect as if key were pressed many times. This was a quality problem which might be overcome via debouncing circuitry often used in such devices.
- Second, it was defined as a requirement that the system should be able to operate in master-to-slave and slave-to-master modes endlessly without reset, but it proved difficult to implement in software. It was due to the fact that if, for instance, the user first operated the system in master-to-slave-2 mode, the 485-Direction pins of master and slave-1 were set, and those of slave-2 were reset by the master itself, and the communication could be completed in that way successfully, but it was a one-time process. After that, master still remained driver and slave-2 remained receiver, in spite of the user now first entering via slave-2 keypad, since 485-direction bits were not changed by software at the end of first mode. This problem only arose while communicating between master and slave-2, since there was no such issue involved in the all-time-slave slave-1. To counter this problem through software was much challenging. However there was a

much simpler hardware solution to this. A three-terminal button was added to the hardware to get the option from the user as to what type of communication he desired. Then 485-direction pins of all the three controllers were manipulated through hardware as shown in the table 1, and the result was a perfect communicating system.

TABLE 1
485-DIRECTION PINS' REQUIRED STATUS FOR DIFFERENT COMMUNICATION MODES

Mode	Master	Slave-2	Slave-1
Master to slave-2	1	0	1
Slave-2 to Master	0	1	1
X to Slave-1	1	1	0

XV. CONCLUSION

Multiprocessor communications are an integral part of modern industrial environments as today communications are of much more important than ever. The hardware developed is a simplified prototype of the communication models used in industrial environments, with master controller representing the control room from where instructions and commands are sent to different sections of the industrial setting, and slaves mimicking the role of locations where these commands are sent to be executed. More often than not, such destinations are also capable to send latest data to the control room (e.g. sensors), as was the

case with slave-2. The communication protocol used is a very simplified one, but gives insight into how such communications are accomplished in high-tech industrial environments. Moreover, RS-485 is perfect for transferring small blocks of information over long distances, and the RS-485 standard is found to be extremely flexible.

REFERENCES

- [1]. Nunnally, C.E., "Teaching Microcontrollers," Proc. of the 26th Frontiers in Education Annual Conference, Vol. 1, Nov. 1996, pp. 434-436.
- [2]. Rodriguez Andina, J.J., del Río, A., "Aplicación de un simulador al desarrollo de prácticas con microcontroladores," Proc. of the 2nd Conference on Tecnologías Aplicadas a la Enseñanza de la Electrónica, Vol. III, Sept. 1996, pp. 42-46 (in Spanish).
- [3]. Avocet Systems, AVS High-Level simulator/ Debugger, <http://www.avocetsystems.com/>.
- [4]. Keil Software, dScope Debugger, <http://www.keil.com/>.
- [5]. Virtual Micro Design, Universal microprocessor Program Simulator (UMPS), <http://www.vmdesign.com/>.
- [6]. Alfredo del Río and Juan José Rodríguez Andina, UVI51: a simulation tool for teaching/learning the 8051 Microcontroller, 30th ASEE/IEEE Frontiers in Education Conference, Kansas City, 2000.
- [7]. J. Axelson, "Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks", Lakeview Research, Madison, WI, 1998.
- [8]. Circuit Cellar INK, the Computer Applications Journal, Issue 107, June 1999.
- [9]. www.pjrc.com/tech/8051/autobaud.html.
- [10]. http://www.pccompci.com/The_RS485_Serial_port.html
- [11]. [www.yahoosearch.com/PIC Tutorial Nine, HEX Keypad](http://www.yahoosearch.com/PIC_Tutorial_Nine_HEX_Keypad)
- [12]. Scott Mckenzie, "The 8051 Microcontroller", Third edition, pp 218.
- [13]. [www.google.com/ LCD interfacing reference page](http://www.google.com/LCD_interfacing_reference_page).
- [14]. [www.8052.com/Tutorial: Introduction to LCD Programming](http://www.8052.com/Tutorial:_Introduction_to_LCD_Programming)
- [15]. Hitachi HD44780 datasheet available at <http://semiconductor.hitachi.com/>
- [16]. RS485 datasheet available at <http://www.maxim-ic.com>