# MICROCONTROLLER APPLICATIONS TEACHING A PRACTICAL APPROACH

*Clóvis Fischer and José Carlos de Souza Jr.*
*Department of Electrical Engineering*
*Esc. de Engenharia Mauá, São Caetano do Sul - SP, Brazil 09580-900*
*Universidade São Judas Tadeu, São Paulo - SP, Brazil 03169-040*

***Abstract -*** *There's a broad field of applications for the microcontroller technology, that can be enumerated from simple, like small electric equipment operation, until sophisticated, like single loop digital controllers with man-machine and machine-machine interface, also using complex digital control techniques. The main preoccupation for us is to transmit to an undergraduate student, who hadn't had any contact with this field, hardware and software concepts, in an integrated and step by step form, to develop and document a project using the microcontroller technology. Among the various microcontroller families existing in the market, we choose the 8051 family, because of its popularity, but the techniques can be migrated to another microcontroller family, with small routine modifications. The hardware modules were constructed with low cost devices and with two connectors types, to allow fast connection between the modules, project integration and flexibility.*

## I. INTRODUCTION.

When teaching microprocessor applications in an undergraduate course, we detected a lack of teaching material that deals with real time and real software location in memory, because the only available resources were a microprocessor based board, with a monitor program, keyboard and seven segment display, where the instructions would be entered by typing on the keyboard their hexadecimal code. This was a very tedious task, especially when the student discovered that there was a mistake in the beginning of his hundred bytes code.

The solution taken was the construction of a low cost PC based EEPROM EMULATOR, the acquisition of a low cost PC ASSEMBLER - LINKER, and the construction of a new board, now focusing the microcontrollers, because of their simplicity and functionality. We used the so popular 8051 family.

The EEPROM EMULATOR has 8 kbytes of size, which is sufficient for all teaching and more elaborated projects.

The MICROCONTROLLER BOARD has only the microcontroller socket, with Port 1, Port 3, Port 0- Address/Data Bus, and Port 2-Address Bus connected to dual connectors (one hole type and the other, pin type), the latch for address demultiplexing, one RAM and one EEPROM soquets connected to the low microcontroller addresses and two small 16 pin dip soquets for general propose logic connections. The dual connectors allows fast connections between the microconttroller board and the other modules.

With this equipment, we can write the program in a PC based text editor, in assembler level language (or C language if disposable), generate the code for the microcontroller, down-load the code in the EEPROM EMULATOR, that was connected to the MICROCONTROLLER BOARD and verify if the program is running correctly, by means of signal inspection through the microcontroller I/O ports, with another hardware module connected to the I/O ports or an oscilloscope if needed. This make possible the real software location, with respect to the RESET and interruption routines.

The most important aspects of this approach are the low cost, easy to construct and the flexibility achieved, that's in accordance with our goals.

## II. THE FIRST STEP: BASIC I/O WITH PORTS.

The starting work with the teaching material is to execute a simple program that transfers the bits presented at the Port 3 to the Port 1, using a 8 LEDS - 8 KEYS I/O MODULE, as shown in Fig. 1.

The program listing is the following:

```
        ORG     0000H
        RESET
        JMP     START

        ORG     0030H

START
        MOV     A, P1    ;  reads from the 8 KEYS
                         ; and transfer from Port 1 to
                         ; Accumulator

        MOV      P3, A   ; and from it to Port 3,
                         ; writing on the 8 LEDS
```

```
        JMP     START

        .END
```

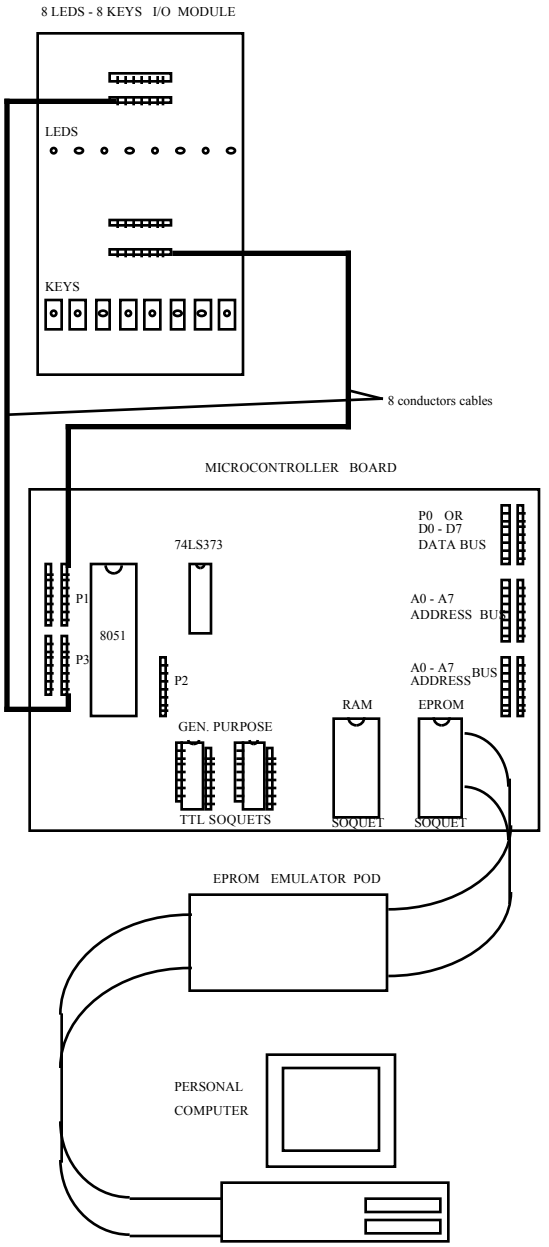



Fig. 1. MICROCONTROLLER BOARD connected to the EEPROM EMULATOR and the 8 LEDS - 8 KEYS I/O MODULE

This program performs the content transfer from Port 1 ( microcontroler 8 pins parallel I/O, that is connected to the 8 KEYS as an input driver) to the accumulator, and from the accumulator to Port 3 (microcontroller 8 pins parallel I/O, connected to the 8 LEDS panel, as an output driver).

At this step, the student learns what is a label, an assembler directive (ORG and .END) and its difference with respect to a microcontroller instruction (JMP and MOV), learning also what is a MOVement instruction, an unconditional JuMP instruction, and its mnemonics.

Another subject of attention is the code generation steps, that comprises the Source Code text editing, the assembler phase and the linking phase. The final binary file, that is chosen in an ASCII format is also analyzed by the student.

## III. EXPLORING THE ARITHMETIC AND LOGIC INSTRUCTIONS

To explore the arithmetic and logic instructions, the student is asked for implementing a modification on his first program, that breaks the original byte, that came from the 8 KEYS through Port 1, in two registers located in the internal microcontroller RAM. The first register must store the 4 least significant bits, and the second register must store the 4 most significant bits, shifted to the right four times.

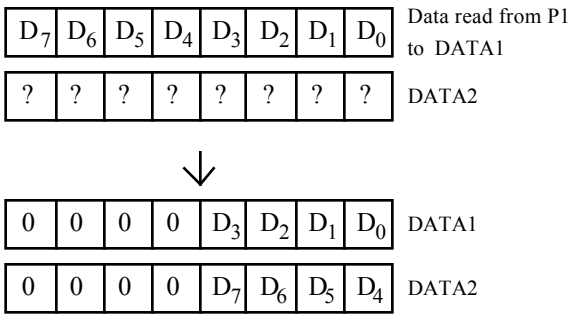



Fig. 2. Register contents before and after program execution

The program listing is the following:

```
DATA1       .REG    30H
DATA2       .REG    31H

            ORG     0000H

            RESET

            JMP     START

            ORG     0030H

START
```

```
        MOV    DATA1, P1      ; read from the 8
                              ; KEYS ( Input )

        MOV    A, DATA1
        ANL    A, #F0H

        RR     A
        RR     A
        RR     A
        RR     A


        MOV    DATA2, A

        MOV    A, DATA1

        ANL    A, #0FH
        MOV    DATA1, A
```

| "ARITHMETIC - LOGIC INSTRUCTIONS" |
|---|

```
        ; at this point, the student can implement
        ; his arithmetic and logic instructions between
        ; DATA1 and DATA2, through the
        ; Accumulator, leaving the results on the
        ; Accumulator

        MOV    P3, A          ; write to the 8
                              ; LEDS ( Output )

        JMP    START

        .END
```

With this modifications, the student learned the "RR A" (Rotate Accumulator Right), the "ANL A, #data" ( logic and immediate byte with the accumulator) instructions working as a masking instruction, and how to specify an internal RAM memory position as a register (DATA1 and DATA2). Other instructions, like "RL A" (Rotate Accumulator Left) and the "SWAP" instructions, can also be explored.

With another small instruction modifications in the "ARITHMETIC - LOGIC INSTRUCTIONS" source code area, he can explore the all ANL (Logical And), "ORL" (Logical Or), "XRL" (Logical Ex-Or) and "CPL A" (Complement Accumulator) instructions performed by the 8051, and show its results in the 8 LEDS panel, in a very interactive way.

The Arithmetic instructions, like "ADD" (Add to Accumulator ), "ADDC" (Add to accumulator with carry), "SUBB" (Subtract from Accumulator with borrow), "INC" (Increment), "DEC" (Decrement), "MUL" (Multiply) and "DIV" (Divide) can also be explored and its operation learned with this program and hardware modules.

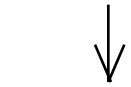
| "ARITHMETIC - LOGIC INSTRUCTIONS" |
|---|

```
        MOV    A, DATA1
        ADD    A, DATA2
```

Another points to explore are the Subroutines and Macros.

The subroutines are explained substituting the code portion that deals with the "ARITHMETIC - LOGIC" block by a "CALL label" instruction, and creating a subroutine that performs the same task as before. The result must be the same, if the program was correctly written by the student.

| "ARITHMETIC - LOGIC INSTRUCTIONS" |
|---|



```
        CALL   ADDITION
```

The other program part becomes:

```
        MOV    P3, A

        JMP    START
- - - - - - - - - - - - - - - - -
ADDITION

        MOV    A, DATA1
        ADD    A, DATA2

        RET
- - - - - - - - - - - - - - - - -
        .END
```

subroutine

The Macro, if disposable by the Assembler-Linker software, can also be explained in a similar way, only pointing out the differences between it and the subroutine. Macro is faster than subroutine because it hasn't the "CALL label" - "RET" instruction pair, but expends more program memory. A single macro is expanded with its instructions each time it's used.

If it's desirable to show an increment/decrement action, and the microcontroller is so fast to observe its counting with the 8 LEDS panel, we can write a delay subroutine and include it in the main program.

Other details have to be emphasized, like:

- the various addressing modes (immediate, register, direct and indirect);
- the stack operation and the Stack Pointer Register in subroutine calls;
- the flags and the Conditional Jumps, including more elaborated loop instructions ( CJNE - Compare and Jump if Not Equal, and DJNZ - Decrement and Jump if Not Zero)
- the internal memory arrangement (general purpose RAM and Special Function Registers).

All this resources can be explored in a very easy, fast and interactive way with small programs, a little bit of imagination and creativity, developing this method of work on the student.

## IV. EXPLORING THE BIT ORIENTED INSTRUC-TIONS

This kind of instructions, very powerful and very common in microcontoller families, are explored in an application way: a Logic Controller. Three bits (P1.0, P1.1 and P1.2 in connection with 3 of the 8 KEYS) are used as the input of the Logic Controller and three bits (P3.0, P3.1 and P3.2 in connection with 3 of the 8 LEDS panel) are used as the output signals. A State Machine, specified by a Ladder Diagram is implemented using bit movement and bit manipulation logic instructions (MOV C,bit; ANL C,bit and ORL C,bit for example) as shown in Fig. 3.

The RL1, RL2 and RL3 labels are internal bit registers, which are located in the bit addressable area of the internal RAM. These bits can be set (logic level 1) or reset (logic level 0) by means of the "SETB bit" and "CLR bit" instructions respectively.

At this point, the student learned almost all 8051 instructions and internal memory organization.

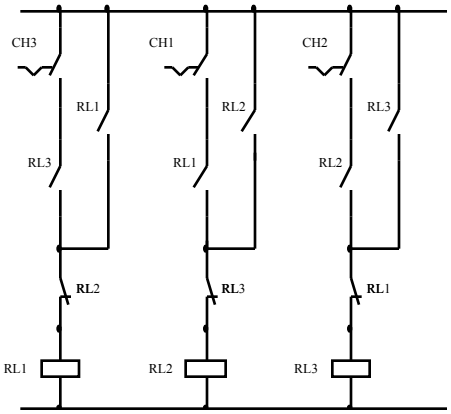



Fig. 3 The Logic Controller Ladder Diagram.

## V. THE INTERRUPTIONS

The interrupts are very important when asynchronous interface between the high speed microcontroller and low speed real world is required.

Its hardware interface and software can be learned using an external adjustable clock generator (a function generator or a small 555 oscillator for example) with TTL compatible output and variable frequency. This signal must be applied to an interrupt input (P3.2 for INT0 or P3.3 for INT1), the application program written, taking care to enable the selected external interruption source . As a suggestion, we can implement an increment counter with the accumulator, that increments each time one interrupt is recognized, and the accumulator value showed by means of a "MOV P1, A" instruction, that show its contents on the 8 LEDS panel.

Care must be taken when dealing with interrupts, principally with respect to the stack growing. "PUSH" and "POP" instructions are explained and how to use them.

## VI. PRECISION TIMING: THE TIMERS

A very important hardware resource is the Timer/Counter. The Timer 0 is programmed as a timer with auto reload and operate in interrupt mode, to temporize 10 ms. This timing is used to increment a counter (one position of internal RAM, labeled COUNT_1S) from 1 to 100, resulting in a total timing of 1 second.

The student can observe the 10 ms timing using an oscilloscope and analyzing the signal of one port pin (P3.7 for example), toggling this pin with the instruction "CPL P3.7" each time the 10 ms timer elapses. This can be done putting this instruction in the timer service routine. The 1 s timing can be observed using the same method, but moving the "CPL P3.7" instruction from the timer interrupt service routine to the COUNT_1S increment routine, and applying the P3.7 signal to one led of the 8 LEDS panel.

## VII. 16 KEYS MATRICIAL KEYBOARD INTERFACE APPLICATION

The next application focuses the keyboard data input. This topic is exercised with the MATRICIAL KEYBOARD MODULE, connected to the PORT 1, as shown in Fig. 4.

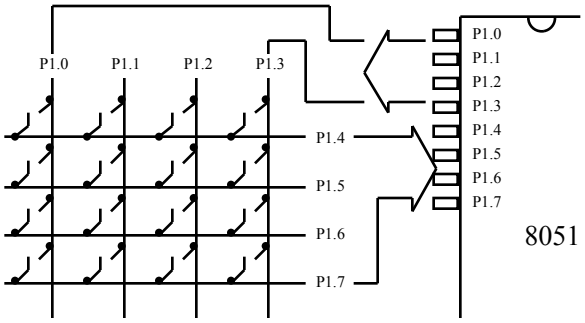



Fig. 4. MATRICIAL KEYBOARD MODULE connected to 8051 PORT 1

The software for this application does a column scan with zero level from P1.0 to P1.3 pins of PORT1 and detects which key was stroke checking in which row (P1.4 to P1.7 of PORT 1) the zero level appear. This scan is done using "SETB bit" and "CLR bit" instructions. A set of 16 flag bits, in the bit addressable internal RAM region, are used to store the information of which key was pressed, and can be used by another software module, allowing the exploration of software modularity and the utilization of variables as interfaces for software modules.

The bouncing problem that arises, when contact keys are used, can also be explored. A software counter can be implemented between the scans, and the counter variable can be modified to match the debouncing requirements.

To show the result of the keyboard application , an ASCII code (hexadecimal code 0, 1, 2, 3, ... , 9, A, B, C, D, E, F, for example) can be assigned to each key, and showed at the 8 LEDS panel, which is connected to PORT 3. At each time a key is pressed, the assigned code appear on the 8 LED panel.

The assembly for this task is shown in Fig. 5.

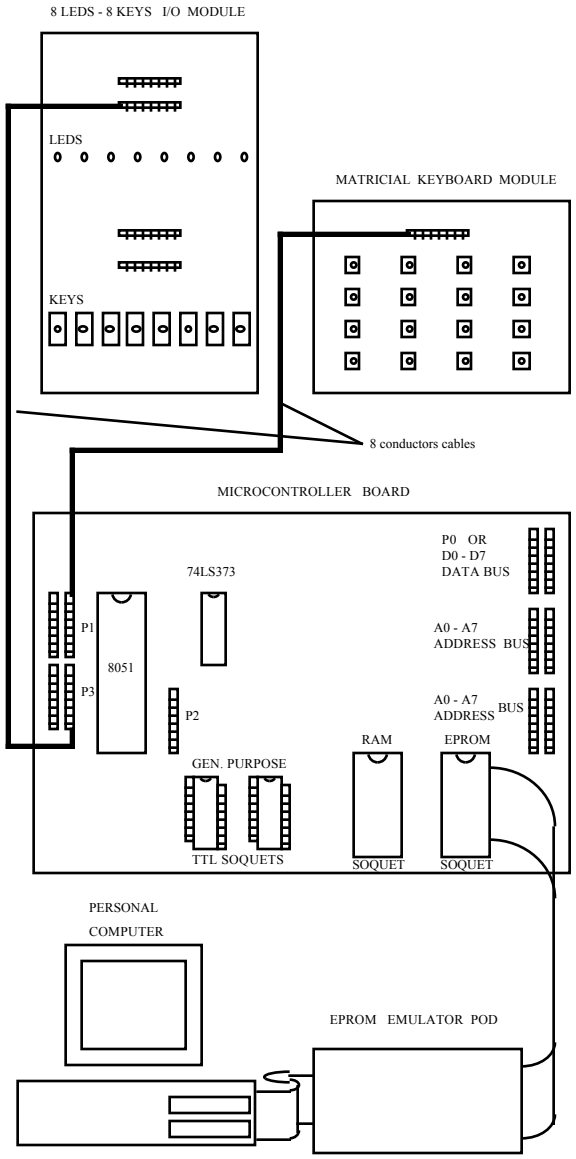



Fig. 5. Module interconnections to develop the keyboard application

## VIII. SERIAL COMMUNICATION APPLICATION

The suggestion now is to transmit the key pressed code to another computer by means of a serial communication channel.

First, all the concepts of serial communication are presented to the students, like the RS-232 standard, the DB9 and DB25 connectors pins and signal assignments, the start bit - stop bit, the parity bit, the complete signal wave form and the signal voltage level adaptation using commercial buffers.

Then, we connect the signals of the SERIAL COMMUNICATION MODULE (which contain the buffers and a DB25 connector) to the MICROCONTROLLER BOARD pins that are used to implement serial communication ( RXD/P3.0 and TXD/P3.1 from PORT 3).

The software is written as an upgrading of the 16 KEYBOARD application software, inserting the serial initialization and service routine, that transmit the key pressed to the serial channel.

This application needs another microcomputer, operating as a terminal, to receive data, and must use any commercial serial communication program. The complete modules and computers arrangement is shown in fig. 6.

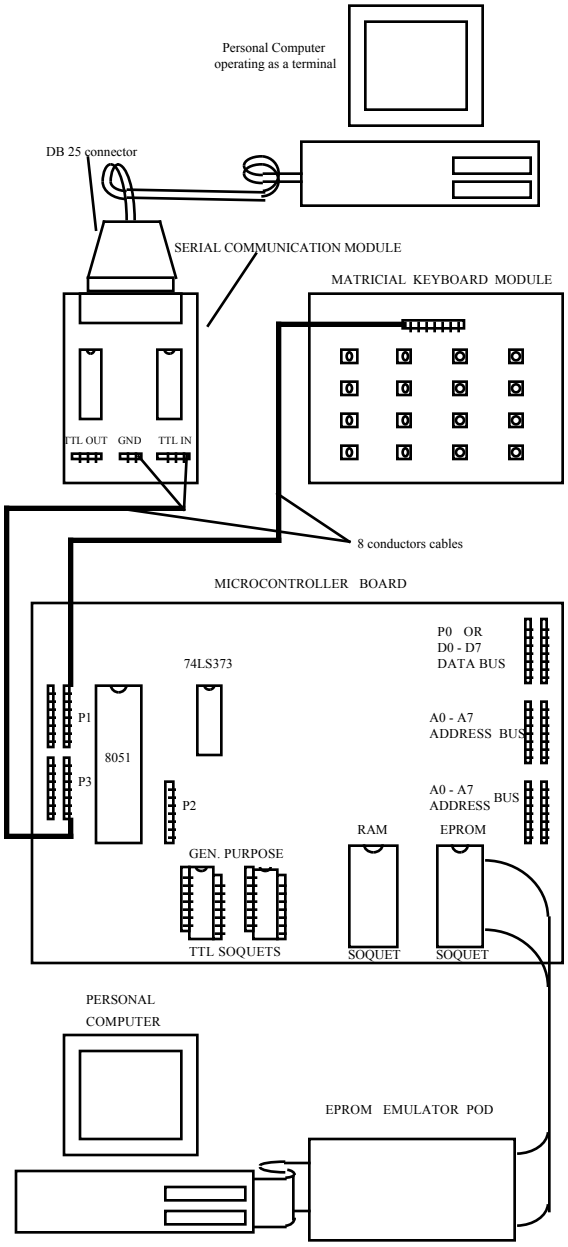



Fig. 6. Arrangement to implement Serial communication application.

Another possibility in this work, is to analyze the serial channel operation as a function of the crystal frequency, and the software modifications need.

## IX. LIQUID CRYSTAL DISPLAY APPLICATION

This very important application is explored connecting a LIQUID CRYSTAL DISPLAY MODULE (LCD MODULE), through the MICROCONTROLER BOARD data bus (D0 - D7) and address bus (A0 - A15) demultiplexed signals. The complete arrangement to implement this work is shown in Fig. 7.

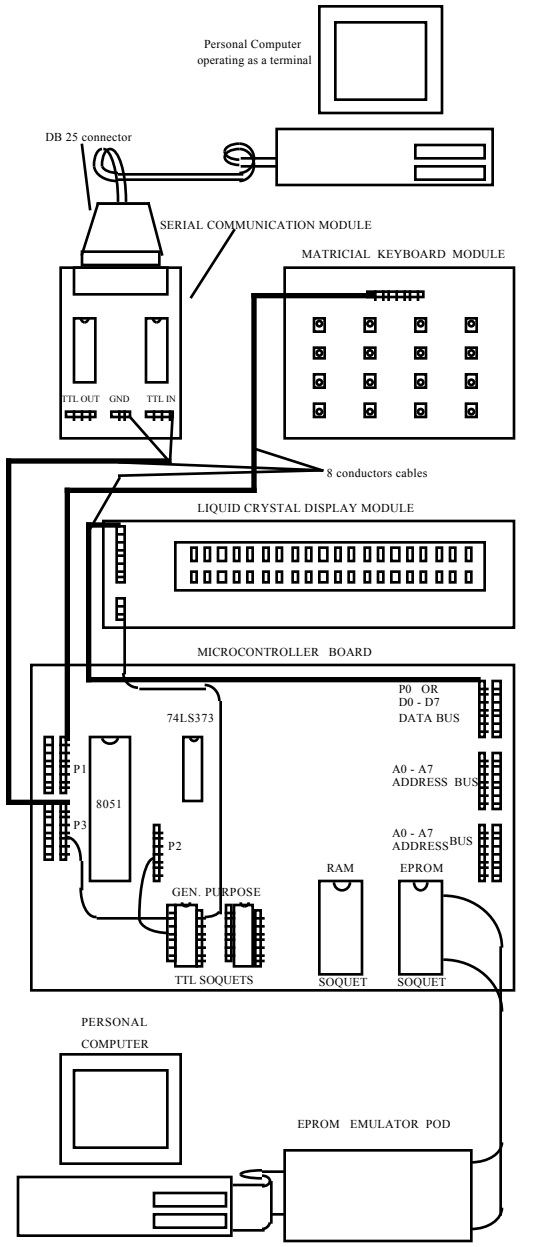



Fig. 7. Arrangement to work with the LCD.

The LCD MODULE is composed of a standard commercial 2 lines x 20 columns LCD, a TTL inverter and connection pins.

The first program simply executes the display initialization and performs a simple character transfer to the display.

The next application explores the LCD commands, like cursor positioning, left and right character scroll and cursor format.

Then, an internal RAM character buffer is transferred to the display, and the proposition is to use indirect memory addressing mode instructions. The student makes the first contact with the pointer concept applied to microcontrollers.

Finally, the last application with LCD uses the keyboard driver software to implement a data logger system, that receives and treat all the keys pressed, show them in the display, stores them in the internal RAM, working as a data transmitter buffer, and transmit these characters to the host PC using the serial channel. A character transfer from the host PC to the LCD can also be implemented, where the received caritas are stored in an internal RAM receiver buffer, and then, when the transmission is finished, the received caricatures are shown in the LCD.

Another aspect of a more complex system design that is important to be well specified is the communication protocol between the microcontroller based equipment and the host computer. This subject is also covered, and a personal protocol is implemented in the software. Some other variations of the protocol are explained and suggested as future work.

## X. D/A AND A/D CONVERTION

The D/A AND A/D CONVERTER MODULE, used in this step, was implemented with the DAC800 and AD0800 commercial devices, with a octal latch and some operational amplifiers to perform high input impedance, signal level and off-set adjustment.

For the initial D/A application, is suggested to the student to implement a simple triangular and saw-tooth wave form generator, the analog output being observed with an oscilloscope, as shown in Fig. 8.

For the initial A/D application, the suggestion is to implement a voltmeter, operating in a free running mode, where the conversion rate can be adjusted by software. The conversion result is displayed on the LCD, or can be transmitted to the host PC, using the serial channel learned before, as an example of a telemetric system.

## XI. NUMERICAL METHODS

Simple numeric routines are explained. This routines include multi-precision addition, subtraction, multiplication and division. Signaled numbers and fixed point are also considered. These routines are implemented and run, with the results checked using the LCD as an multiple digit output.

The approximation of integral and derivative operations are also implemented.

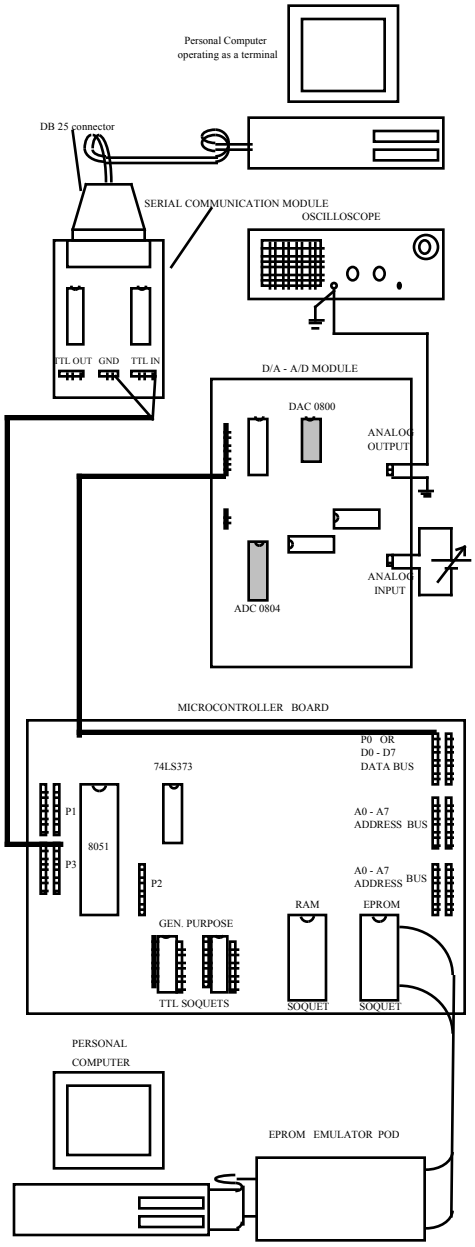



Fig. 8. The D/A and A/D application.

## XII. DIGITAL CONTROL APPLICATION

With the concepts developed in the D/A AND A/D CONVERTION section and NUMERICAL METHODS section, the student is apt to develop a simple control application. The suggestion is to control a motor shaft speed, using a servomechanism laboratory set, applying the tachometer signal to a FREQUENCY TO VOLTAGE CONVERTER, which output is applied to the A/D channel input, processing of this signal by means of a control algorithm, and the resulting action value written to the D/A latch. The D/A output is applied to a PUSH-PULL AMPLIFIER, supplying the controlled voltage

to the DC MOTOR. The control algorithm used is the Digital Proportional-Integral-Derivative (PID), where the transfer function is translated from s domain to z domain using the bilinear approximation .

To drive the motor, it's possible to use a push-pull amplifier module between the D/A output and the motor power connectors, or it can be implemented using the PWM technique, with the PWM BRIDGE POWER MODULE

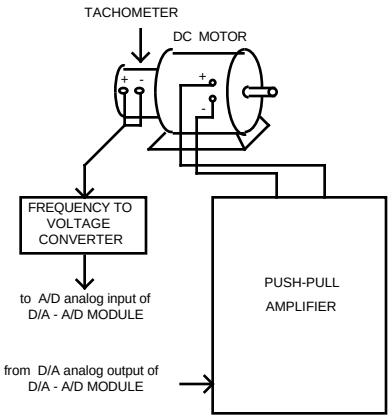



Fig. 9. Digital control application assembly.

## XIII. ADVANCED APPLICATION

As a final subject, software structures using state machine approach, with state variables and a case structure, is presented.

This technique can be used to solve more complex problems, like systems with keyboard configuration, and systems with many modes of operation.

The configuration storing may be implemented with serial electrically erasable memories (EEPROM). A set of software routines to drive this devices is presented and discussed. This routines can be linked in a more complex software, and called as subroutines, in conjunction with address and data variables previously declared.

## XIII. CONCLUSIONS

The presented approach is a result of four years of microcontroller teaching at an undergraduate engineering course and the summing of a day by day experience and industrial needs. The effect on the student staff is a very satisfactory response during the course.

An evaluation was applied at the end of the course, using simple questions to evaluate the reactions of the students and the results are showed on the following table:

TABLE 1

| | Score | | | | | |
|---|---|---|---|---|---|---|
| | 5 | 4 | 3 | 2 | 1 | 0 |
| Course Importance | 100 % | - | - | - | - | - |
| Interest on the Subject | 80 % | 20 % | - | - | - | - |
| Course Coverage | 80 % | 20 % | - | - | - | - |
| Method of Approach | 80 % | 20 % | - | - | - | - |
| Equipment Used | 100 % | - | - | - | - | - |

Another positive observation made by the students was the ability to understand any other microcontroller family, developed during the training, because this approach cover many microcontroller details witch are common to other microcontrollers, and the principal subjects are the real applications.

Some students showed his personal and immediate interest because they're working directly with this technology.

They also approved the progressively increase of difficulty and complexity used, and the integration caracteristic of the course. We noted a great satisfaction by the students at the end of our work, when they came with problem's solution that uses not only the software/hardware ideas, but also the philosophy of development learned.

## ACKNOWLEDGMENT

The authors thank Mr. Artur Selmikaites, Dr. Antônio. O. M. Andrade and the Depto. de Eng. Elétrica da Escola de Engenharia Mauá and the undergraduate electrical engineering students, who made possible this project.

## NOTE

All power supply equipment and connections was suppressed.

## REFERENCES

[1] _____ "8-Bits Embedded Controllers", Intel Corporation, 1990.

[2] S. Yeralan and A. Ahluwalia, *Programming and Interfacing the 8051 Microcontroller*. Reading, MA : Addison Wesley Publishing Company, 1995.

[3] D. V. Hall, *Microprocessors and Interfacing : Programming and Hardware, 2nd ed.* Singapure: McGraw-Hill, 1992

[4] D. Morgan, *Numerical Methods / Real-Time and Embedded Systems Programming.* San Mateo: M&T Books, 1992

[5] K. J. èstrîm, B. Wittenmark, *Computer-Controlled Systems.* Upper Saddle River: Prentice-Hall, Inc. 1997.