

**Embedded System Design
Using 8031 Microcontrollers**

FRONTLINE
ELECTRONICS

Embedded System Design Using 8031 Microcontrollers

It is my great pleasure in introducing this eBook to your eyes. It covers many steps that should go into an Embedded System Design.

Frontline Electronics has about 15 years of experience using 8031 microcontrollers and I was happy to be with 8031 in all these years.

The contents of this eBook has been derived from our team's vast hands-on experience.

Along the way, you may find the discussion about Frontline Electronics' software tools. I am sure that these discussions should give you many useful information that will help you in finishing your target design fast.

As an author, I am eagerly looking forward to get your feedback on this eBook. Any suggestion is most welcome. If I get enough feedback, I may include the source code for all the projects in the next edition.

You can freely distribute this eBook to the fellow designers or any one interested in embedded electronics.

Welcome to the Embedded world...!!!

Balaji

Technical Director

Frontline Electronics

India.

Date : 05.09.2002

Email : Balaji@frontlinemail.com

CopyRight : Frontline Electronics Private Limited, India, 2002.

FRONTLINE
ELECTRONICS

Contents

Chapter 1: Introduction To Embedded Systems

Introduction	2
--------------------	---

Chapter 2: 8031 Microcontrollers

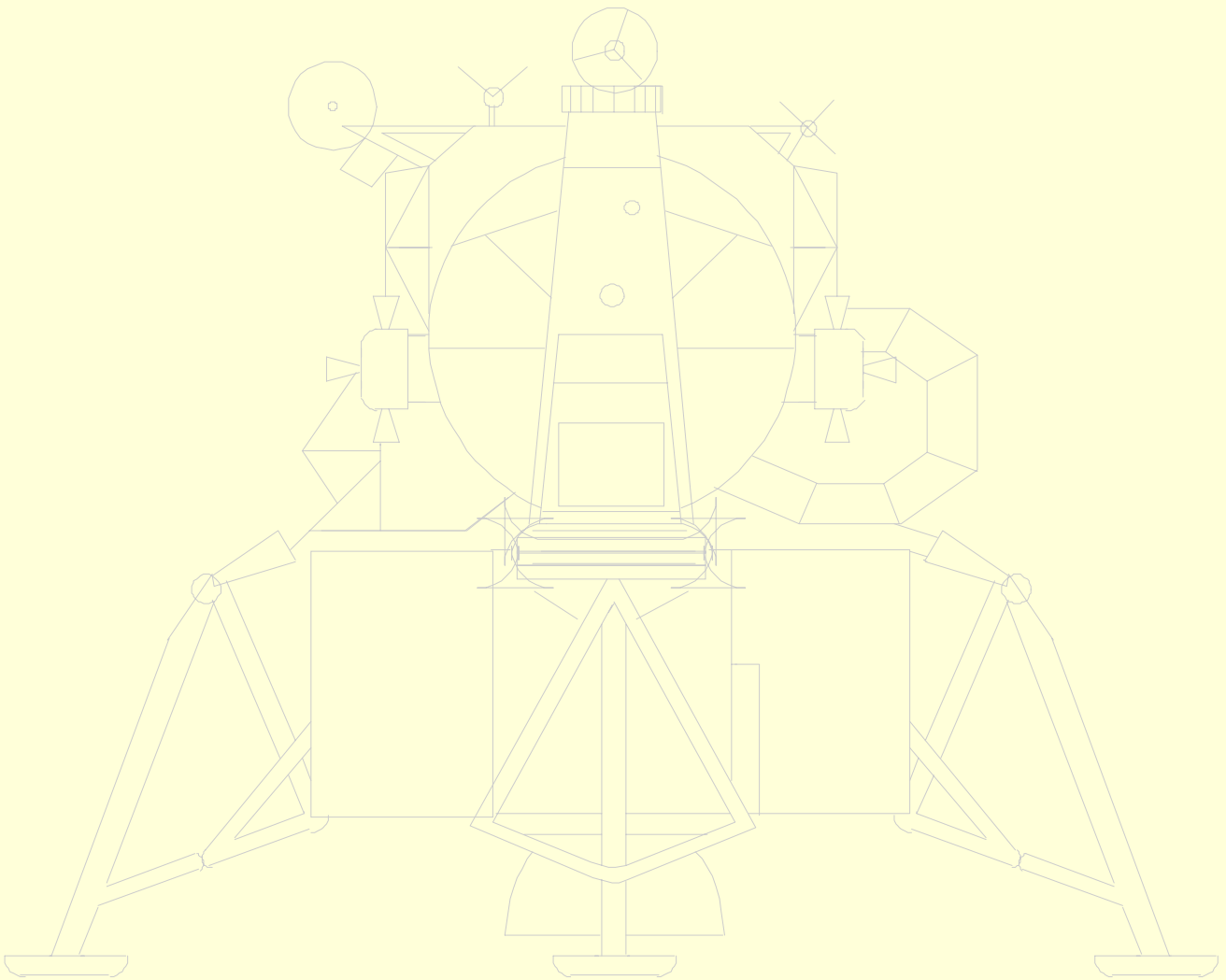
2.1 Intel's 8031 Architecture	4
2.2 Central Processing Unit	6
2.3 Input / Output Ports	7
2.4 Timers / Counters	8
2.5 Serial Port	8
2.6 Memory Organization	9
2.7 Common Memory Space	12
2.8 Interrupts	12
2.9 Addressing Modes	
2.9.1 Register Addressing	13
2.9.2 Direct Addressing	13
2.9.3 Register Indirect Addressing	14
2.9.4 Immediate Addressing	14
2.9.5 Index Addressing	14
2.10 Instruction Set	
2.10.1 Data Transfer Instructions	15
2.10.2 Data Transfer In External RAM	16
2.10.3 Lookup Tables	17
2.10.4 Arithmetic Instructions	18
2.10.5 Logical Instructions	19
2.10.6 Program Control - Jumps, Calls, and Returns	20
2.10.7 Jump Instructions	20
2.10.8 Conditional Jump Instructions	22
2.10.9 Operate and Branch Instructions	23
2.10.10 Boolean Instructions	24

Chapter 3:	8031 Derivatives	
3.1	8031 Derivatives	27
3.2	Why Atmel Devices?	29
Chapter 4:	Real Life Projects	
4.1	16KHz Monitor for the Public Call Office	31
4.2	Telephone Line Interface	32
4.3	16KHz Metering Pulse Detection	33
4.4	Stepper Motor Controller	36
4.5	Programmable Timer	39
4.6	8 Channel Data Acquisition System	43
4.7	8 Channel Sequential Controller	46
4.8	Frequency Counter	50
Chapter 5:	Project Tools	
5.1	Software Options	53
5.2	Stand Alone Device Assemblers	54
5.3	Stand Alone Remote Debugger	54
5.4	Standalone Simulators	55
5.5	In Circuit Emulators	56
Chapter 6:	Topview Simulator	
6.1	Introduction	58
6.2	Device Selection	60
6.3	Program Editing	62
6.4	Clearview	63
6.5	Program Execution	65
6.6	Simulation Facilities	67
6.6.1	LED Modules	69
6.6.1.1	Plain Point LEDs	69
6.6.1.2	Seven Segment Displays	70
6.6.2	LCD Module	71

6.6.3	Keyboard Module	72
6.6.4	I ² C Module	74
6.6.5	SPI Modules	75
6.7	Code Generation Facilities	75
6.7.1	Internal Peripheral Functions	76
6.7.1.1	Serial Port	76
6.7.2	External Peripheral Modules	76
6.7.2.1	LED Display Functions	76
6.7.2.2	LCD Module Selection	77
6.7.2.3	Keyboard Interfacing	78
6.7.2.4	I ² C / SPI Buses	80
Chapter 7:	Topview Debugger	
7.1	Introduction	82
Chapter 8:	Topview Programmer	
8.1	Introduction	88
8.2	GUI Features	90
Chapter 9:	Contact Frontline Electronics	
9.1	Contact Frontline Electronics	93

Chapter 1 - Introduction To Embedded Systems

- Introduction



Introduction

We are living in the Embedded World. You are surrounded with many embedded products and your daily life largely depends on the proper functioning of these gadgets. Television, Radio, CD player of your living room, Washing Machine or Microwave Oven in your kitchen, Card readers, Access Controllers, Palm devices of your work space enable you to do many of your tasks very effectively. Apart from all these, many controllers embedded in your car take care of car operations between the bumpers and most of the times you tend to ignore all these controllers.

In recent days, you are showered with variety of information about these embedded controllers in many places. All kinds of magazines and journals regularly dish out details about latest technologies, new devices, fast applications which make you believe that your basic survival is controlled by these embedded products. Now you can agree to the fact that these embedded products have successfully invaded into our world. You must be wondering about these embedded controllers or systems. What is this Embedded System?

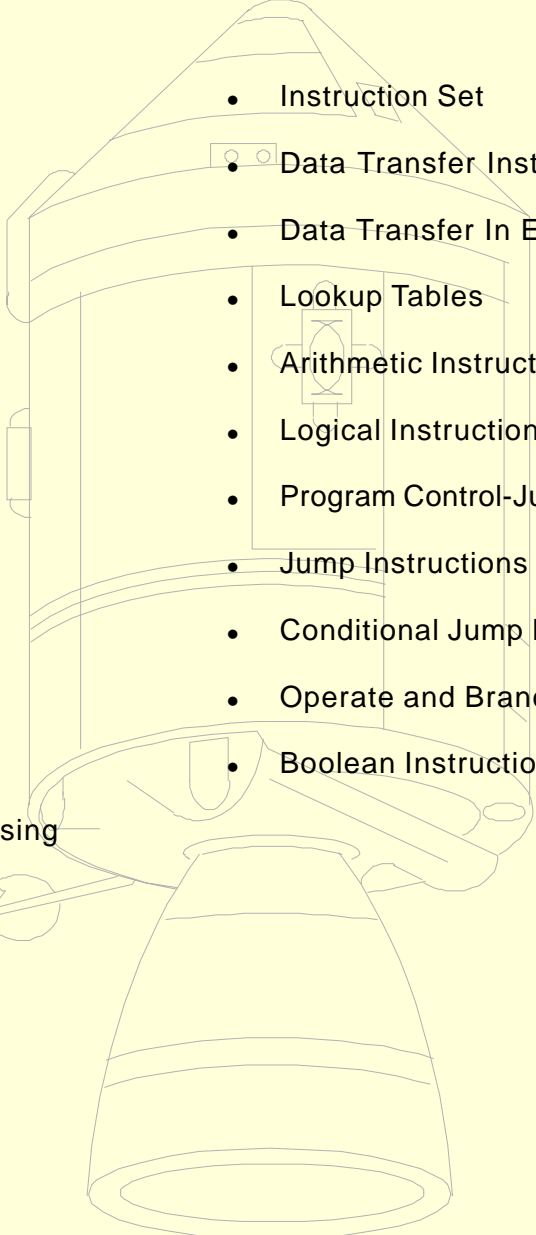
The computer you use to compose your mails, or create a document or analyze the database is known as the standard desktop computer. These desktop computers are manufactured to serve many purposes and applications.

You need to install the relevant software to get the required processing facility. So, these desktop computers can do many things. In contrast, embedded controllers carryout a specific work for which they are designed. Most of the time, engineers design these embedded controllers with a specific goal in mind. So these controllers cannot be used in any other place.

Theoretically, an embedded controller is a combination of a piece of microprocessor based hardware and the suitable software to undertake a specific task.

These days designers have many choices in microprocessors/microcontrollers. Especially, in 8 bit and 32 bit, the available variety really may overwhelm even an experienced designer. Selecting a right microprocessor may turn out as a most difficult first step and it is getting complicated as new devices continue to pop-up very often.

In the 8 bit segment, the most popular and used architecture is Intel's 8031. Market acceptance of this particular family has driven many semiconductor manufacturers to develop something new based on this particular architecture. You can find this 8031 in a variety of configurations, flavour and definitely you cannot see this kind of variations with any other architecture. Even after 25 years of existence, semiconductor manufacturers still come out with some kind of device using this 8031 core.

- Intel's 8031 Architecture
 - Central Processing Unit
 - Input / Output Ports
 - Timers / Counters
 - Serial Port
 - Memory Organization
 - Common Memory Space
 - Interrupts
 - Addressing Modes
 - Register Addressing
 - Direct Addressing
 - Register Indirect Addressing
 - Immediate Addressing
 - Index Addressing
- 
- Instruction Set
 - Data Transfer Instructions
 - Data Transfer In External RAM
 - Lookup Tables
 - Arithmetic Instructions
 - Logical Instructions
 - Program Control-Jumps, Calls and Returns
 - Jump Instructions
 - Conditional Jump Instructions
 - Operate and Branch Instructions
 - Boolean Instructions

2.1 - Intel's 8031 Architecture

The generic 8031 architecture sports a Harvard architecture, which contains two separate buses for both program and data. So, it has two distinctive memory spaces of 64K X 8 size for both program and data. It is based on an 8 bit central processing unit with an 8 bit Accumulator and another 8 bit B register as main processing blocks. Other portions of the architecture include few 8 bit and 16 bit registers and 8 bit memory locations.

Each 8031 device has some amount of data RAM built in the device for internal processing. This area is used for stack operations and temporary storage of data.

This base architecture is supported with onchip peripheral functions like I/O ports, timers/counters, versatile serial communication port. So it is clear that this 8031 architecture was designed to cater many real time embedded needs.

The following list gives the features of the 8031 architecture:

- Optimized 8 bit CPU for control applications.
- Extensive Boolean processing capabilities.
- 64K Program Memory address space.
- 64K Data Memory address space.
- 128 bytes of onchip Data Memory.
- 32 Bi-directional and individually addressable I/O lines.
- Two 16 bit timer/counters.
- Full Duplex UART.
- 6-source / 5-vector interrupt structure with priority levels.
- Onchip clock oscillator.

Now you may be wondering about the non mentioning of memory space meant for the program storage, the most important part of any embedded controller. Originally this 8031 architecture was introduced with onchip, 'one time programmable' version of Program Memory of size 4K X 8. Intel delivered all these microcontrollers (8051) with user's program fused inside the device. The memory portion was mapped at the lower end of the Program Memory area. But, after getting devices, customers couldn't change any thing in their program code, which was already made available inside during device fabrication.

Intel's 8031 Architecture

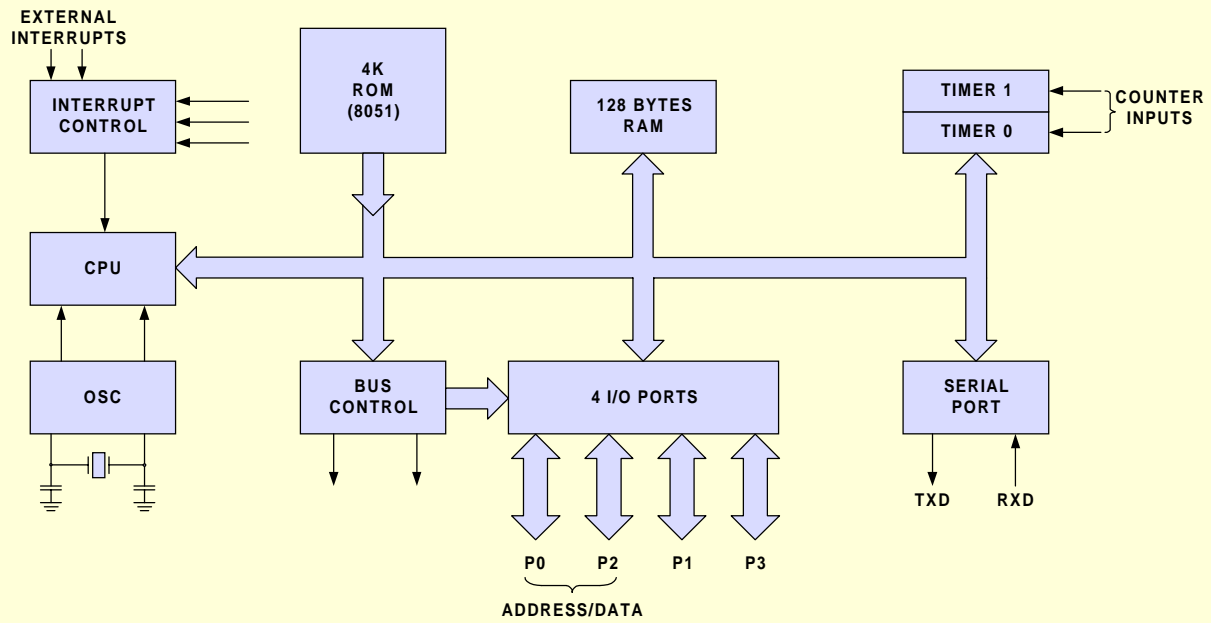


Figure 1 - Block Diagram of the 8031 Core

So, very soon Intel introduced the 8031 devices (8751) with re-programmable type of Program Memory using built-in EPROM of size 4K X 8. Like a regular EPROM, this memory can be re-programmed many times. Later on Intel started manufacturing these 8031 devices without any onchip Program Memory.

Now I go ahead giving more information on the important functional blocks of the 8031.

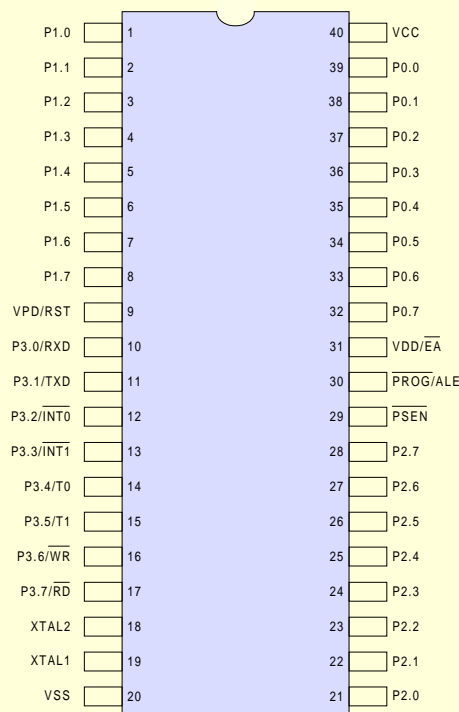


Figure 2 - 8031 Microcomputer Pinout Diagram

Intel's 8031 Architecture

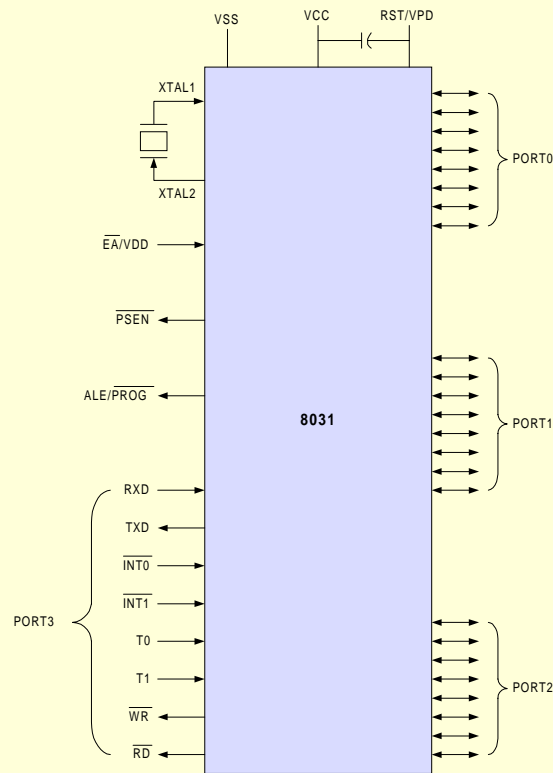


Figure 3 - 8031 Microcomputer logic symbol

2.2 - Central Processing Unit

The CPU is the brain of the microcontrollers reading user's programs and executing the expected task as per instructions stored there in. Its primary elements are an 8 bit Arithmetic Logic Unit (ALU), Accumulator (Acc), few more 8 bit registers, B register, Stack Pointer (SP), Program Status Word (PSW) and 16 bit registers, Program Counter (PC) and Data Pointer Register (DPTR).

The ALU (Acc) performs arithmetic and logic functions on 8 bit input variables. Arithmetic operations include basic addition, subtraction, multiplication and division. Logical operations are AND, OR, Exclusive OR as well as rotate, clear, complement and etc. Apart from all the above, ALU is responsible in conditional branching decisions, and provides a temporary place in data transfer operations within the device.

B register is mainly used in multiply and divide operations. During execution, B register either keeps one of the two inputs and then retains a portion of the result. For other instructions, it can be used as another general purpose register.

Program Status Word keeps the current status of the ALU in different bits.

Central Processing Unit

Stack Pointer (SP) is an 8 bit register. This pointer keeps track of memory space where the important register information are stored when the program flow gets into executing a subroutine. The stack portion may be placed in any where in the onchip RAM. But normally SP is initialized to 07H after a device reset and grows up from the location 08H. The Stack Pointer is automatically incremented or decremented for all PUSH or POP instructions and for all subroutine calls and returns.

Program Counter (PC) is the 16 bit register giving address of next instruction to be executed during program execution and it always points to the Program Memory space.

Data Pointer (DPTR) is another 16 bit addressing register that can be used to fetch any 8 bit data from the data memory space. When it is not being used for this purpose, it can be used as two eight bit registers.

2.3 - Input / Output Ports

The 8031's I/O port structure is extremely versatile and flexible. The device has 32 I/O pins configured as four eight bit parallel ports (P0, P1, P2 and P3). Each pin can be used as an input or as an output under the software control. These I/O pins can be accessed directly by memory instructions during program execution to get required flexibility.

These port lines can be operated in different modes and all the pins can be made to do many different tasks apart from their regular I/O function executions. Instructions, which access external memory, use port P0 as a multiplexed address/data bus. At the beginning of an external memory cycle, low order 8 bits of the address bus are output on P0. The same pins transfer data byte at the later stage of the instruction execution.

Also, any instruction that accesses external Program Memory will output the higher order byte on P2 during read cycle. Remaining ports, P1 and P3 are available for standard I/O functions. But all the 8 lines of P3 support special functions: Two external interrupt lines, two counter inputs, serial port's two data lines and two timing control strobe lines are designed to use P3 port lines. When you don't use these special functions, you can use corresponding port lines as a standard I/O.

Even within a single port, I/O operations may be combined in many ways. Different pins can be configured as input or outputs independent of each other or the same pin can be used as an input or as output at different times. You can comfortably combine I/O operations and special operations for Port 3 lines.

2.4 - Timers / Counters

8031 has two 16 bit Timers/Counters capable of working in different modes. Each consists of a 'High' byte and a 'Low' byte which can be accessed under software. There is a mode control register and a control register to configure these timers/counters in number of ways.

These timers can be used to measure time intervals, determine pulse widths or initiate events with one microsecond resolution upto a maximum of 65 millisecond (corresponding to 65, 536 counts). Use software to get longer delays. Working as counter, they can accumulate occurrences of external events (from DC to 500KHz) with 16 bit precision.

2.5 - Serial Port

Each 8031 microcomputer contains a high speed full duplex (means you can simultaneously use the same port for both transmitting and receiving purposes) serial port which is software configurable in 4 basic modes: 8 bit UART; 9 bit UART; Interprocessor Communications link or as shift register I/O expander.

For the standard serial communication facility, 8031 can be programmed for UART operations and can be connected with regular personal computers, teletype writers, modem at data rates between 122 bauds and 31 kilobauds. Getting this facility is made very simple using simple routines with option to select even or odd parity. You can also establish a kind of Interprocessor communication facility among many microcomputers in a distributed environment with automatic recognition of address/data.

Apart from all above, you can also get super fast I/O lines using low cost simple TTL or CMOS shift registers.

2.6 - Memory Organization

The 8031 architecture provides both onchip memory as well as off chip memory expansion capabilities. It supports several distinctive 'physical' address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write controls signals or both:

- On chip Program Memory
- On chip Data Memory
- Off chip Program Memory
- Off chip Data Memory
- On chip Special Function Registers

The Program Memory area (EPROM incase of external memory or Flash/EPROM incase of internal one) is extremely large and never lose information when the power is removed. Program Memory is used for information needed each time power is applied: Initialization values, Calibration data, Keyboard lookup tables etc along with the program itself. The Program Memory has a 16 bit address and any particular memory location is addressed using the 16 bit Program Counter and instructions which generate a 16 bit address.

Onchip Data memory is smaller and therefore quicker than Program Memory and it goes into a random state when power is removed. Onchip RAM is used for variables which are calculated when the program is executed.

In contrast to the Program Memory, On chip Data Memory accesses need a single 8 bit value (may be a constant or another variable) to specify a unique location. Since 8 bits are more than sufficient to address 128 RAM locations, the onchip RAM address generating register is single byte wide.

Different addressing mechanisms are used to access these different memory spaces and this greatly contributes to microcomputer's operating efficiency.

The 64K byte Program Memory space consists of an internal and an external memory portion. If the \overline{EA} pin is held high, the 8051 executes out of internal Program Memory unless the address exceeds 0FFFH and locations 1000H through FFFFH are then fetched from external Program Memory. If the \overline{EA} pin is held low, the 8031 fetches all instructions from the external Program Memory. In either case, the 16 bit Program Counter is the addressing mechanism.

Memory Organization

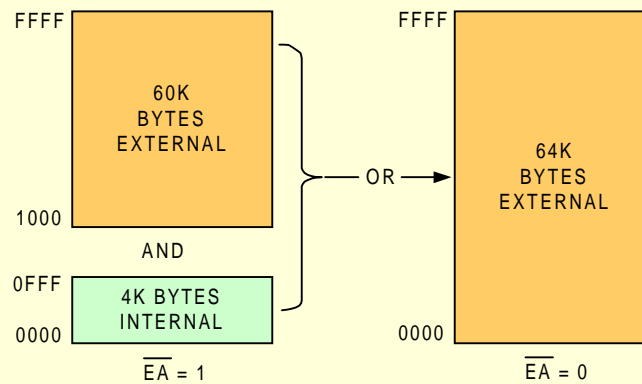


Figure 4 - Program Memory

The Data Memory address space consists of an internal and an external memory space. External Data Memory is accessed when a MOVX instruction is executed.

Apart from onchip Data Memory of size 128/256 bytes, total size of Data Memory can be expanded upto 64K using external RAM devices.

Total internal Data Memory is divided into three blocks:

Lower 128 bytes.

Higher 128 bytes

Special Function Register space.

Higher 128 bytes are available only in 8032/8052 devices.

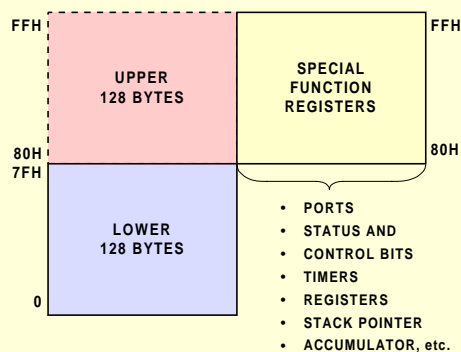


Figure 5 - Internal Data Memory

Even though the upper RAM area and SFR area share same address locations, they are accessed through different addressing modes. Direct addresses higher than 7FH access SFR memory space and indirect addressing above 7FH access higher 128 bytes (in 8032/8052).

Memory Organization

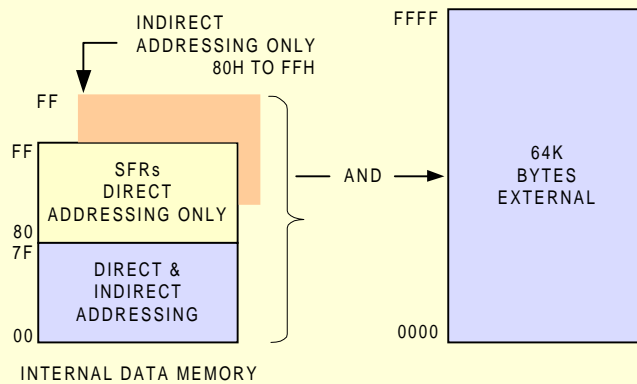


Figure 6 - Data Memory

The next figure indicates the layout of lower 128 bytes. The lowest 32 bytes (from address 00H to 1FH) are grouped into 4 banks of 8 registers. Program instructions refer these registers as R0 through R7. Program Status Word indicates which register bank is being used at any point of time.

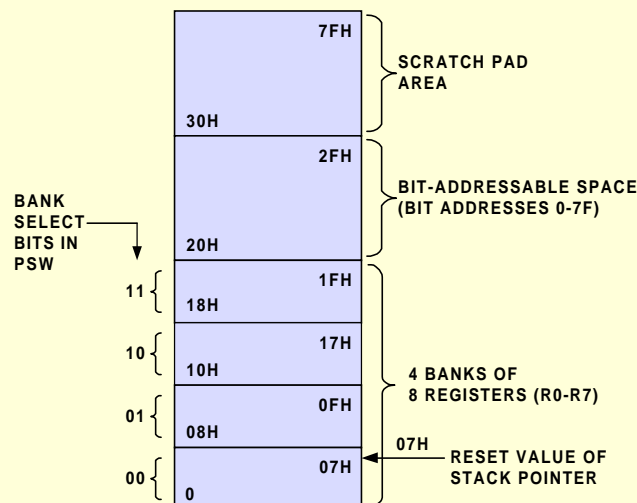


Figure 7 - Lower 128 Bytes of Internal RAM

The next 16 bytes above these register banks form a block of bit addressable memory space. The instruction set of 8031 contains a wide range of single bit processing instructions and these instructions can directly access the 128 bits of this area.

The SFR space includes port latches, timer and peripheral control registers. All the members of 8031 family have same SFR at the same SFR locations. There are some 16 unique locations which can be accessed as bytes and as bits.

2.7 - Common Memory Space

The 8031's Data Memory may not be used for program storage. So it means you can't execute instructions out of this Data Memory.

But, there is a way to have a single block of offchip memory acting as both Program and Data Memory. By gating together both memory read controls (\overline{RD} and \overline{PSEN}) using an AND gate, a common memory read control signal can be generated.

In this arrangement, both memory spaces are tied together and total accessible memory is reduced from 128 Kbytes to 64 Kbytes.

The 8031 can read and write into this common memory block and it can be used as Program and Data Memory.

You can use this arrangement during program development and debugging phase. Without taking Microcontroller off the socket to program its internal ROM (EPROM/Flash ROM), you can use this common memory for frequent program storage and code modifications.

2.8 - Interrupts

The 8031 has five interrupt sources: one from the serial port when a transmission or reception operation is executed; two from the timers when overflow occurs and two come from the two input pins INT0, INT1. Each interrupt may be independently enabled or disabled to allow polling on same sources and each may be classified as high or low priority.

A high priority source can override a low priority service routine. These options are selected by interrupt enable and priority control registers, IE and IP.

When an interrupt is activated, then the program flow completes the execution of the current instruction and jumps to a particular program location where it finds the interrupt service routine. After finishing the interrupt service routine, the program flows return to back to the original place.

The Program Memory address, 0003H is allotted to the first interrupt and next seven bytes can be used to do any task associated with that interrupt.

<u>Interrupt Source</u>	<u>Service routine starting address</u>
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/counter 1	001BH
Serial port	0023H

2.9 - Addressing Modes

8031's assembly language instruction set consists of an operation mnemonic and zero to three operands separated by commas. In two byte instructions the destination is specified first, and then the source. Byte wide mnemonics like ADD or MOV use the Accumulator as a source operand and also to receive the result.

The 8031 supports five types of addressing modes:

- Register Addressing
- Direct Addressing
- Register Indirect Addressing
- Immediate Addressing
- Index Addressing

2.9.1 - Register Addressing

Register Addressing accesses the eight working registers (R0-R7) of the selected register bank. The least significant three bits of the instruction opcode indicate which register is to be used for the operation. One of the four banks of registers is to be predefined in the PSW before using register addressing instruction. ACC, B, DPTR and CY, (the Boolean Accumulator) can also be addressed in this mode.

2.9.2 - Direct Addressing

Direct addressing can access any onchip variables or hardware register. To indicate the address of the location, an additional byte is attached to the opcode. Depending on the highest order bit of the direct address byte one of two physical memory space is selected.

Direct Addressing

When the direct address range is between 0 and 127 (00H - 7FH) one of the 128 low order onchip RAM location is accessed. All I/O ports, special function, control registers are assigned between 128 and 255 (80H - FFH). When direct addressing indicates any location in this range, corresponding hardware register is accessed.

This is the only method available for accessing I/O ports and special function registers.

2.9.3 - Register Indirect Addressing

Register indirect addressing uses the contents of either R0 or R1 (in the pre selected register bank) as a address pointer to locate in a 256 byte block (the lower 128 bytes of internal RAM in 8031 or 256 bytes in 8032) or the lower 256 bytes of external data memory. Note that the special function registers are not accessible in this mode. Access to full 64K external data memory address space is indicated by the 16 bit Data Pointer register, DPTR.

Execution of PUSH and POP instructions also involve indirect register addressing. The Stack Pointer indicates the correct stack location anywhere in the internal RAM.

2.9.4 - Immediate Addressing

When a source operand is a constant rather than a variable, then the constant can be embedded into the instruction itself. This kind of instructions take two bytes and first one specifies the opcode and second byte gives the required constant.

2.9.5 - Index Addressing

Only the Program Memory can be accessed by this mode. This mode is intended for reading lookup tables in the Program Memory. A 16 bit base register (either DPTR or the Program Counter) points to the base of the lookup tables and the Accumulator carries the constant indicating table entry number. The address of the exact location of the table is formed by adding the Accumulator data to the base pointer.

2.10 - Instruction Set

8031 architecture sports a powerful and versatile instruction set that enables the user to develop a compact program. There is a facility to manipulate both byte data and 1 bit binary data. The table gives complete information on the instruction set.

2.10.1 - Data Transfer Instructions

Mnemonic	Operation	Execution Time (μ s) for 12MHz operation
MOV A, <src>	A = <src>	1
MOV <dest>,A	<dest> = A	1
MOV <dest>, <src>	<dest> = <src>	2
MOV DPTR, #data16	DPTR = 16-bit immediate constant	2
PUSH <src>	INC SP : MOV "@SP",<src>	2
POP <dest>	MOV <dest>, "@SP" : DEC SP	2
XCH A,<byte>	ACC and <byte> exchange data	1
XCHD a,@Ri	ACC and @Ri exchange low nibble	1

The above table gives the instructions that can be used to move data around internal memory spaces and the addressing modes that can be used with each one.

The MOV <dest>, <src> instruction allows data to be transferred between any two internal RAM locations without disturbing the Accumulator. The upper 128 bytes of the data RAM can be accessed only by indirect addressing and SFR space by direct addressing.

In all 8031 devices, the stack resides in onchip RAM, and grows upwards. The PUSH instruction first increments the Stack Pointer (SP), then copies the byte into the stack. PUSH and POP use only direct addressing to identify the byte being saved or restored. But the stack itself is accessed by indirect addressing using the SP register. This means the stack can go into the upper 128 bytes if they are implemented but not into SFR space.

Instruction Set

In devices that don't have upper 128 bytes, if the SP points to anywhere in upper 128 bytes, pushed bytes are lost and popped bytes are indeterminate.

The data transfer instructions include a 16 bit MOV that can be used to initialize the Data Pointer (DPTR) for lookup tables in Program Memory or for 16 bit external Data Memory accesses.

The XCH A, <byte> instruction causes the Accumulator and the addressed byte to exchange the data. The XCHD A, @Ri instruction exchange only low nibble between the Accumulator and the addressed byte.

2.10.2 - Data Transfer In External RAM

This following table gives possible data transfer operations in external data memory space. Only indirect addressing can be used. The choice is whether to use a one byte address by @Ri, where Ri can be either R0 or R1 of the selected register bank or a two byte address, @DPTR.

Note that in all external data RAM accesses the Accumulator is always either the destination or source of data.

The read and write strobes to external RAM are activated only during the execution of a MOVX instruction. Normally these signals are inactive and in fact if they are not going to be used at all, then their pins are available as extra I/O lines.

Address width	Mnemonic	Operation	Execution Time (μ s) for 12MHz operation
8 bits	MOVX A, @Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri, A	Write external RAM @ Ri	2
16 bits	MOVX A, @DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR, A	write external RAM @DPTR	2

2.10.3 - Lookup Tables

This table shows the two instructions that are available for reading lookup tables from Program Memory. Since they access only Program Memory, the lookup tables can only be read not updated.

The mnemonic MOVC is for 'move constant'. If the table access is to external Program Memory, then the read strobe is $\overline{\text{PSEN}}$.

The first MOVC instruction of the table can read a byte from 256 entries, numbered 0 through 255. The number of the desired entry loaded into the Accumulator and the Data Pointer is set up to point to beginning of the table.

The other MOVC instruction works with the Program Counter (PC). Hence PC acts as the table base and the Accumulator should carry the table entry value.

Mnemonic	Operation	Execution Time (μs) for 12MHz operation
MOVC A,@A+DPTR	Read Program memory at (A+DPTR)	2
MOVC A,@A+PC	Read Program memory at (A+PC)	2

2.10.4 - Arithmetic Instructions

Note that most of the operations use Accumulator and any byte in the internal data memory space can be increased or decrease without using Accumulator.

The instruction MUL AB multiplies the unsigned eight bit integer values held in the Accumulator and B registers. The lower order byte of the 16 bit product is left in the Accumulator and the higher order byte in B.

DIV AB divides the unsigned eight bit integer in the Accumulator by the unsigned eight bit integer in the B register. The integer part of quotient stays with the Accumulator and the remainder in the B register. The DAA instruction is for BCD arithmetic operations. In BCD arithmetic, ADD and ADD C instructions should always be followed by a DAA operation, to ensure that the result is also in BCD. The DAA operation produces a meaningful result only in the second step when adding two BCD bytes.

Mnemonic	Operation	Execution Time (μ s) for 12MHz operation
ADD A, <byte>	$A = A + \text{<byte>}$	1
ADDC A, <byte>	$A = A + \text{<byte>} + C$	1
SUBB A, <byte>	$A = A - \text{<byte>} - C$	1
INC A	$A = A + 1$	1
INC <byte>	$\text{<byte>} = \text{<byte>} + 1$	1
INC DPTR	$DPTR = DPTR + 1$	2
DEC A	$A = A - 1$	1
DEC <byte>	$\text{<byte>} = \text{<byte>} - 1$	1
MUL AB	$B:A = B \times A$	4
DIV AB	$A = \text{Int} [A/B]$ $B = \text{Mod} [A/B]$	4
DA A	Decimal adjust	1

2.10.5 - Logical Instructions

The following table gives the list of 8031's logical instructions. The instructions that perform Boolean operations (AND, OR, Exclusive OR, NOT) on bytes perform the operation on a bit by bit basis. All of the logical instructions that are Accumulator specific execute in 1 μ s (using a 12MHz clock). Others take 2 μ s.

Boolean operations can be performed on any byte in the lower 128 internal data memory space or the SFR space using direct addressing without having to use the Accumulator. If the operation is in response to an interrupt, not using the Accumulator saves time and effort in the interrupt service routine.

Mnemonic	Operation	Execution Time (μ s) for 12MHz operation
ANL A, <byte>	A = A .AND. <byte>	1
ANL <byte>,A	<byte> = <byte> .AND. A	1
ANL <byte>,#data	<byte> = <byte> .AND. #data	2
ORL A, <byte>	A = A .OR. <byte>	1
ORL <byte>,A	<byte> = <byte> .OR. A	1
ORL <byte>,#data	<byte> = <byte> .OR. #data	2
XRL A, <byte>	A = A .XOR. <byte>	1
XRL <byte>,A	<byte> = <byte> .XOR. A	1
XRL <byte>,#data	<byte> = <byte> .XOR. #data	2
CLR A	A = 00H	1
CPL A	A = .NOT. A	1
RL A	Rotate ACC Left 1 bit	1
RLC A	Rotate Left through Carry	1
RR A	Rotate ACC Right 1 bit	1
RRC A	Rotate Right through Carry	1
SWAP A	Swap Nibbles in A	1

Logical Instructions

The Rotate instructions (RL A, RLC A, etc.) shift the Accumulator 1 bit to the left or right. For a left rotation, the MSB rolls into the LSB position and for a right rotation, the LSB rolls into MSB position.

The SWAP A instruction interchanges the high and low nibbles within the Accumulator. This is a useful operation in BCD manipulation.

2.10.6 - Program Control – Jumps, Calls, and Returns

LJMP (long jump) encodes a 16 bit address in the 2nd and 3rd instruction bytes. The destination may be anywhere in the 64K byte Program Memory address space.

The two byte AJMP (Absolute jump) instruction encodes its destination using a 11 bit address which is embedded in the instruction itself. Address bits 10 through 8 form a 3 bit field in the opcode and address bits 7 through 0 form a second byte.

Address bits 15-11 remain unchanged from the incremented contents of the PC, so AJMP can only be used when the destination is known to be within the same 2K memory block.

2.10.7 - Jump Instructions

Following table gives a list of unconditional jumps.

Mnemonic	Operation	Execution Time (µs) for 12MHz operation
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

Jump Instructions

As you can see, there are three types of jump operations:

SJMP – Short Jump

LJMP – Long Jump

AJMP – Absolute Jump

Basically these jump operations differ from each other by the way of address generation meant for that jump operation.

SJMP (Short Jump) determines the destination with a Program Counter relative address mentioned in the second byte. The CPU calculates the destination at run time by adding the signed 8 bit displacement value to the incremented PC. Negative offset values will cause jumps upto 128 bytes backwards; positive values upto 127 bytes forward.

In contrast, LJMP instructions jump into any place in the 64K bytes program space. The instruction is three bytes long, first byte is the opcode and next two bytes give destination address.

Like SJMP, AJMP instructions jump into much longer space, anywhere into 2K block. The instruction is 2 bytes long, containing the opcode in the first byte and the second byte holds low byte of the destination address. The opcode itself carries higher order bits of 2K space (bits 8 – 10). During the instruction execution, these 11 bits are simply substituted for the low 11 bits in the PC. Hence, the destination has to be within the same 2K block as the instruction following the AJMP.

The JMP @A+DPTR instruction supports case jumps. The destination address is calculated during run time as the sum of the 16 bit DPTR register and the Accumulator.

Typical application for this case jump is the facility to jump to the exact location in a lookup table. This particular type of jumps is one of the most wanted operations. Normally, the DPTR holds the address of a lookup table. During runtime, the Accumulator is made to calculate the exact position of the required byte. This instruction sums up both DPTR and the Accumulator and then jumps to that specific byte.

These are two CALL instructions: LCALL and ACALL. These instructions are used to call subroutine and they differ in a way the subroutine address is determined.

Each instruction increments the PC to the first byte of the following instruction and then pushes it onto the stack (low byte first). Saving both bytes increments the Stack Pointer by two.

Jump Instructions

LCALL instruction is for the long subroutine call operation and the instruction is a 3 byte long one and second and third bytes carry the address of the subroutine. So the subroutine can be anywhere in the 64K Program Memory space.

But the ACALL instruction drives an absolute subroutine call operation and uses the 11 bit address format. The subroutine should be within the 2K block as the instruction following the ACALL.

Subroutines should end with a RET instruction that pops the high and low order bytes of the PC successively from the stack, decreasing the Stack Pointer by two and program execution continues at the address pushed, the first byte of the instruction immediately following the call.

RETI can be used to return from an interrupt service routine. The only difference between RET and RETI is that RETI tells the interrupt control system that the interrupt in progress is done. If there is no active interrupt operation, then the RETI is functionally same as RET.

2.10.8 - Conditional Jump Instructions

Like SJMP, all conditional jump instructions use relative addressing: JZ (jump if Zero) and JNZ (jump if not Zero) monitor the state of the Accumulator as implied by their names while JC (jump on carry) and JNC (jump on no carry) test whether or not the carry flag is set. All these 4 instructions are two byte instructions.

Mnemonic		Operation	Execution Time (μ s) for 12MHz operation
JZ	rel	Jump if A = 0	2
JNZ	rel	Jump if A \neq 0	2
JC	rel	Jump if C = 1	2
JNC	rel	Jump if C = 0	2

The 8031 instruction set supports another set of conditional jump instruction using Boolean processing. They are covered separately.

2.10.9 - Operate and Branch Instructions**CJNE, DJNZ**

This group of instructions combine a byte operation with a conditional jump based on the results. CJNE (Compare and Jump if Not Equal) compares two byte operands and execute a jump if they disagree. The carry flag is set following the rules of subtraction. If the unsigned integer value of the first operand is less than that of second, then it is set. Otherwise, it is cleared.

The CJNE instruction can also be used for loop control. Two bytes are specified in the operand field of the instruction. The jump is executed only if the two bytes are not equal.

DJNZ (Decrement and Jump if not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) using a single instruction.

CJNE and DJNZ, like all conditional jumps use Program Counter relative addressing for the destination address.

Mnemonic	Operation	Execution Time (μ s) for 12MHz operation
DJNZ <byte>,rel	Decrement and jump if not zero	2
CJNE A,<byte>,rel	Jump if A \neq <byte>	2
CJNE <byte>,#data,rel	Jump if <byte> \neq #data	2

2.10.10 - Boolean Instructions

The 8031 devices contain a complete Boolean (single bit) processor. The internal RAM contains 128 addressable bits and the SFR space can support upto 128 other addressable bits. All of the port lines are bit addressable and each one can be treated as a separate single bit port.

The instructions that access these bits are not just conditional branches, but a complete menu of MOVE, SET, CLEAR, COMPLEMENT, OR, AND instructions. This range of bit operations are not easily obtained in other architectures with any amount of byte oriented software.

The instruction set for the Boolean processor is shown here. All bit accesses are direct addressing. Bit addresses 00H through 7FH are in the lower 128 bytes and bit addresses 80H through FFH are in SFR space.

The carry bit is used as the Boolean processor. Bit instructions that refer to the carry bit as C assemble as carry specific instructions. The carry bit also has a direct address, since it resides in the PSW register which is bit addressable.

Boolean instructions support JB (Jump on Bit), JNB (Jump on No Bit), JBC (Jump on Bit and Clear), JC (Jump on Carry) and JNC (Jump on No Carry) operations.

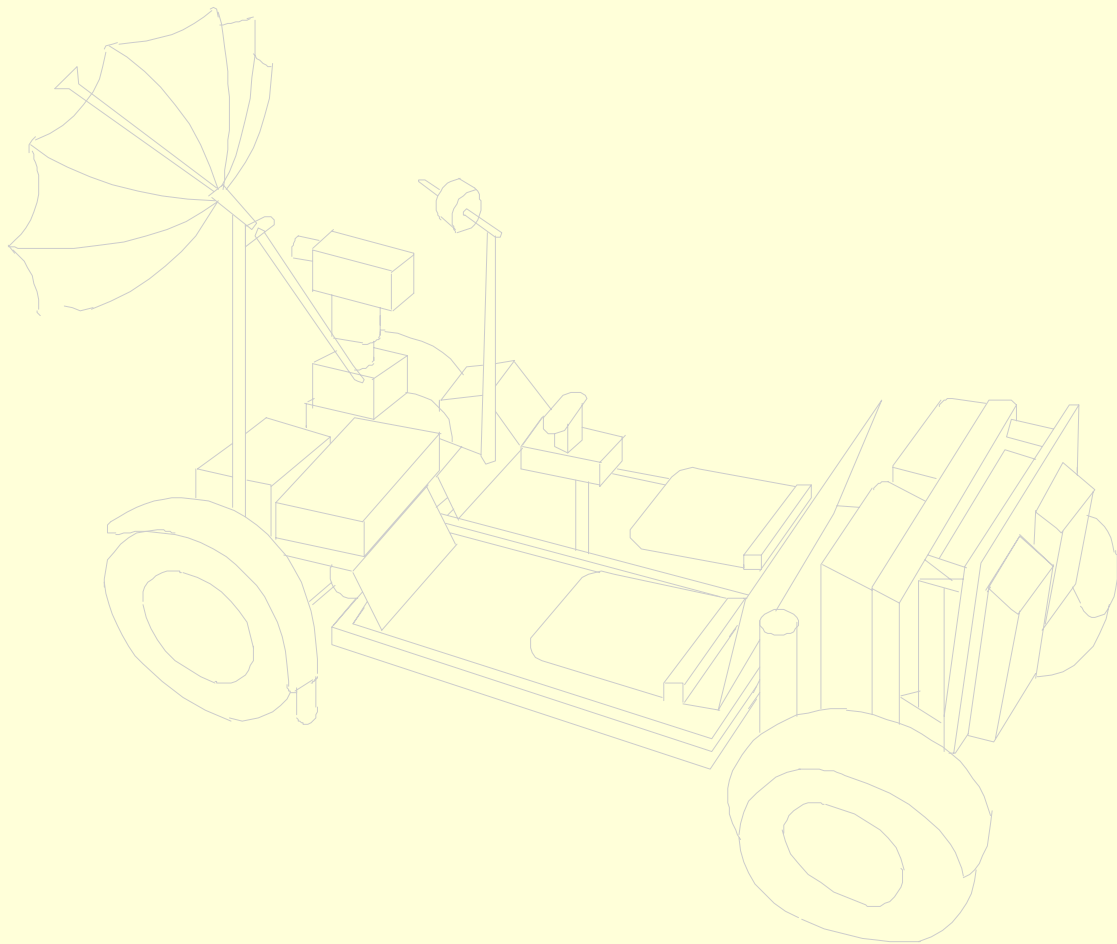
The destination address for these jumps is specified in the second byte of the instruction. This is a signed (two's complement) offset byte that is added to the PC if the jump is executed. The range of jump is therefore -128 to +127 Program Memory bytes relative to the first byte following the jump instruction.

Boolean Instructions

Mnemonic	Operation	Execution Time (μ s) for 12MHz operation
ANL C, bit	$C = C \text{ .AND. bit}$	2
ANL C,/bit	$C = C \text{ .AND. .NOT. bit}$	2
ORL C, bit	$C = C \text{ .OR. bit}$	2
ORL C,/bit	$C = C \text{ .OR. .NOT. bit}$	2
MOV C,bit	$C = \text{bit}$	1
MOV bit.C	$\text{bit} = C$	2
CLR C	$C = 0$	1
CLR bit	$\text{bit} = 0$	1
SETB C	$C = 1$	1
SETB bit	$\text{bit} = 1$	1
CPL C	$C = \text{.NOT. C}$	1
CPL bit	$\text{bit} = \text{.NOT. bit}$	1
JC rel	Jump if $C = 1$	2
JNC rel	Jump if $C = 0$	2
JB bit,rel	Jump if $\text{bit} = 1$	2
JNB bit,rel	Jump if $\text{bit} = 0$	2
JBC bit,rel	Jump if $\text{bit} = 1$; CLR bit	2

Chapter 3 - 8031 Derivatives

- 8031 Derivatives
- Why Atmel Devices?



3.1 - 8031 Derivatives

Along the way, this 8031 architecture gained enviable market acceptance. Many semiconductor manufacturers started either manufacturing the 8031 devices as such (Intel was liberal in giving away license to whoever asked) or developing a new kind of microcontrollers based on 8031 core architecture.

Manufacturers modified the basic 8031 architecture and added many new peripheral functions to make them attractive to the designers.

Because of the rush, electronic community started getting a variety of 8031 based devices with range of options. To beat the competition, manufacturers developed different microcontrollers with many unique features.

These parts are popularly known as '8031 Derivatives'. Almost every decent manufacturer boasted of having an 8031 based microcontroller in the line card.

First major manufacturer was the Philips who brought out more than 40-50 derivatives with a variety of I/O options, memory combinations, and peripheral functions. Devices became available in regular DIP and SMD packages. With the basic 8031 core, Philips ported high capacity Program Memory (upto 32K/64K), its patented I²C interface bus, 8/10 bit Analog to Digital Converters, CAN Bus, Capture and Compare registers, Watch dog timer, PWM facilities and etc. More I/O ports (as many as eight ports), additional timer/counter, second serial port were also made available in Philips devices.

Apart from all these, Philips developed many consumer devices meant for telecom, computer and TV applications. A smart card controller was also developed by incorporating a cryptographic engine. So Philips clearly established itself as the market leader in 8031 derivatives and still caters to this segment.

Then came Dallas semiconductor. Dallas redesigned the 8031 architecture and eliminated waste clock cycles of original core and made all instructions executed in less clock cycles (maximum of 4) which has traditionally taken upto 12 clock cycles. So, came the birth of High speed 8031 Derivatives.

Dallas also maintained the same device pin out configurations to enable the user get upto 3X performance by replacing slower parts with a Dallas device. So, existing compiled code started running faster without any modification. These days, you can find Dallas devices giving upto 50 MIPS (Million Instructions Per Second).

Apart from this, Dallas introduced additional Serial port, Watch Dog Timer, Precision Reset Circuitry, Real Time Clock, Power Fail Monitor in the 8031 devices. Also a second data pointer, more onchip RAM space and more interrupt lines were also made available.

Dallas semiconductor also has got a range of secure microcontrollers based on 8031 core. This microcontroller family uses non volatile RAM to keep both program and data. Because of this RAM, the controller gives the In System Reprogrammability. Dallas has combined this microcontroller, SRAM and lithium cell in a single pack. This device guarantees 10+ years of data retention in the RAM area. This 8031 also boasts the tamper proof security features like Real Time Memory Encryption, user selected 48 bit Encryption key, memory contents, security lock and the facility to hide interrupt vector table. As you can agree, this particular 8031 device has found a niche market in banking and security related applications.

Atmel Corporation is the another major semiconductor manufacturer who introduced many flash memory based 8031 derivatives at a competitive cost. Atmel used its expertise in flash memory technology into the basic 8031 core and brought out microcontrollers with a variety of flash memory options and few devices also carry In System Reprogramming facility. You can program/reprogram this microcontroller after soldering the device in the target board. If this programming facility is embedded in the system software, then the tasks like remote calibration, onsite system upgradation become as easy as sending your data/program in a floppy disk or by internet. Atmel devices sport security lock to its flash memory to protect the contents from the prying eyes.

Meantime, Intel itself tried to cash in the popularity of this 8031 architecture and introduced improved versions of microcontrollers: 80151 and 80251 families. These devices sport 16 bit architecture using 8031 core and unfortunately these devices have not become as popular as 8031.

Even after many years of introduction, 8031 core is still going strong in 8 bit arena. Major manufacturers like AMD, Siemens slowly discontinued making 8031 devices in favour of their own babies. Yet, many new companies try their luck in making these devices. It seems like this 8031 core is the solid bet for anyone wants to enter into microcontrollers.

3.2 - Why Atmel Devices?

In this part of the world (precisely India where I live) Atmel devices are freely available everywhere at an affordable cost. Since Atmel has very good distribution network in many countries, I am sure you can also find Atmel devices in your place.

I like Atmel's flash memory very much. In early days, we used to get EPROM devices from 'God - only - knows' sources and we always worried about the life of our target hardware. Atmel really relieved us from this. Without caring about EPROM erases, we can program/reprogram the device during prototype development and most of the time device stays cool even after many hours of running. Also we can go upto 24MHz with these devices using 3V to 6V supply.

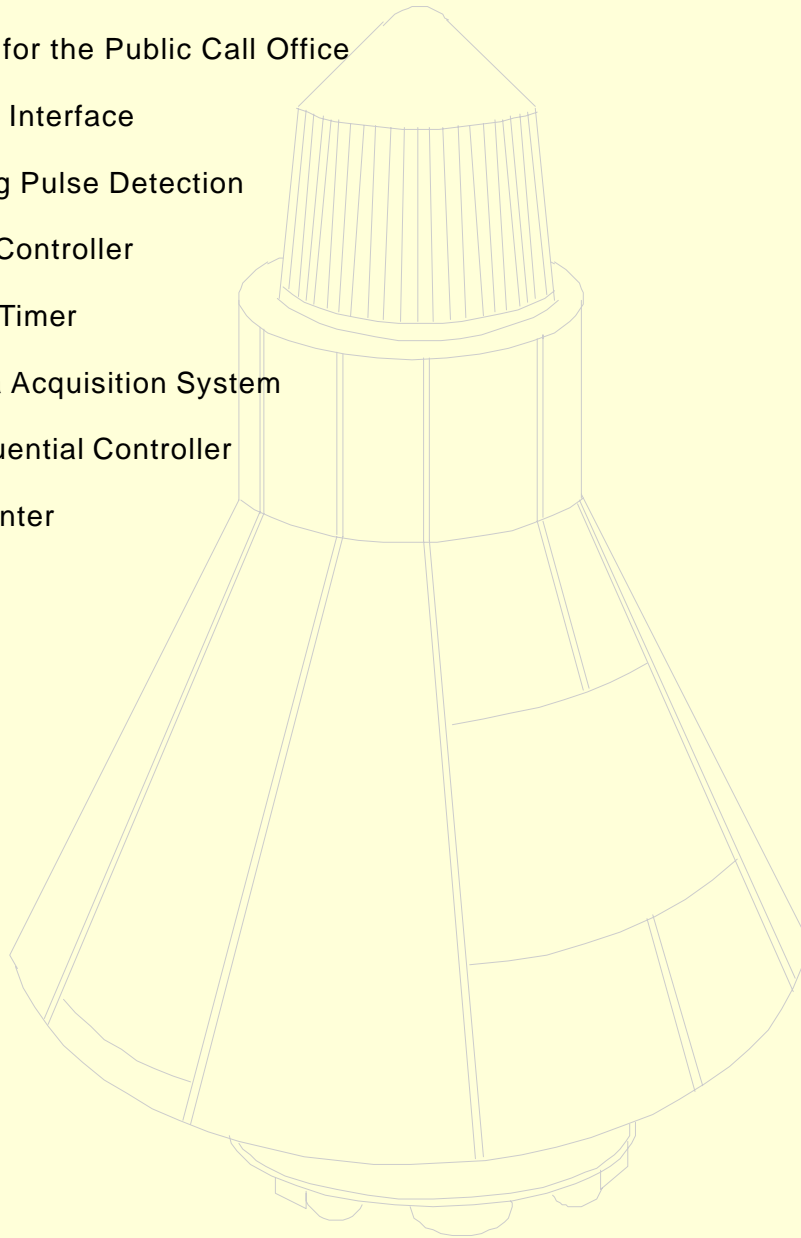
Also Whatever we are going to discuss can be applied to many other 8031 devices without any problem.

The following table gives overall picture on available Atmel devices and the salient features.

	AT89C51	AT89LV51	AT89C52	AT89LV52	AT89C2051	AT89C1051	AT89S8252	AT89S53	AT89LS8252	AT89LS53
Bytes Flash Program Memory	4K	4K	8K	8K	2K	1K	8K	12K	8K	12K
Bytes Data Memory	128 RAM	128 RAM	256 RAM	256 RAM	128 RAM	64 RAM	256 RAM	256 RAM	256 RAM	256 RAM
Bytes Onchip EEPROM							2K EEPROM		2K EEPROM	
I/O Pins	32	32	32	32	15	15	32	32	32	32
16-Bit Timer/Counters	2	2	3	3	2	1	3	3	3	3
UART	1	1	1	1	1		1	1	1	1
Interrupt Sources	6	6	8	8	6	3	9	9	9	9
Power Down and Idle Mode	X	X	X	X	X	X	Yes	Yes	Yes	Yes
Low Voltage Operation		X		X	X	X			X	X
Security Lock Bits	3	3	3	3	2	2	3	3	3	3
SPI Serial Interface							Yes	Yes	Yes	Yes
Watchdog Timer							Yes	Yes	Yes	Yes
Dual Data Pointer							Yes	Yes	Yes	Yes
Interrupt Recovery from Power Down							Yes	Yes	Yes	Yes

Chapter 4 - Real Life Projects

- 16KHz Monitor for the Public Call Office
- Telephone Line Interface
- 16KHz Metering Pulse Detection
- Stepper Motor Controller
- Programmable Timer
- 8 Channel Data Acquisition System
- 8 Channel Sequential Controller
- Frequency Counter



4. - Real Life Projects

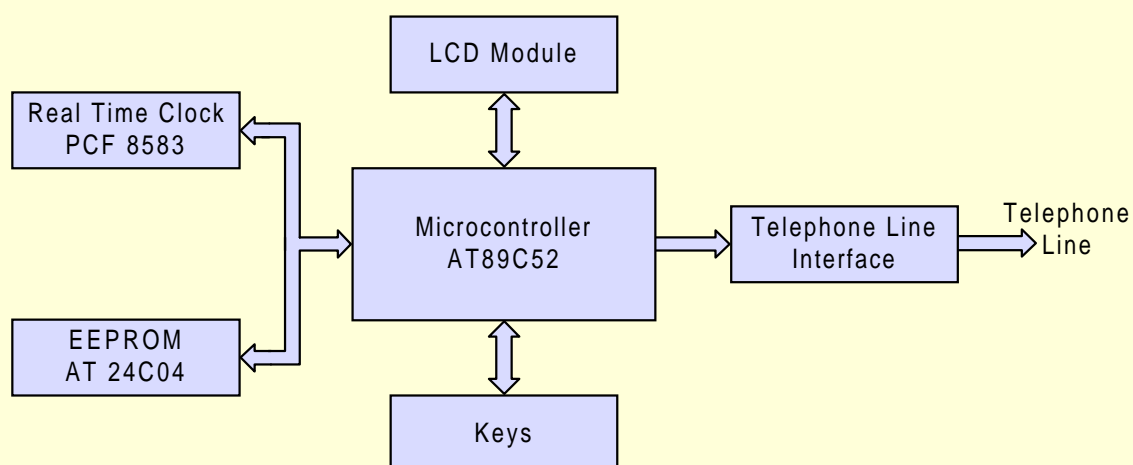
Having introduced 8031 architecture in your knowledge base, it is time for me to make you understand what is actually going on in the Embedded World.

Now I am going to discuss about 6 simple yet useful embedded applications to give you more idea. All of these projects are modeled from the commercial products. There are many companies manufacturing these products for the general market. Then I proceed with telling you about various hardware and software development tools you may need during project implementation.

This flow of discussion should make you comfortable in facing more complex and elaborate designs.

4.1 - 16KHz Monitor for the Public Call Office

This project is a full fledged commercial one and is widely used in India and many other countries to manage billing operations in public call offices. Basically, it monitors the telephone line in phone booths where these equipments generate billing information for the long distance calls.



This project is a complex one and needs much space for the complete description. Here I give overall picture about the design and explain briefly about the tasks handled by the microcontroller.

Many years back, we were manufacturing this equipment for the commercial applications and then slowly discontinued making them.

During long distance calls, the concerned telephone exchange sends out a 16 KHz metering signal to that particular telephone line. The number of metering pulses we get for a call indicates the cost for that call. The phone booth gets this metering signals periodically till the call is completed. Longer the distance, faster is the metering signal and for the long distance calls, the caller gets more metering pulses and has to pay more for the call.

16KHz Monitor for the Public Call Office

To get the complete picture of the operations, observe the following with attention:

First, the caller dials his required phone number. During dialing, the equipment indicates the phone number in the LCD module.

Once the call is established, the caller's exchange sends a metering signal to that phone booth.

Now this equipment starts counting these pulses and at the end of the call, it even prints out details as a bill.

The display indicates the pulse counting or the progressive call charges for the caller's convenience. Almost all of these equipments calculate the call charges in the local currency.

The equipment keeps the copy of call records for further reference. All relevant information about the calls originated from that particular phone booth like dialing number, duration of the phone call, number of metering pulses, and the charges meant for that call are stored in the memory of that equipment.

Now you might have gotten the total picture about this particular project and now I go on giving you more information about the actual design.

This design is meant for the low cost compact solution. It just monitors the outgoing call and generates the billing information in the display. If you need the printing facility and call recording features, you may have to expand this basic design. But principle is the same.

4.2 - Telephone Line Interface

Now you look into this telephone line interface with attention. A pair of opto couplers are used to monitor the current flow going in the telephone line. The ON HOOK / OFF HOOK condition of the telephone is sensed through these opto couplers.

Also the same information can be used during pulse dialing to get the dialing number. A relay connected in series with the telephone line can be activated to disconnect the line.

Opto couplers give logic one level when there is no current and a logic zero level when current flows in the telephone lines. These opto coupler signals are conditioned using schemit triggers, 74LS14 and then given to the port lines of the microcontroller.

Telephone line Interface

A DTMF receiver, 8870 is connected in parallel with the telephone line to sense the dialing number in tone dialing mode. The device monitors the DTMF dialing signals coming out of the phone and generates the corresponding digit information in 4 output lines. The microcontroller's port lines, P0.0 to P0.3 get this information and the port line P0.4 monitors the strobe signal coming out of 8870.

4.3 - 16KHz Metering Pulse Detection

This is the most important part of the design. Yet it is very simple to implement. Main part is played by the Tone Detector Device, LM567. Infact, the metering pulse is a burst of signal of 16 KHz frequency and it is available for about 50-70 milliseconds. For an example, if you assume a 4 second metering signal, then you should get this 16 KHz signal for about 50 milliseconds at every 4 seconds once the call is connected.

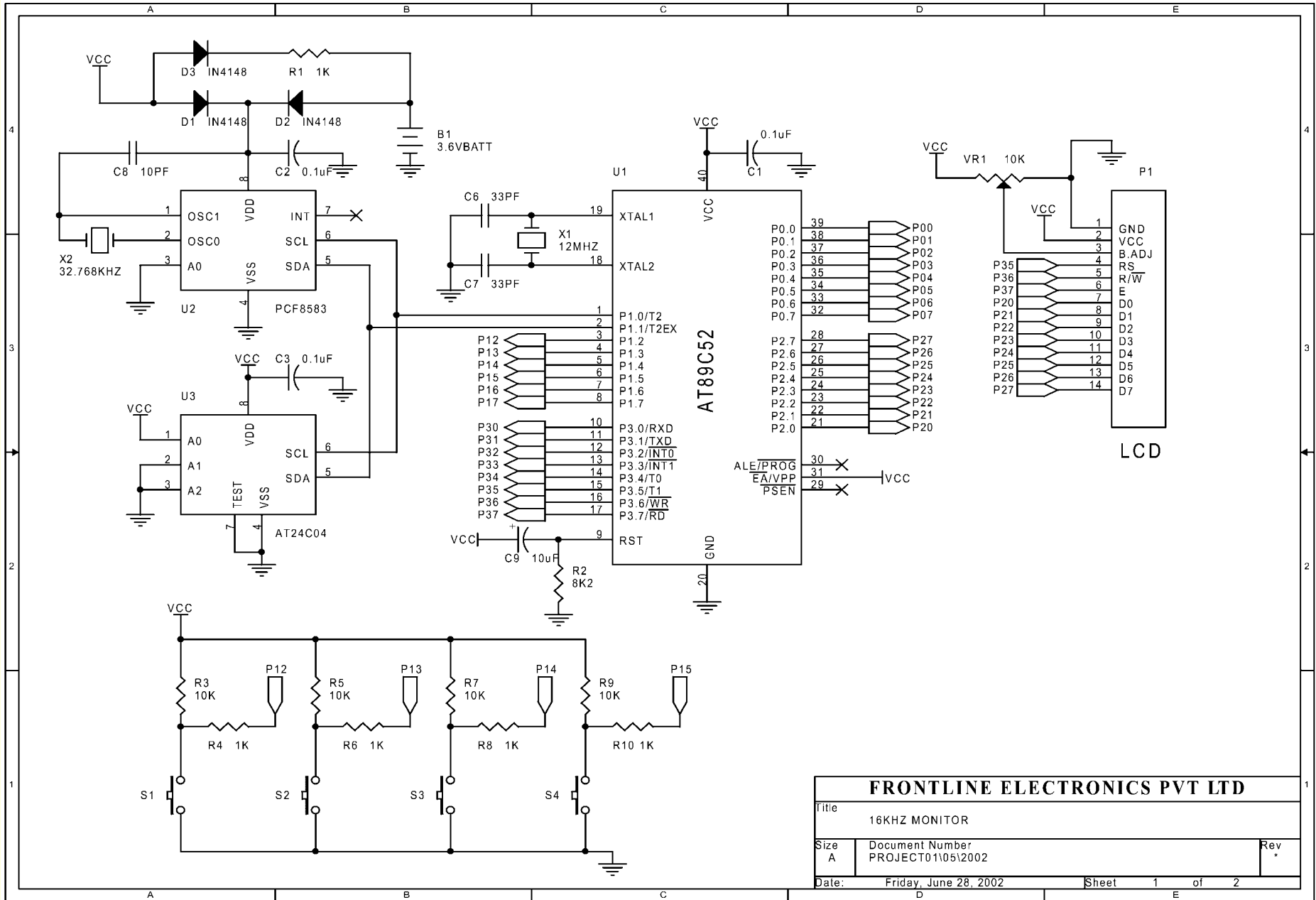
It is a simple PLL based detector that has an internal oscillator running at 16 KHz. If the incoming tone matches this oscillator frequency, then a logic zero is generated at the output pin 8. So, you can see that this output stays in logic high when there is no metering pulse, and goes to low level when the detector senses a 16 KHz metering pulse. The detector's output is connected with the interrupt line of the microcontroller.

The microcontroller is supported with many devices in the project. Since the design requires the timing information for its operations, naturally the RTC has come into picture. A serial EEPROM of required size keeps the reference data and also acts as storage space to keep call records for further reference.

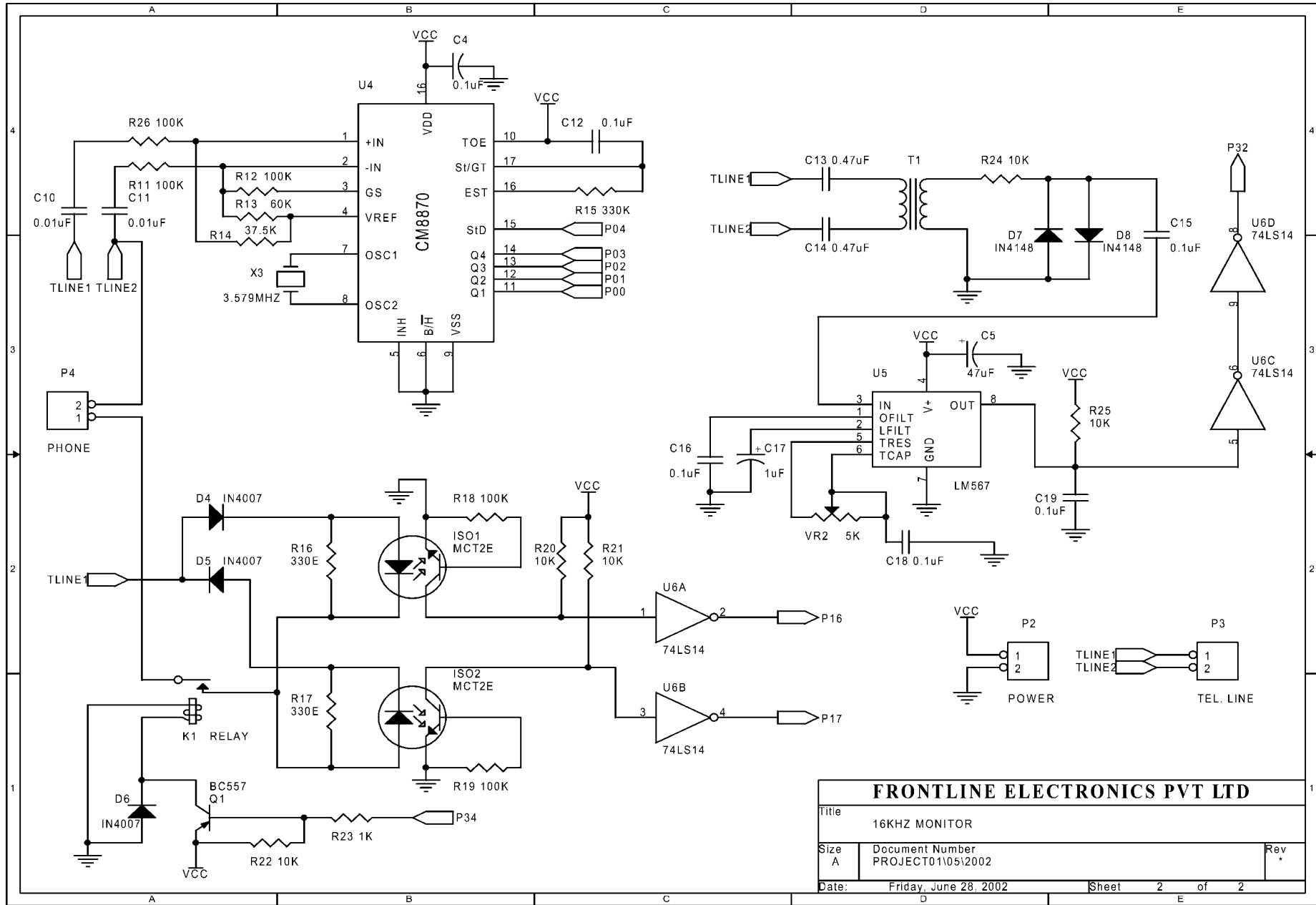
For the user interactivity, LCD module of 1 line X16 characters and 4 keys (S1-S4) are used. LCM is interfaced with the microcontroller through I/O lines of Ports 2 and 3.

Because of the space restriction, I have omitted detailed information on the devices, LM567, MT 8870. You can get them from data books of National Semiconductor, and MITEL.

The complete circuit of the design is made available here for your convenience.



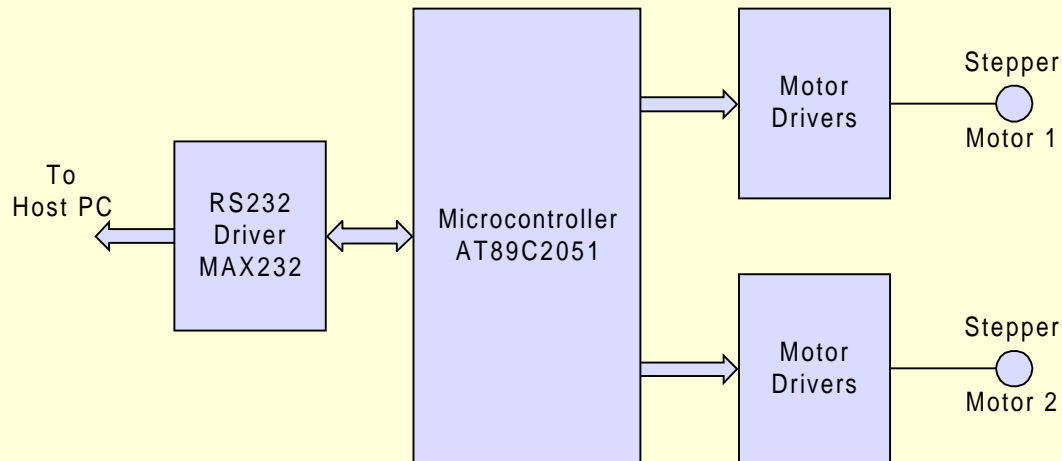
FRONTLINE ELECTRONICS PVT LTD		
Title 16KHZ MONITOR		
Size A	Document Number PROJECT01105\2002	Rev .
Date: Friday, June 28, 2002	Sheet 1	of 2



FRONTLINE ELECTRONICS PVT LTD		
Title 16KHZ MONITOR		
Size A	Document Number PROJECT01\05\2002	Rev *
Date: Friday, June 28, 2002	Sheet 2	of 2

4.4 - Stepper Motor Controller

This is another interesting project. It is a very simple one yet very important one in many places. One of the most useful module in the robotic field. It controls two stepper motors as per the commands it receives from the host computer.

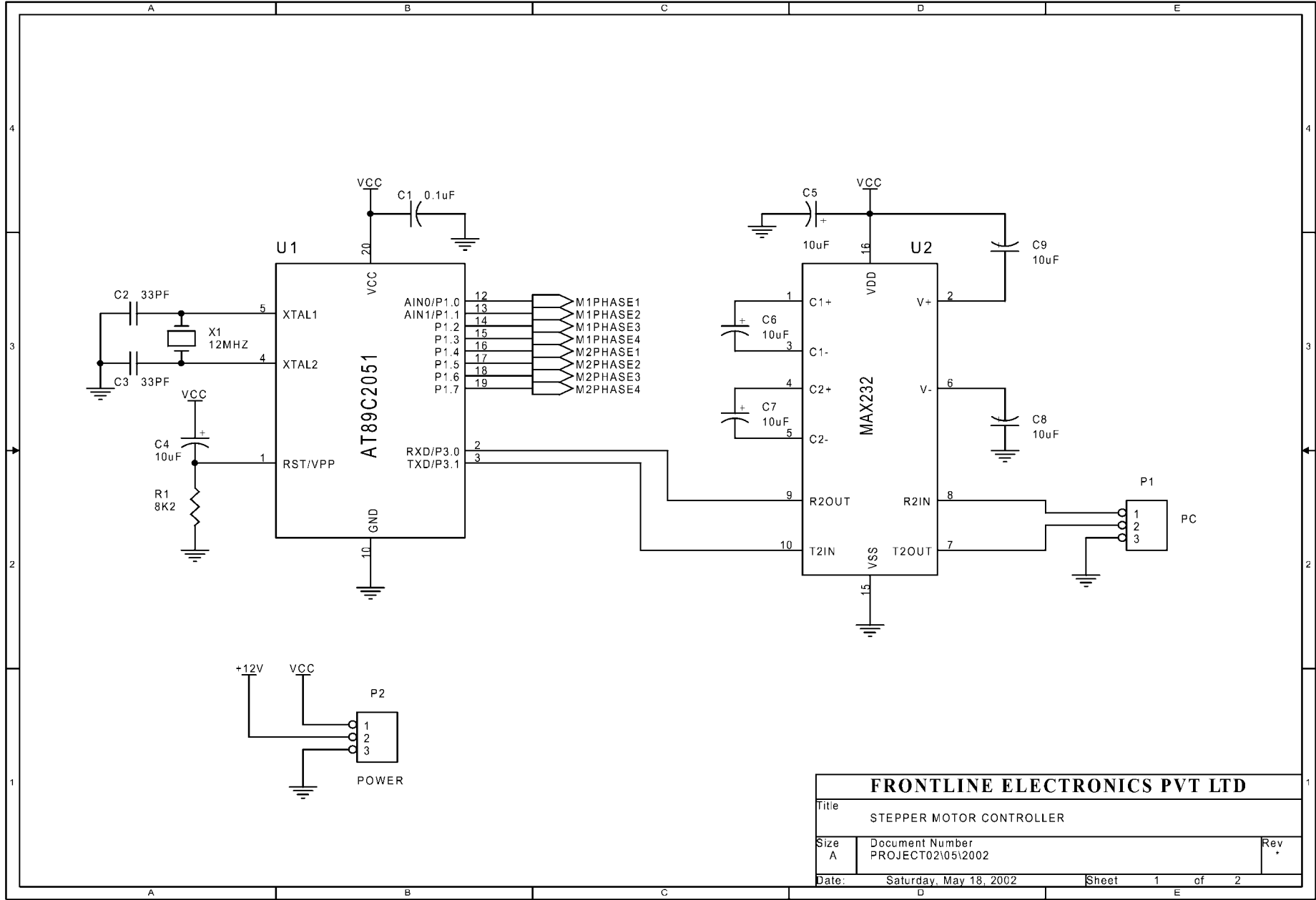


Stepper motors play a vital role in the robotic environments. They are widely used to implement arms, handles, or other moving/rotating mechanisms. Stepper motor is very simple in operation and it rotates in steps unlike other motors. When you activate the motor with a proper control, it's shaft rotates in a step. It may be 1.8 or 3.6 degrees. To make the shaft rotate upto a certain angle, you activate the windings by a fixed number of times using proper control sequence.

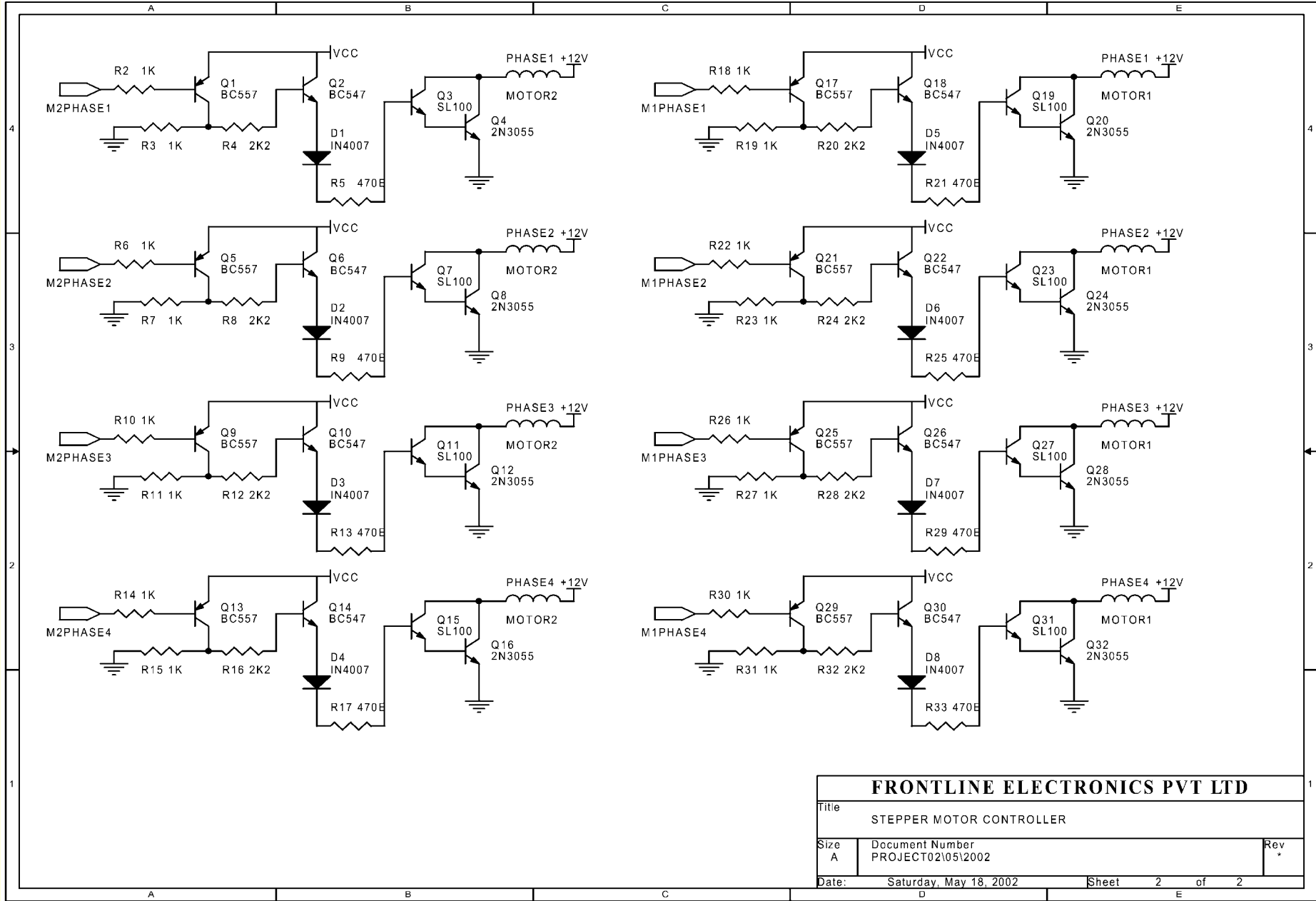
Here I am not going deep into stepper motor operation or controls. Any standard text book will give you relevant information.

Basically this controller takes care of operation of two stepper motors. The controller can be connected to another master or to a host computer. This master sends commands to this controller on how many steps it should move, in which direction, in what speed and etc. Then this controller locally manages these two stepper motors relieving the master from worrying this part of the control.

Here is the complete circuit for the controller.



FRONTLINE ELECTRONICS PVT LTD		
Title STEPPER MOTOR CONTROLLER		
Size A	Document Number PROJECT02\05\2002	Rev *
Date: D	Saturday, May 18, 2002	Sheet 1 of 2



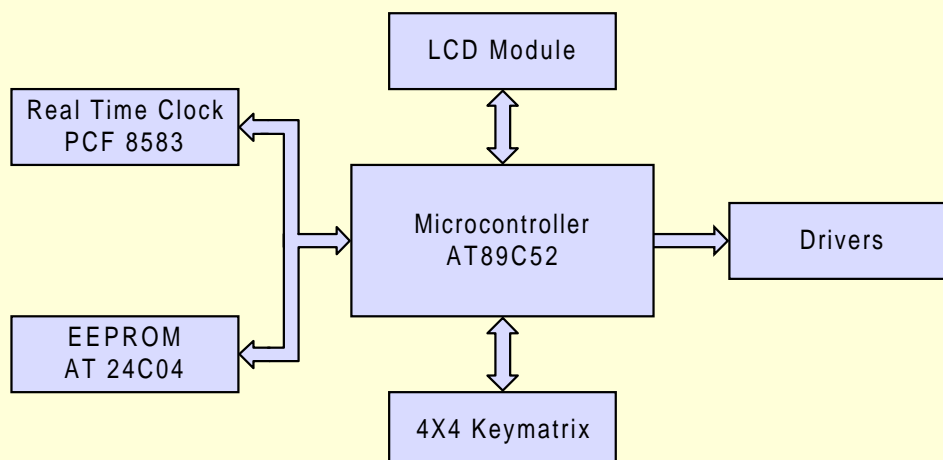
FRONTLINE ELECTRONICS PVT LTD		
Title STEPPER MOTOR CONTROLLER		
Size A	Document Number PROJECT02\05\2002	Rev *
Date: D	Saturday, May 18, 2002	Sheet 2 of 2

4.5 - Programmable Timer

This one is a simple timing control project that executes many time-based tasks.

An Atmel microcontroller has become our target microcontroller. A Real Time Clock device is used to keep track of time. Basically this RTC is an useful device to keep track of time in applications like Data Logging, Point Of Sale terminals, Credit Card Readers, Access controllers where the timing information should be recorded for many events. The RTC we use comes from Philips, PCF 8583, that features an IIC interface. This interface is a kind of serial interfacing standard patented by Philips that needs only 2 I/O lines for interfacing. The IIC bus is for bi-directional, two line communication between different ICs or modules. The two lines are a Serial Data line (SDA) and a Serial Clock Line (SCL).

Using this RTC in the given application is very simple. After proper initialization, we can leave the chip to operate on its own. We need to set the device with current timing information like seconds, minutes, hours, year, date, weekdays. Then the device automatically tracks the time and updates its internal timing registers.



So whenever we read this device, we can know the current time at that moment.

Apart from this timing function, the device also sports 256 bytes of RAM area which can be used to keep timing reference information and also an interrupt facility that interrupts the microcontroller when the internal compare registers find the current timing match with the preset values.

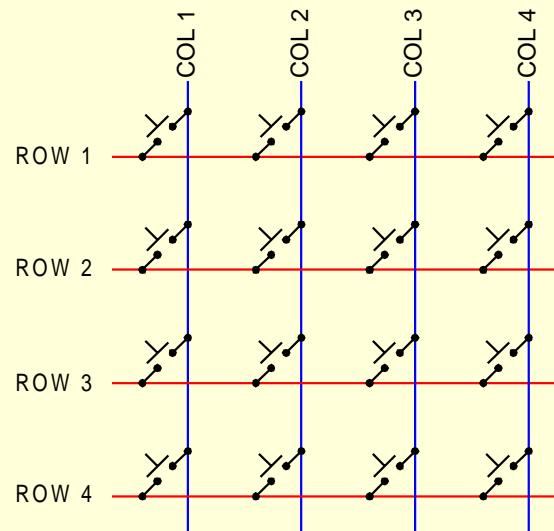
So you can note that this RTC device has become an important part of this project.

Then comes human-machine interactive part: Displays and the keyboard. We can use Liquid Crystal Display of size (1 line X 16 characters) and also a keyboard with 16 keys. We can make the LCD module displays the prompt messages to the users and also make it display whatever information it gets from the user for verification.

Programmable Timer

Most of the character type LCD modules have a parallel bus for interfacing within the embedded controls. It has an eight bit bus, a module select signal (CS), a Read/ Write (R/W) signal and another line (A0) to select the address. So, the microcontroller has to spare 11 I/O lines to have this LCD facility.

The keyboard has 16 keys arranged as a keymatrix of size 4X4. These keys are located at the sixteen intersecting points of the matrix.



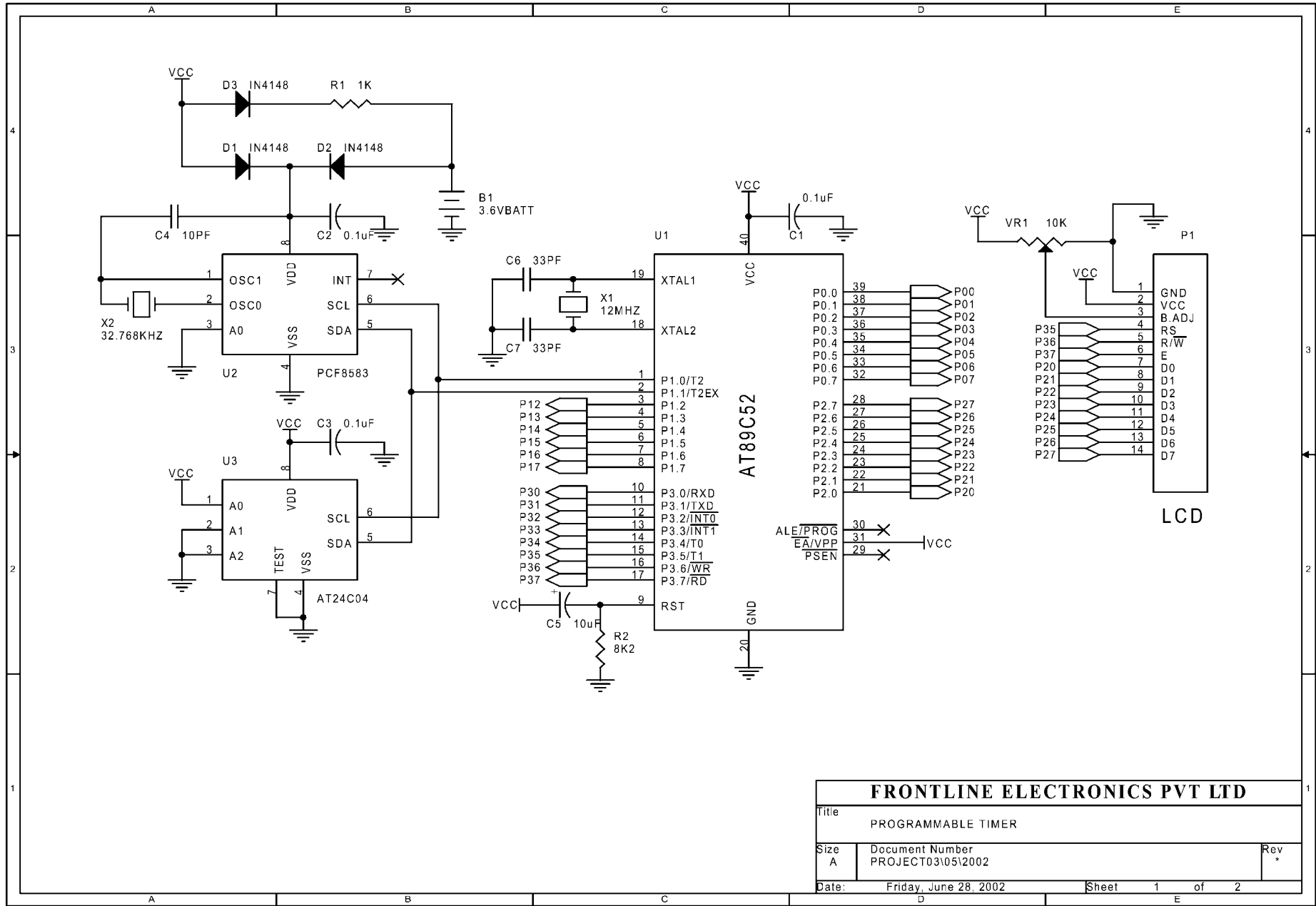
The Column and Row lines are connected with the I/O lines. So we need 8 I/O lines to interface this key matrix.

Now comes the controller part of the project. We can add contactors, switches, relays, solenoids and etc to control many gadgets around our home or work place.

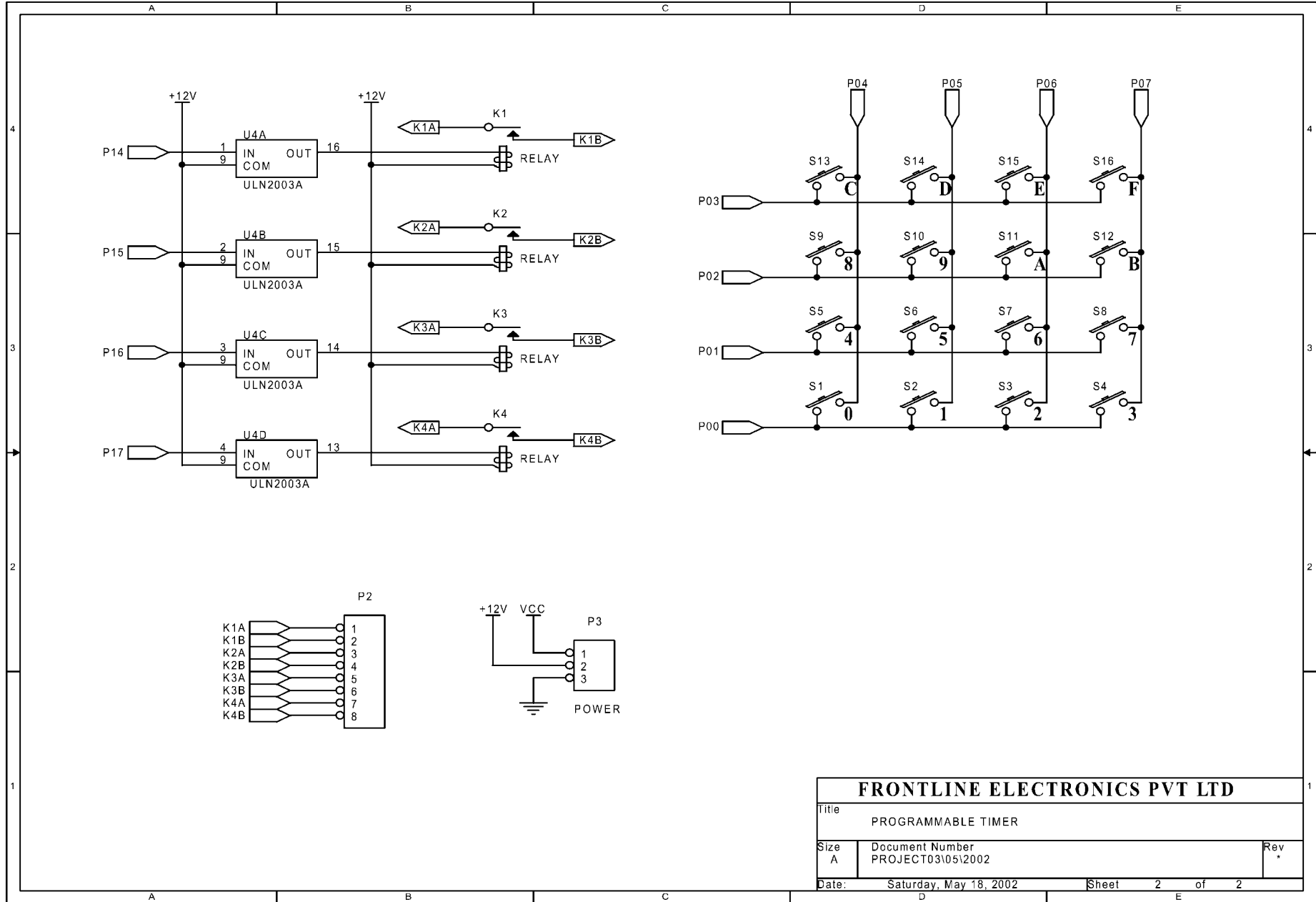
We can write programs to control time based daily chores. We can make Motors/ Compressors, lights switch on and off at predetermined times.

Another interesting application is, you can make this to monitor and watch the activities of the watchman or guard in your home or work space. You ask him press a key or button at fixed timing intervals and make the equipment record the same for your reference. You can find out how well be guards or sleeps in your place in your absence. There is a commercial equipment available preciously for this purpose.

So the block diagram has turned out as the complete circuit diagram here:



FRONTLINE ELECTRONICS PVT LTD		
Title		
PROGRAMMABLE TIMER		
Size	Document Number	Rev
A	PROJECT03\05\2002	*
Date:	Friday, June 28, 2002	Sheet 1 of 2

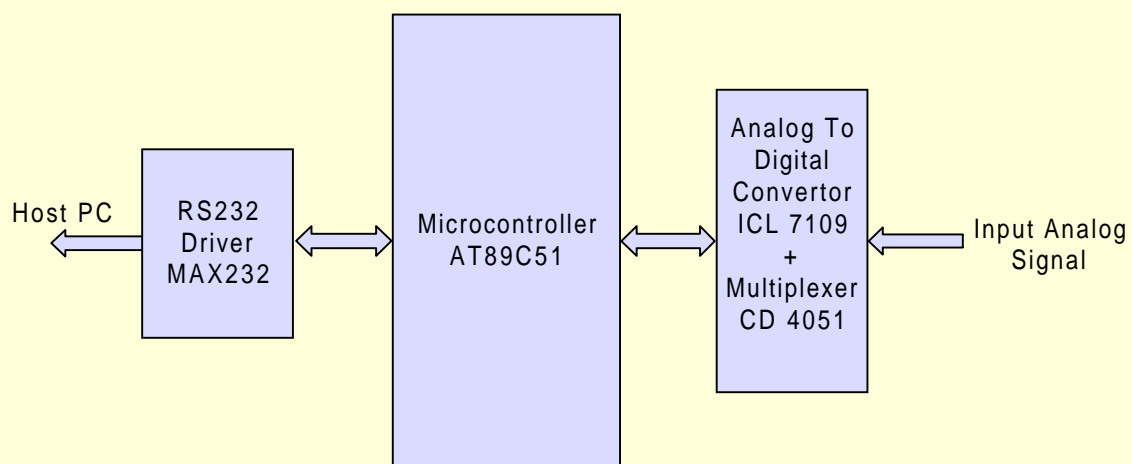


FRONTLINE ELECTRONICS PVT LTD		
Title PROGRAMMABLE TIMER		
Size A	Document Number PROJECT03\05\2002	Rev *
Date: Saturday, May 18, 2002	Sheet 2	of 2

4.6 - 8 Channel Data Acquisition System

Now comes another most used project of the industrial utility. It is one of the basic equipment you can find anywhere in industrial environments. The main task of this system is to log many analog inputs and sends the digital data corresponding to these inputs to the host controller.

This system stays at a remote part of the plant and gets commands from the host, monitors the various analog input signals. It sports a 12 bit Analog to Digital converter from the Intersil, 7109, the most popular one in the market. It is suitable for low speed analog signals (data conversion rate is less than 50 samples per second). Multiple Analog inputs are given to this ADC through an 8 channel analog multiplexer, CD 4051.



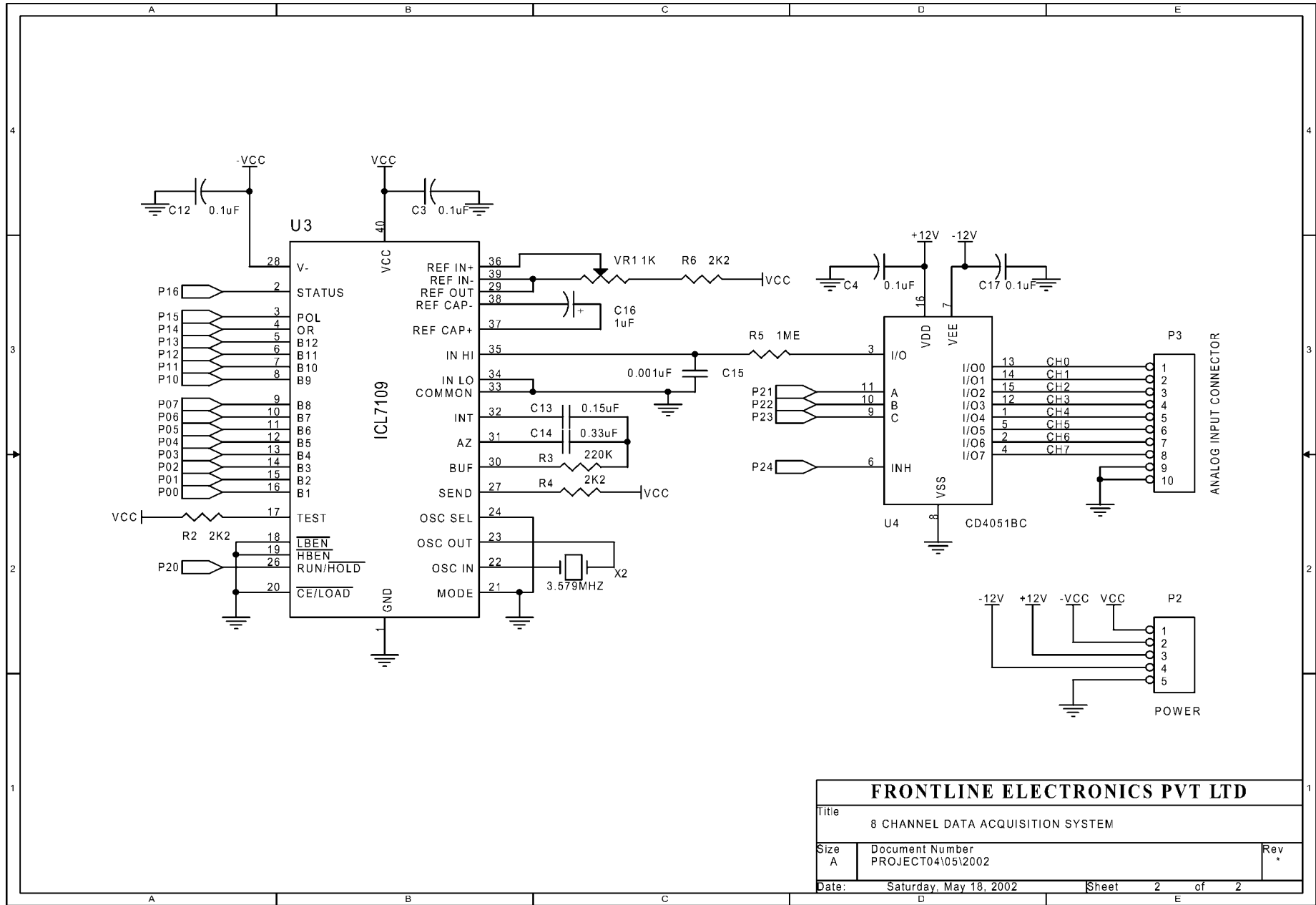
Depends upon the transducer and the application you may have to use a signal conditioning part between the multiplexer and the transducer.

The serial port of the 89C51 is connected to the host computer through the level shifting IC, MAX 232.

During normal operation, this system monitors all the analog inputs at the remote place relieving the main controller from this job. Depends upon the command it gets from the host, it monitors the input signals and it may send alarm if any analog input exceeds the reference value set before. If necessary, we can make the system display or activate an alarm at that workspace to get the operator's attention.

We have used AT 89C51 as the controlling microcontroller. You can add a RTC to time stamp the logging events. If you feel like, you can add keyboard or LCD module to introduce facility for user interaction.

The following circuit diagram gives you the complete picture.

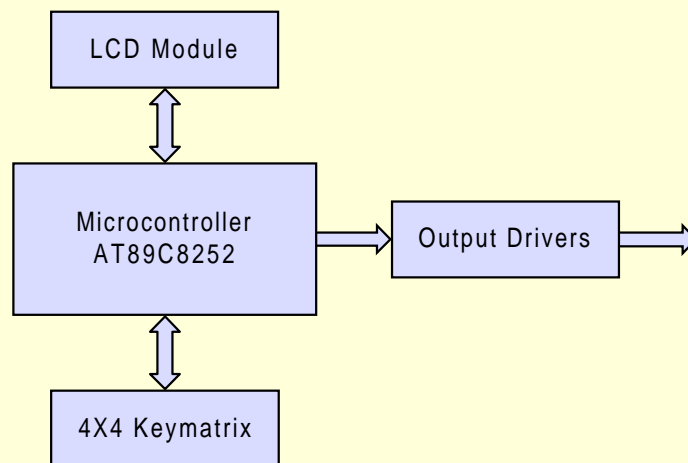


FRONTLINE ELECTRONICS PVT LTD		
Title		
8 CHANNEL DATA ACQUISITION SYSTEM		
Size	Document Number	Rev
A	PROJECT04\05\2002	*
Date:	Saturday, May 18, 2002	Sheet 2 of 2

4.7 - 8 Channel Sequential Controller

This one is an useful industrial controller meant for switching on/off many operations in an equipment. Once we designed a more complex sequential controller meant for waste clearing purpose in a textile processing mill that used similar logic.

Basically the setup is a simple one where it is required to activate lot of solenoids to evacuate the waste accumulated under every processing machine. The solenoids operate at predefined times, and a motor sucks out the waste from every machine periodically. The controller is responsible for coordinating all the operations of this waste evacuating system. Here important task for the controller is switching on solenoids at the correct time and at the correct sequence at every machine and also activating other solenoids at other strategic locations to guide the waste in to the duct till it reaches the packing section.



Similarly, in a bag-sealing environment, a controller has to operate many relays, solenoids, contactors at the correct time to finish the job of filling up the bag with the required quantity of materials. If the precision filling is required, you may have to incorporate a weighing load cell to monitor the quantity.

The controller I present here has control for 8 outputs. You can connect relays, solenoids, solid state relays, contactors and etc. User interaction can be established using the Liquid Crystal Display Module and the keypad. The Atmel microcontroller, AT 89S8252 is selected to handle all these operations. This device also sports on chip EEPROM to keep reference values of timing and etc. Microcontroller's internal timer is used to keep track of timing requirements. The block diagram gives the total picture.

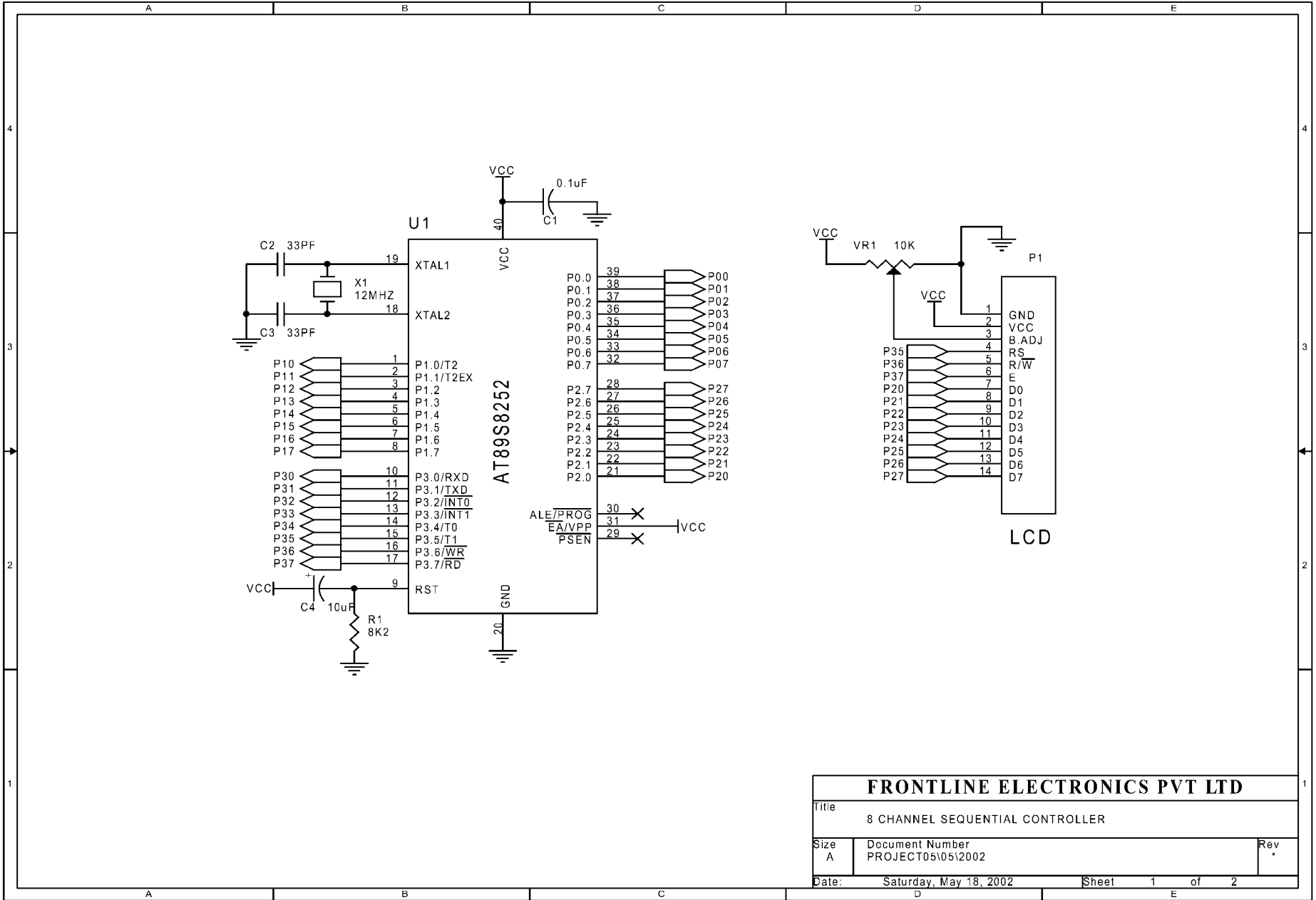
As you can observe, the LCD module is interfaced with the microcontroller using the Port 2 and 3. It is a very simple one: just connect the port lines with the LCD module. That's all. Display facility is incorporated into the design.

8 Channel Sequential Controller

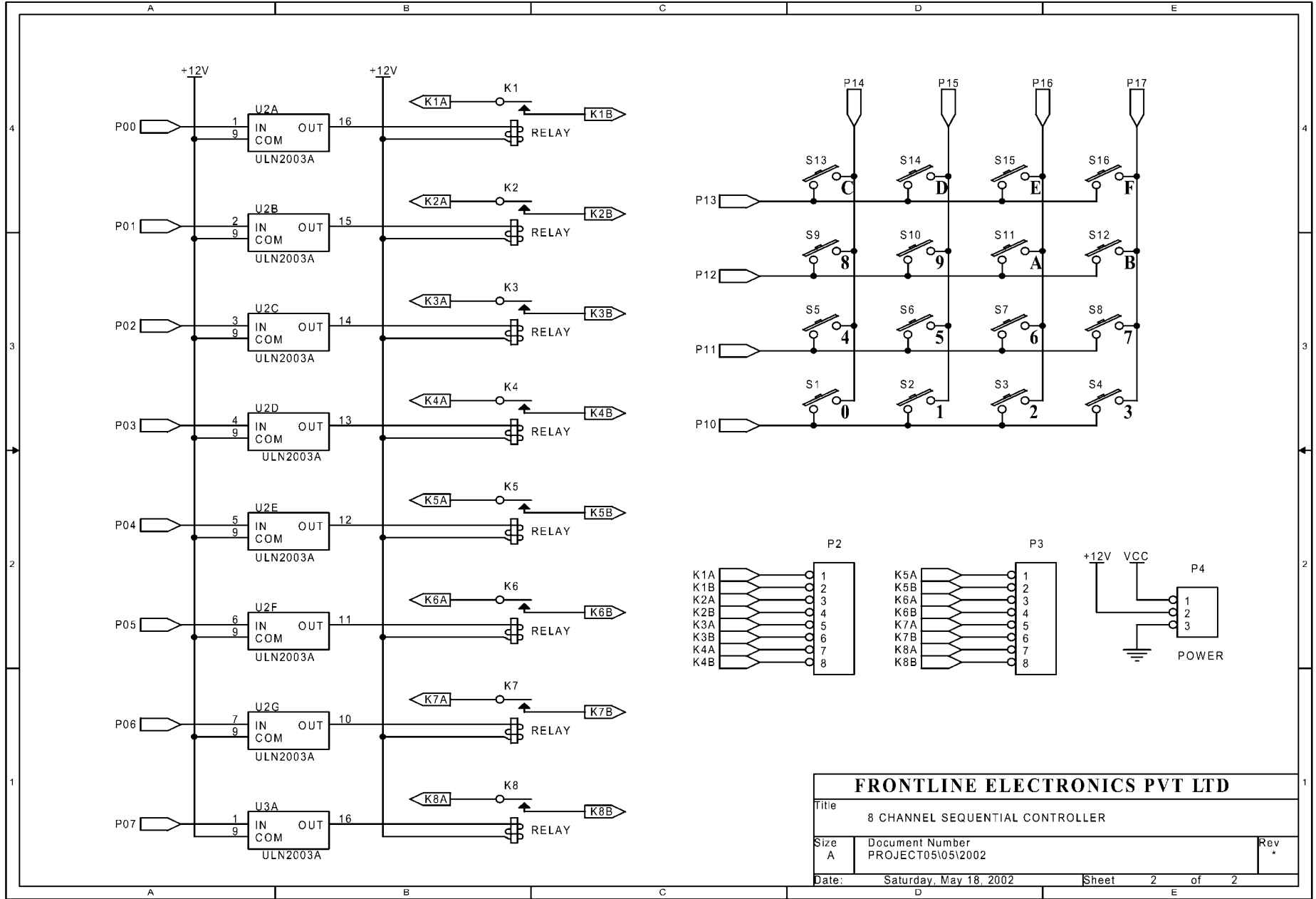
Then comes interfacing keyboard. The keymatrix with four columns and four rows is connected directly with the Port 1 of the microcontroller.

The control part sports a seven line driver using ULN 2003 driver device. This driver is interfaced with Port 0 and controls the operation of relays, solenoids, lamps, and etc.

So, the microcontroller switches on any of these output devices for a specific time or sequentially activates all the seven devices one by one as per the timing requirement.



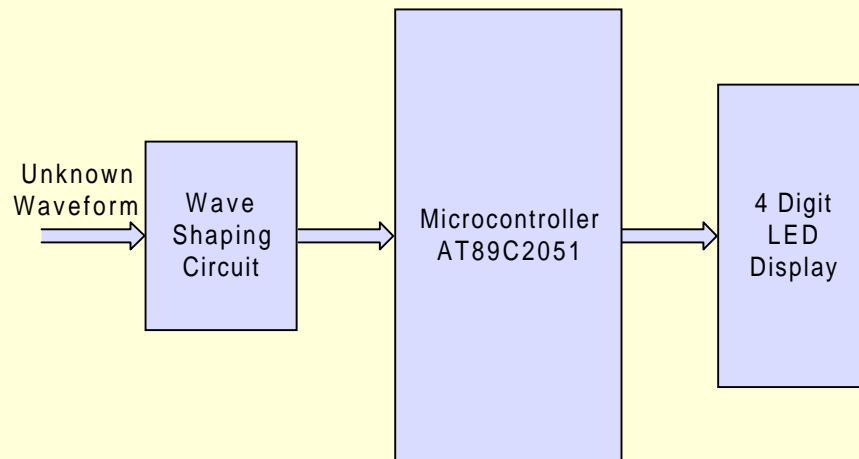
FRONTLINE ELECTRONICS PVT LTD		
Title 8 CHANNEL SEQUENTIAL CONTROLLER		
Size A	Document Number PROJECT05\05\2002	Rev *
Date: Saturday, May 18, 2002	Sheet 1	of 2



4.8 - Frequency Counter

This frequency counter is another simple and useful project meant for measuring unknown frequency. We embedded this counter in one of our educational product where it was used to measure signal frequency of the local oscillator.

The main parts of the design are a wave shaping circuit for the incoming signal and a display portion along with the basic microcontroller as indicated by the block diagram.



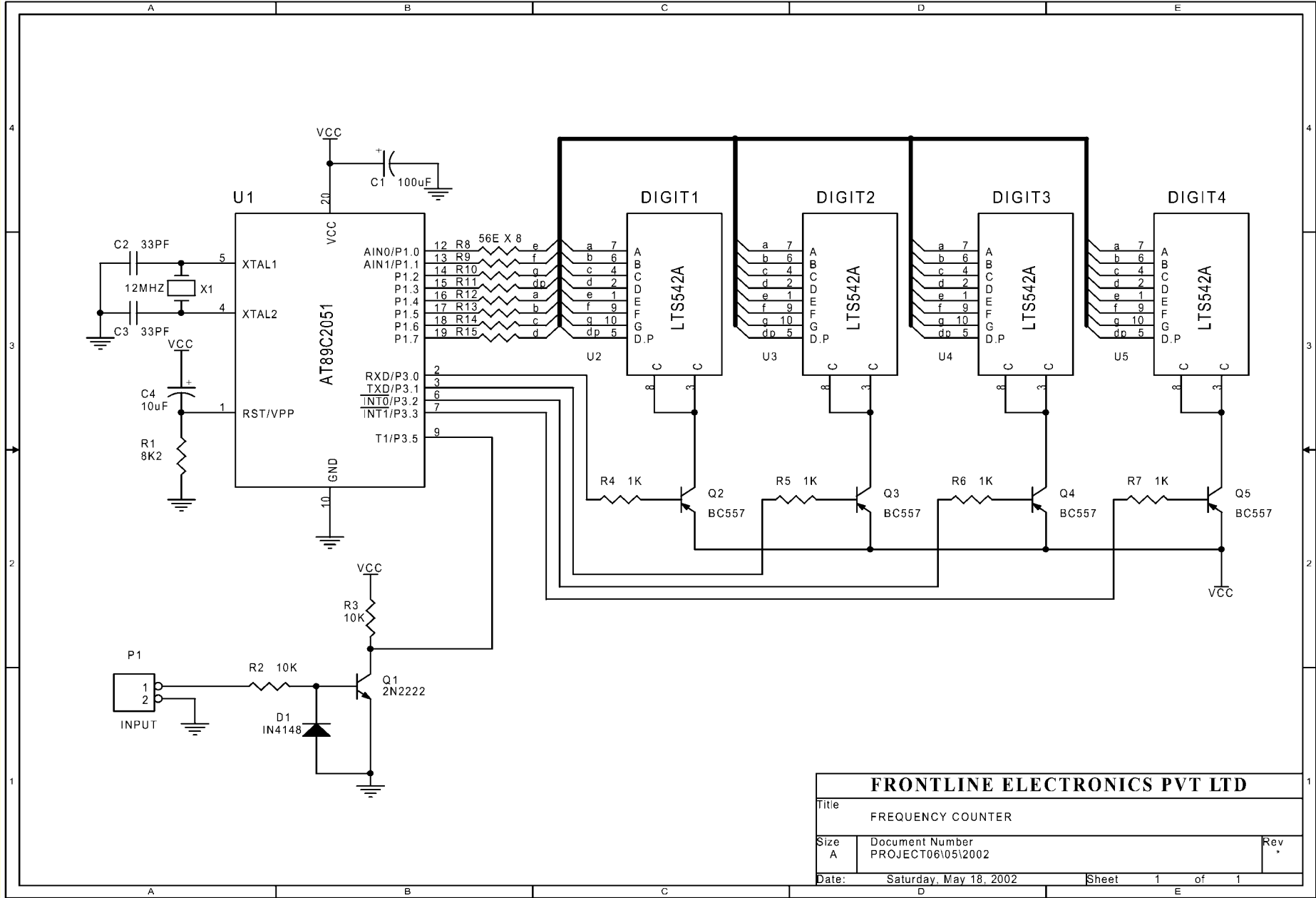
The incoming signal is conditioned by the wave shaping circuit and the input signal is converted into a square wave compatible to the input pin of the 8031's counter.

In principle, the counter averages the square wave signals it is getting for the period of one second and then do a simple calculation. The result is sent to the display.

The display contains multiplexed 4 digits of 7 segment LED displays connected to the port lines.

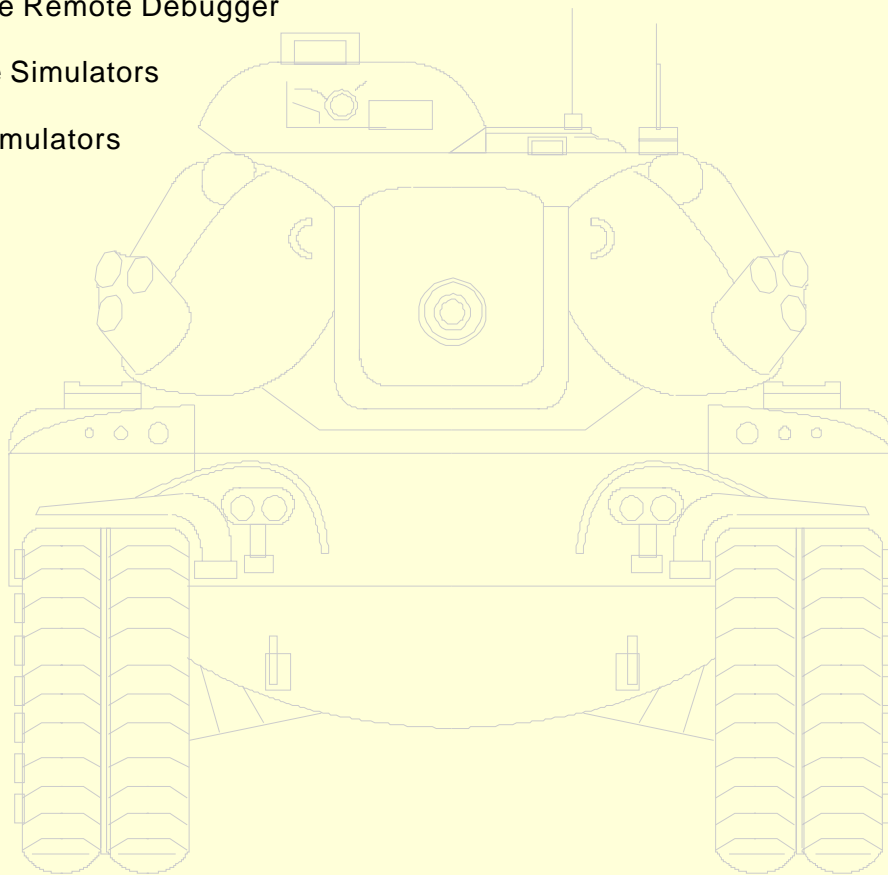
All the segment lines of these LED digits are connected together and then interfaced with the Port 1. Each digit is selected by the transistor driver connected with the Port 3.

Since the I/O requirements of this project has turned out as very simple, the low end Atmel device, AT 89C2051 is selected. The device comes in a 20 pin package with 15 I/O lines which are more than required for this application.



FRONTLINE ELECTRONICS PVT LTD		
Title FREQUENCY COUNTER		
Size A	Document Number PROJECT06\05\2002	Rev *
Date: Saturday, May 18, 2002	Sheet 1	of 1

- Software Options
- Stand Alone Device Assemblers
- Stand Alone Remote Debugger
- Standalone Simulators
- In Circuit Emulators



5. - Project Tools

I hope now you feel confident enough to take next big step in the Embedded System Design. You should start planning for your own embedded application. Consider a project based on ATMEL'S 89CXX device (in standalone configuration).

Naturally you have to arm yourself with required development tools before taking the plunge into the design step.

Let me tell you the general options available to you when you consider the development tools:

5.1 - Software Options

- Integrated Development Environment.
- Standalone Remote Debuggers.
- Standalone Simulators.
- Device Assemblers.

Of all these, IDE is the most sophisticated and useful tool suite that contains a kind of high level C compiler, debugging tool and other optimization tools. But other-side is that this kind of IDEs normally cost a good fortune and stays beyond the reach of individual professional designers, hobby enthusiasts and first time system designers. Even the low-end tool suite costs more than 1,000 \$ which discourages small companies.

On the positive side, you can develop your project using C language and you can get access into other integrated tools like Simulator, Remote debuggers, Optimization tools etc to get a quick start into your project.

But I personally recommend starting your first projects using assembly language and you can try using C once you are comfortable with the all the stages of program development.

I agree that developing your project takes more time than using the C. But the exercise enables you to get close with the target device and it will become friendly to you very soon. Also you can understand your program in a more productive way and eventually you can get tight control over the code that is going into the device.

Finally I draw your attention that many of the mass produced commercial, consumer products are developed using assembly language rather than any high level language.

Software options

So it helps you to get more confidence and save good money if you start your next design using assembly language.

5.2 - Stand Alone Device Assemblers

These assemblers are cross assemblers using the standard personal computers. Most of the time, the device manufacturer gives away a kind of usable assembler free of cost to encourage users to try it for their projects. Well. Atmel has get one assembler made available in its Web site. Get hold of that for your project.

May be you can use any standard text editor to compose your assembly language program and then export it into the assembler to get the required system code which is in Intel Hex format. Some assemblers might have this text editor built in the assembler software itself.

I hope that I need not tell you about the repeated assembling of your target code to remove all bugs, which might have crept into your code during program development.

5.3 - Stand Alone Remote Debugger

This one is very important part of your development process. Basically the debugger gives you the required means to look into your target code when it is getting executed in the microcontroller. It enables you to watch how the constants of various registers, memory area and other parts of the microcontroller vary during the code execution. You should get a quick glimpse of how the target code behaves inside the controller.

May I ask you to think about getting along with your project without debugger for few minutes? You may program your microcontroller with the target code and put it in the application PCB and switch on the power supply praying to all of your favorite gods. Well... many times you may not see the hardware working as you expected it to work. So you need to take the device for another round of reprogramming and rechecking. Even after many tries, you may not get to know the actual working of your code inside of the microcontroller.

This is how you get frustrated with your design capability and as you keep redoing same steps again and again you run into the risk of loosing your confidence little by little.

Now you can appreciate the usefulness of this kind of debugger in any development environment. Proceeding with any design without debugger will be similar to going ahead on a long journey on your foot.

Stand Alone Remote Debugger

Debugger is the most important vehicle you should board for any serious development.

Remote debugger is the two-part software and one sits into your microcontroller and other part gets executed in the personal computer. The part that stays in the microcontroller keeps track on the internal working of the microcontroller and keeps on sending all relevant information to the personal computer for your reference.

The personal computer in turn displays this information in a GUI environment for the complete and required insight on your target code.

Apart from this useful displays, the debugger allows you to control your program code in many ways. You can execute the code up to a certain break point, you can execute the code instruction by instruction in single step, you can set a register to display or a memory location to display the changes happening in the content for tracking purpose. Also you can view the contents of many registers, data, program memory areas.

So all these facilities will help you to understand the working of your target code in the actual hardware.

To get this debugger working properly, you should check few small things both in software and hardware as per guide lines mentioned. Basically you need to establish a kind of reliable communication facility to enable the personal computer to interact with the target hardware. Most of the time, a close inspection of the hardware (or PCB layout) in the serial port area will solve your communication problems.

All IDEs should have this debugging capability built in and many standalone Remote Debuggers are also available at an affordable cost.

5.4 - Standalone Simulators

This is another useful part of your tool collection. Most of the time these simulators act standalone and works in the personal computers. These simulators simulate the actual target micro controller in the personal computer.

The simulator software faithfully simulates the total microcontroller in the personal computer. Again, it is a GUI based environment enables you to get the feel of the microcontroller without waiting for the actual and physical hardware. In this world of Flash memory based microcontrollers, this kind of simulators enables you to find the feasibility of getting the design done with that controller without even seeing the physical hardware.

Standalone Simulators

Just like the debugger, you can export your assembly program in to the simulator and observe the changes going in the different parts of the device architecture.

You can also control the program execution in many ways. Like single stepping, execution up to a certain break point, executing the code in a single stroke etc.

The GUI windows give many other useful information about various registers, CPU registers, memory areas, and other peripheral control registers.

You can also find out execution timing information in time critical applications.

Latest simulators like 'Topview Simulator' even allows you to control all the peripheral functions of the microcontroller like I/O triggering, timer/counter operations, communication facilities like serial ports during the target code simulation.

As you can see, these simulators virtually enables you to complete the whole embedded system design in the simulator itself without even bothering about getting the target hardware. This facility can definitely help you to plan your project well in advance and guide you till you complete the design process.

5.5 - In Circuit Emulators

You can also find many hardware based tools, In Circuit Emulators, meant for many microcontrollers. These tools are based on a testing hardware along with debugging software. Normally this hardware takes charge of the target PCB hardware and captures all the signals at most of the microcontroller pins. The debugging part builds up required information from these signals.

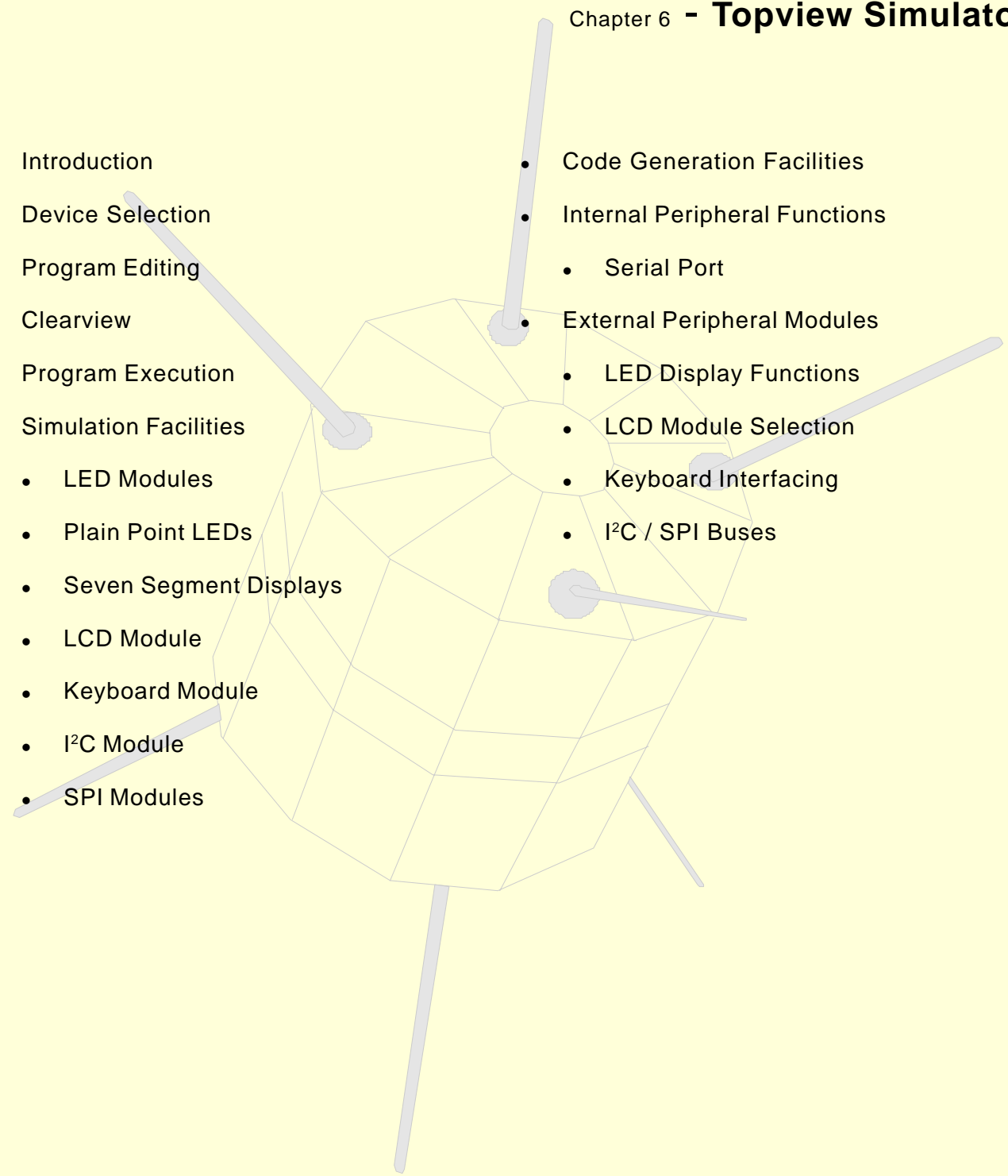
Most of the time, these debuggers work like remote debuggers and almost do same kinds of tasks.

Because of the associated hardware, the emulators have become the costliest tool for any microcontroller.

Normally, for 8 bit system designs based on 8031 microcontrollers, the need for this kind of emulators can be substituted with the remote debuggers.

But for other high-end microcontroller designs starting from 16 bit, the use of hardware emulators will definitely help any serious system designer in many productive ways.

Chapter 6 - Topview Simulator

- 
- Introduction
 - Device Selection
 - Program Editing
 - Clearview
 - Program Execution
 - Simulation Facilities
 - LED Modules
 - Plain Point LEDs
 - Seven Segment Displays
 - LCD Module
 - Keyboard Module
 - I²C Module
 - SPI Modules
 - Code Generation Facilities
 - Internal Peripheral Functions
 - Serial Port
 - External Peripheral Modules
 - LED Display Functions
 - LCD Module Selection
 - Keyboard Interfacing
 - I²C / SPI Buses

6.1 - Introduction

Topview Simulator gives an excellent simulation environment for the most popular 8 bit Microcontroller family, MCS 31. It is the total simulation solution giving many state of art features meeting the needs of the designers possessing different levels of expertise. If you are a beginner, then you can learn about 8031 based embedded solutions without any hardware. If you are an experienced designer, you may find most of the required facilities built in the simulator that enable you to complete your next project without waiting for the target hardware.

As a result, you complete your next embedded solution in record time without leaving the software.

The simulator is designed by the active feedback from demanding industrial designers and when you use this in your next 8031 project, you are assured of definite savings in time and increase in productivity.

The features of the simulator is briefly tabulated here for your reference:

Device Selection

A wide range of device selection, including Generic 8031 devices and Atmel's AT89CXX series 8031 microcontrollers.

Program Editing

Powerful editing feature for generating your programs and the facility to call an external Assembler to process input programs.

Clearview

Clearview facility gives all the internal architectural details in multiple windows. Information about the Program, Data Memory, register, peripherals, SFR bits are clearly presented in many windows to make you understand the program flow very easily.

Program Execution

A variety of program execution options including Single Stroke full speed execution, Single Step, Step Over and Breakpoint execution modes give you total control over the target program.

Clearview updates all the windows with the correct and latest data and it is a convenient help during your debugging operations.

You may find how this Topview Simulator simplifies the most difficult operation of the program development, debugging, into a very simple task.

Simulation Facilities

Powerful simulation facilities are incorporated for I/O lines, Interrupt lines, Clocks meant for Timers Counters.

Many external interfacing possibilities can be simulated:

- Range of Plain Point LEDs and Seven Segment LED options.
- LCD modules in many configurations.
- Momentary ON keys.
- A variety of keypads upto 4 X 8 key matrix.
- Toggle switches.
- All modes of onchip serial port communication facility.
- I²C components including RTC, EEPROMs.
- SPI Bus based EEPROM devices.

Code Generation Facilities

Powerful and versatile Code Generating facility enables you to generate the exact and compact assembly code for many possible application oriented interfacing options.

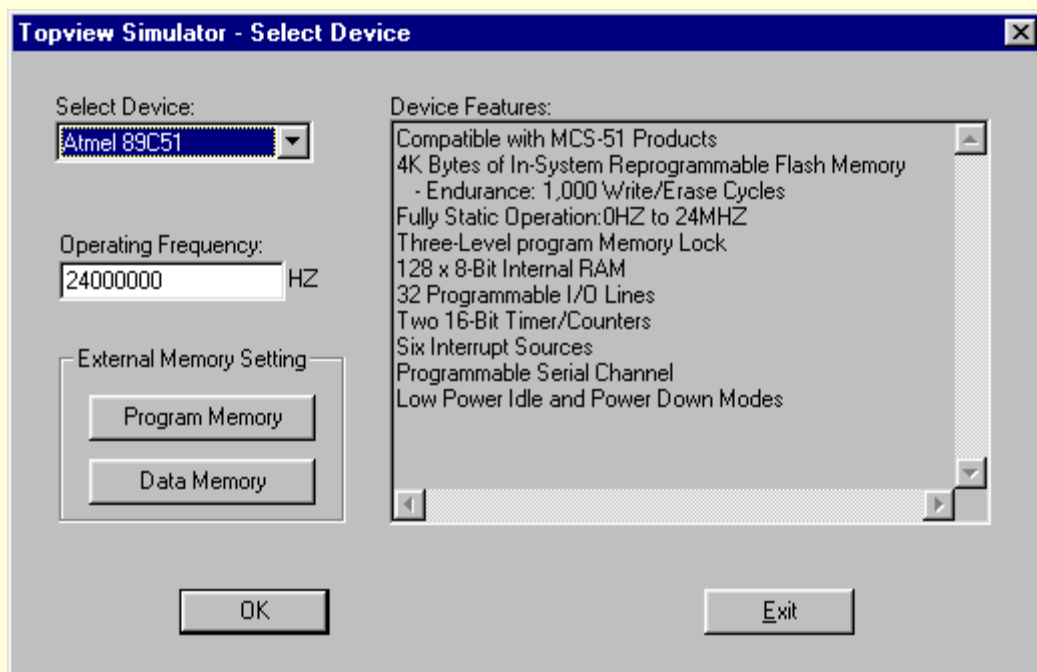
You can simply define your exact needs and get the target assembly code at a press of button at anywhere in your program flow. The code gets embedded into your application program automatically.

You are assured of trouble free working of final code in the real time.

- All modes of the serial port.
- Interfacing I²C/SPI Bus devices.
- Range of keypads.
- Many LED/LCD interfacing possibilities.

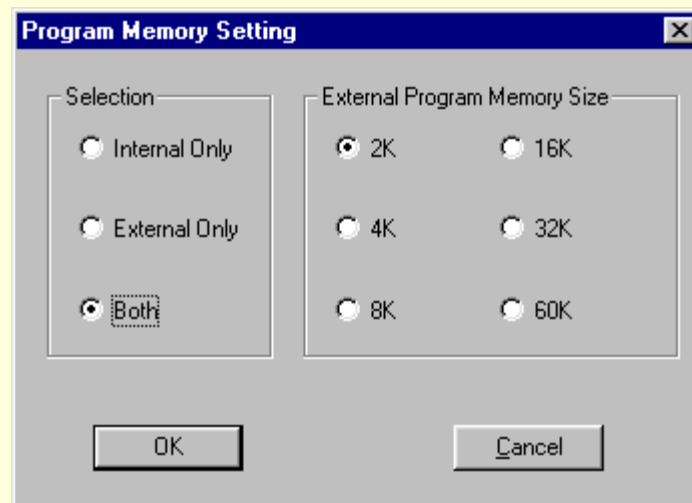
6.2 - Device Selection

- Simulation of Generic 8031 Microcontrollers:
8031, 8051, 8032, 8052.
- All 8031 microcontroller devices of Atmel's AT89CXX Family:
89C1051, 89C2051, 89C4051, 89C51, 89C52, 89C55, 89S53, 89S8252.

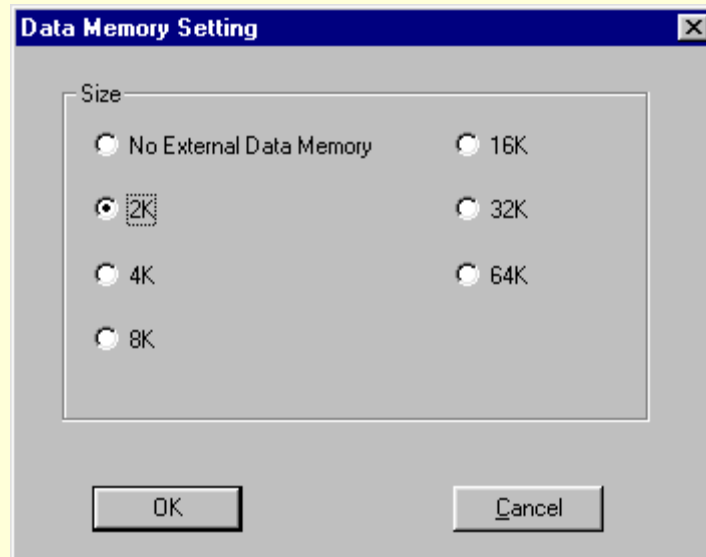


When you select a particular controller, the Topview will be automatically configured to the features of the device. You need not worry about memory, available ports, internal peripherals and etc because the enabling and disabling the features are handled by the simulator itself.

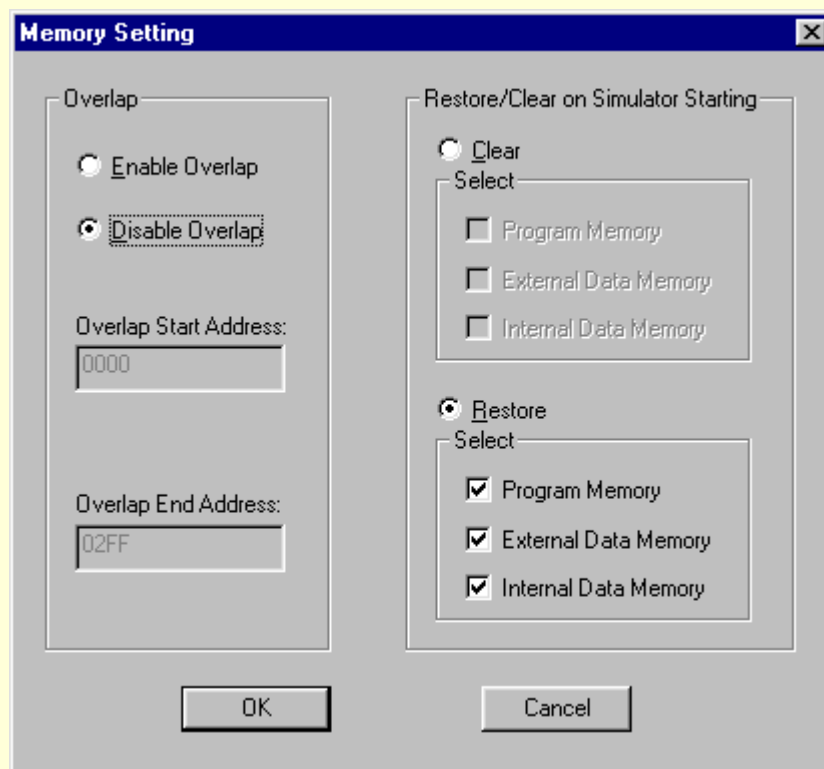
- Facility to simulate 8031 in single chip and expanded configuration.
- Provision to set address range of both external Program and Data Memory, Crystal frequency.



Device Selection



- Facility is provided to overlap a block or whole of Program and Data Memory spaces in expanded configuration.



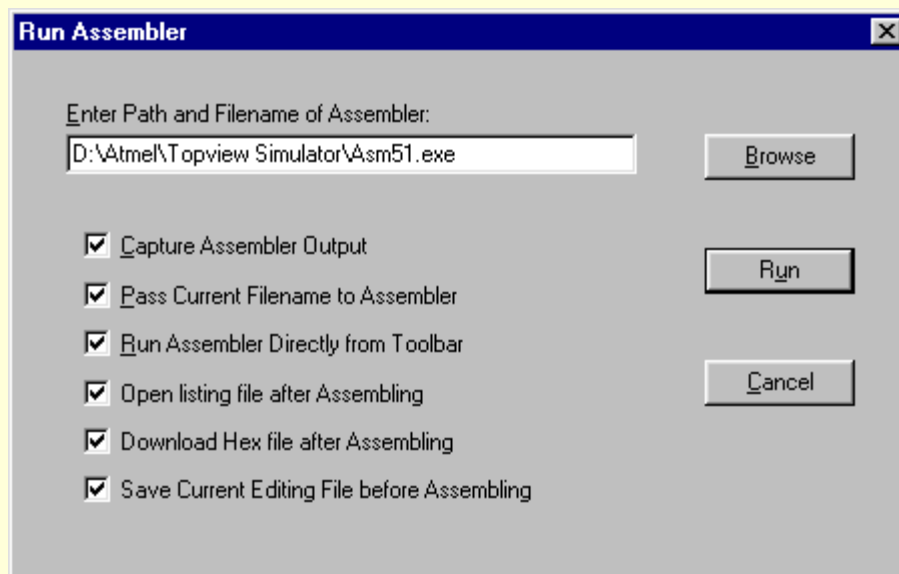
6.3 - Program Editing

The simulator sports a powerful built-in editor to support program entry operations. It can call external assembler software and pass the edited program into that for assembling.

The assembler output can be captured and viewed along with your input program in editor. Also viewing the list file (coming out of assembler) by the side of input assembly file enables you to view the details of error and the location. You can easily correct input file errors before assembling it again. Useful feature when troubleshooting the program.

There is an optional facility to load the assembled program into the specified Program memory.

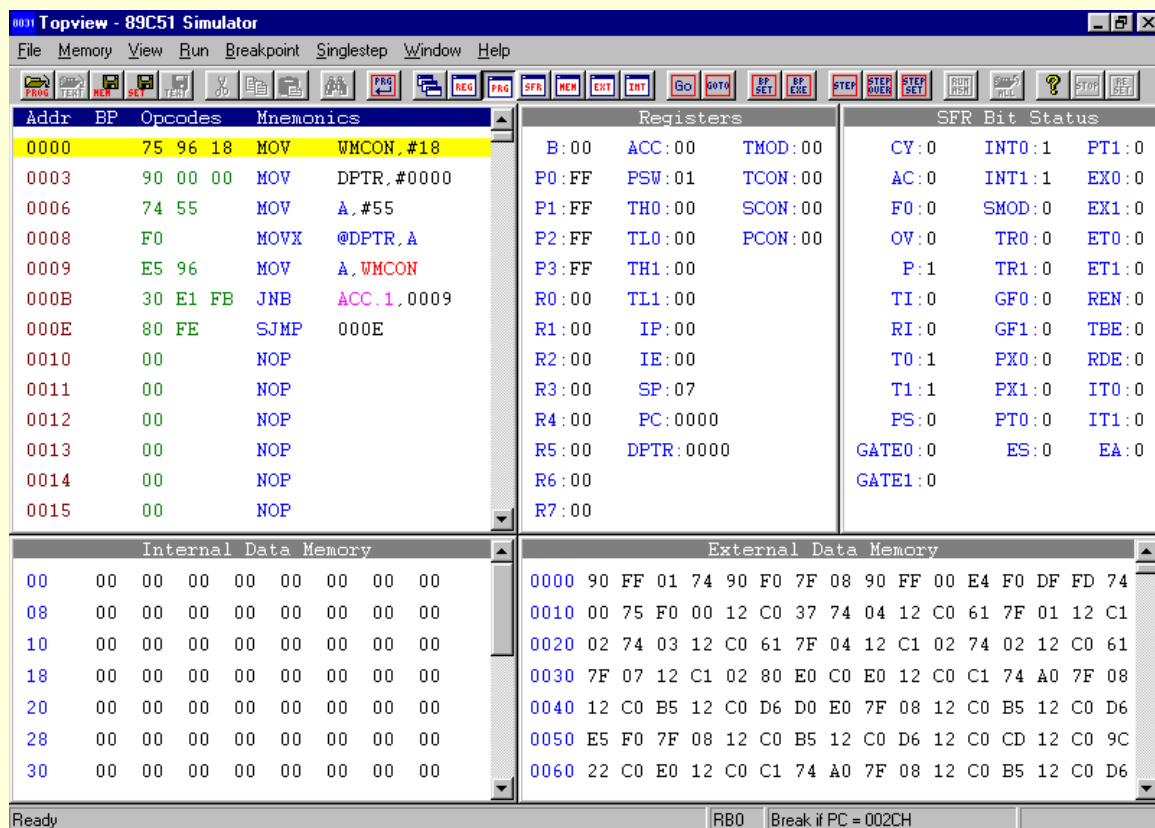
Benefit: Saves time during debugging phase.



6.4 - Clearview

Extensive GUI features are available thanks to clearview presentation. Separate windows are available to display,

- Program Memory.
- Internal Data Memory.
- External Data Memory.
- Registers.
- Memory Bits and
- Special Function Registers.



This multi window clearview capability indicates the total picture of the microcontroller's internal operations at any point of time for your convenience.

There is a single line assembling facility available when editing programs in Program Memory window. You can view Mnemonics, Address, SFR Bits, SFR Registers in different colours.

In Data Memory window you can view and edit data in either Hex code or ASCII.

Benefit: You can simply define alphanumerical messages in ASCII mode. Useful feature when using LCD modules in the projects.

In Register, SFR windows, the name of the registers, SFRs are indicated to make you understand program flow easily. All the registers, SFR are tabulated for your convenience.

SFR Bit Status

Accumulator<ACC>:-

ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
0	0	0	0	0	0	0	0

B Register:-

B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
0	0	0	0	0	0	0	0

Program Status Word<PSW>:-

CY	AC	P0	RS1	RS0	OV	-	P
0	0	0	0	0	0	-	1

Port 0<P0>:-

P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
1	1	1	1	1	1	1	1

Port 1<P1>:-

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
1	1	1	1	1	1	1	1

Port 2<P2>:-

P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
1	1	1	1	1	1	1	1

Port 3<P3>:-

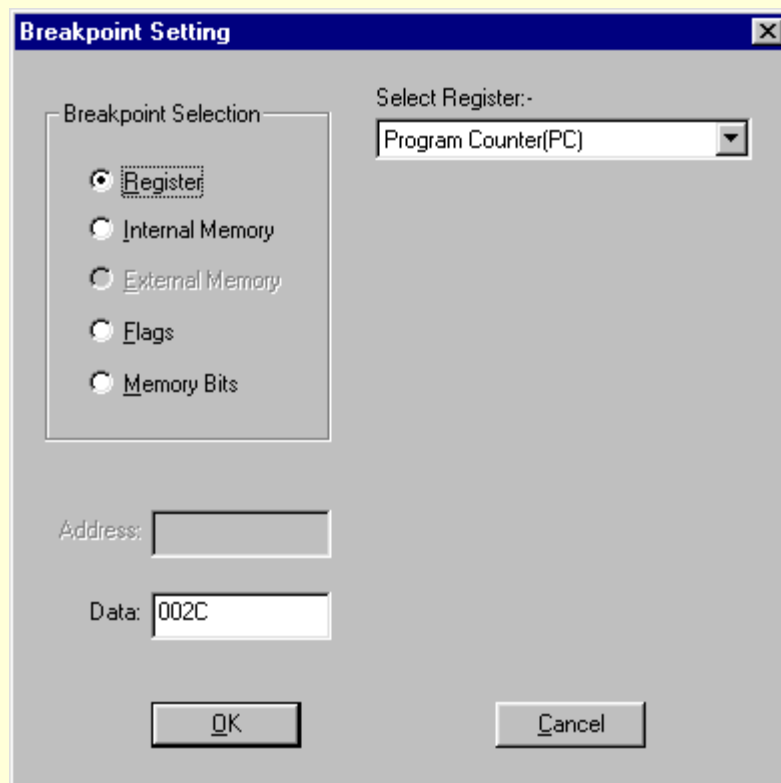
Ready RB0 No Breakpoint Set

6.5 - Program Execution

During execution, all the windows of clearview can be updated according to the time interval set by yourself.

Benefit: You can know the status of your program at any point of time.

Breakpoint execution can be based on the registers, internal Data Memory, SFR bits, Flags, external Data Memory and memory bit status.



The breakpoint conditions are displayed in the status bar for your reference. Usually the breakpoints are based on the PC contents. But in the Topview Simulator, facility is there to use other information as the breakpoint conditions.

After every execution (break point or full speed) the address at which the execution stops/breaks will be highlighted and all the windows are refreshed with the new information. The changes are indicated in a different color to draw your attention.

Benefit: If you stop program execution when the microcontroller hangs up, you can find out where exactly your program flow stays still. Useful handy feature for your program development.

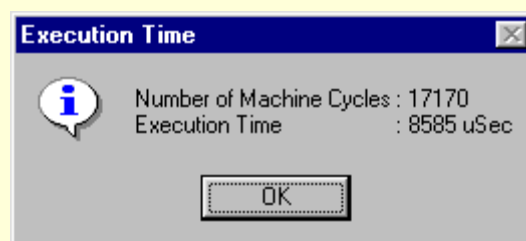
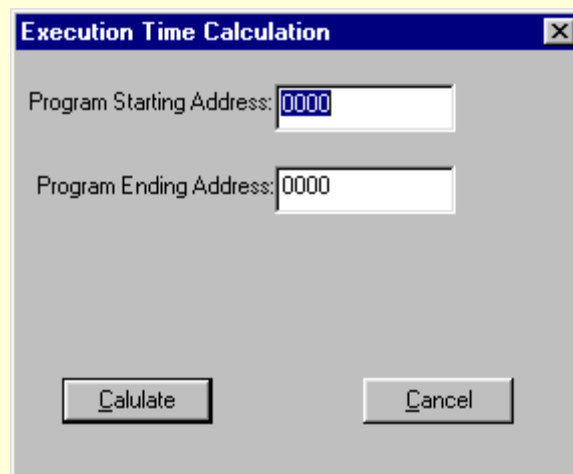
Program Execution

Different options are available in single step execution:

- Selection of either step or step over mode. Step mode executes program instructions one by one. In step over mode, subroutines can be executed in a single step.

A command in the singlestep operation helps to set the step size which is used by both step and stepover executions. The step mode executes the program line by line (or by the step defined by you). In the stepover mode, the subroutine is executed in a single step. In the ' Finish subroutine mode ' the control executes the program until the ' RET ' instruction is encountered and executes it and waits in the next instruction for further commands.

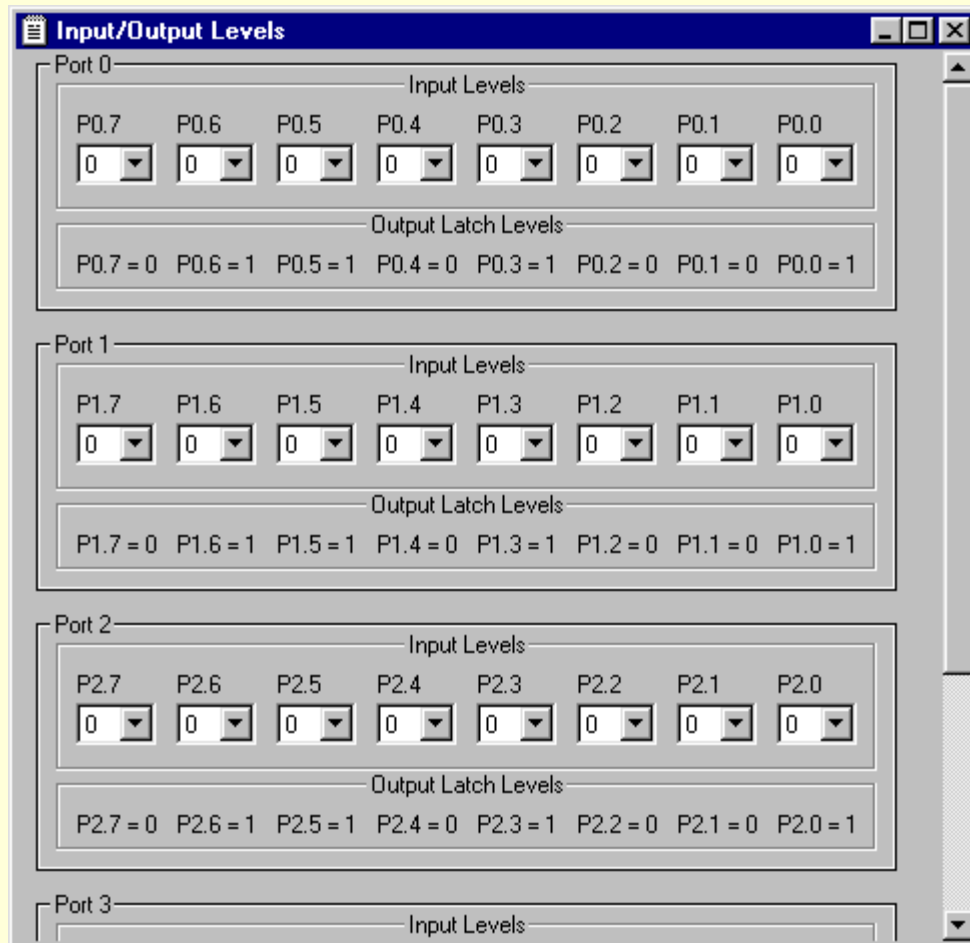
Useful timing analysis is available to calculate program execution time and machine cycles of the target program are accumulated and displayed for your reference.



Benefit: Accurate timing information about the program execution can be obtained for critical operations. For an example, you can accurately define delay routines.

6.6 - Simulation Facilities

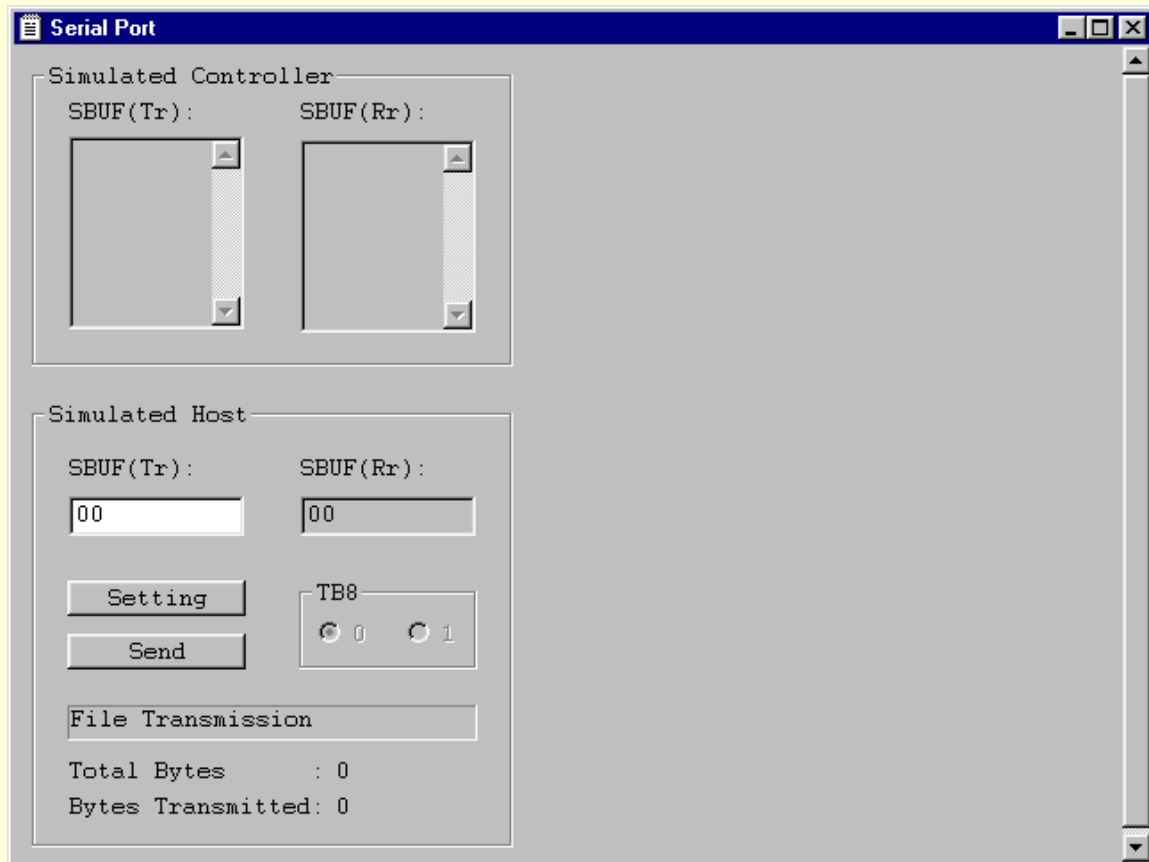
- I/O lines : During program execution, inputs can be set and you can watch output lines for desired results.



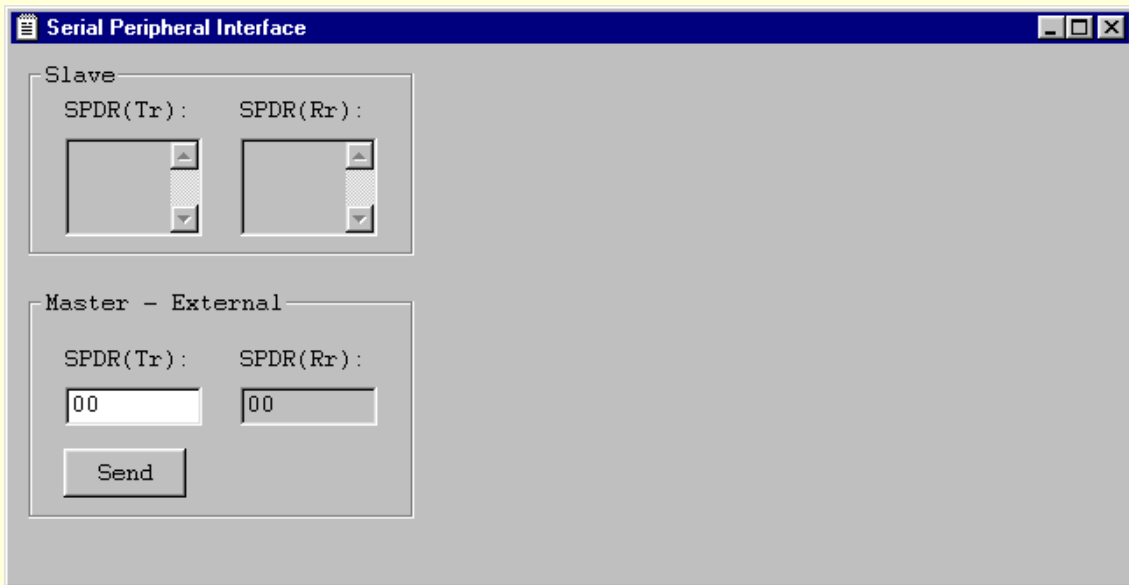
- Interrupt lines (both level and edge triggering) are also supported.
- Clock pulses can be simulated for Timers/Counters 1,2 and 3.
- For the AT89CX051 devices, Onchip Analog Comparator can also be simulated. Facility is there to give variable inputs to the comparator.
- For AT89S53, AT89S8252 devices, Watch dog timer is also be simulated.
- All modes of Serial port are simulated.

Topview simulates all possible four modes of the serial port including Multiprocessing Communication. There is a facility that enables you to send a test data byte out of the serial port and you can view that data going out of the serial port. Similarly you can view the data bytes received at the serial port.

- Separate buffers of size 256 bytes are provided for capturing both transmitted and received data bytes.



- These buffers can be saved separately for verification.
- During execution, data transmitted out of TXD line is captured and displayed continuously for your verification.
- Similar facility is available to monitor and accumulate data from RXD line.
- There is another serial port made available to simulate external host serial port. Using this facility you can make sure serial port of your target hardware works fine with a host computer. This can substitute personal computer's serial port and enables you to test and verify the serial port of the target design.
- The serial port sports the facility to send / receive data with the target hardware. you can either manually key in test data or you can send a data file through this simulated host serial port.
- Internal SPI module of AT89S8252 device can be simulated. An exclusive dialog box gives the facility for this.

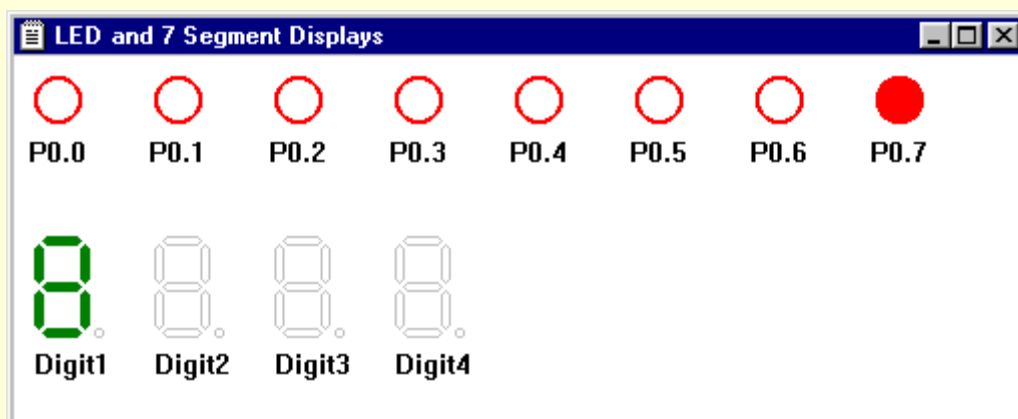


External peripheral modules can also be simulated. These modules can be connected to any available I/O lines. This should assist the user to develop and complete the project without any real target hardware.

6.6.1 - LED Modules

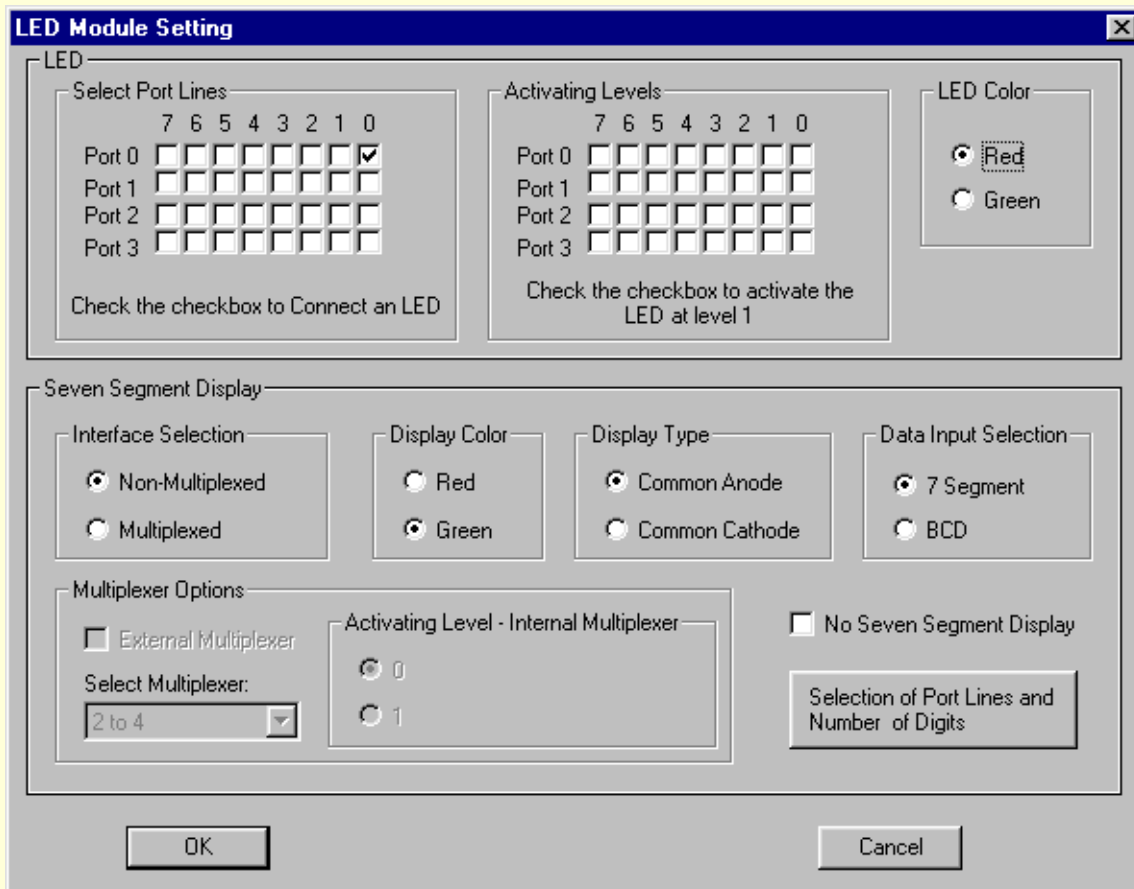
6.6.1.1 - Plain Point LEDs

- Color of LEDs. Red / Green can be defined.
- A maximum of 32 point LEDs can be graphically simulated.
- Activation level (either 1 or 0) can be defined.



6.6.1.2 - Seven Segment Displays

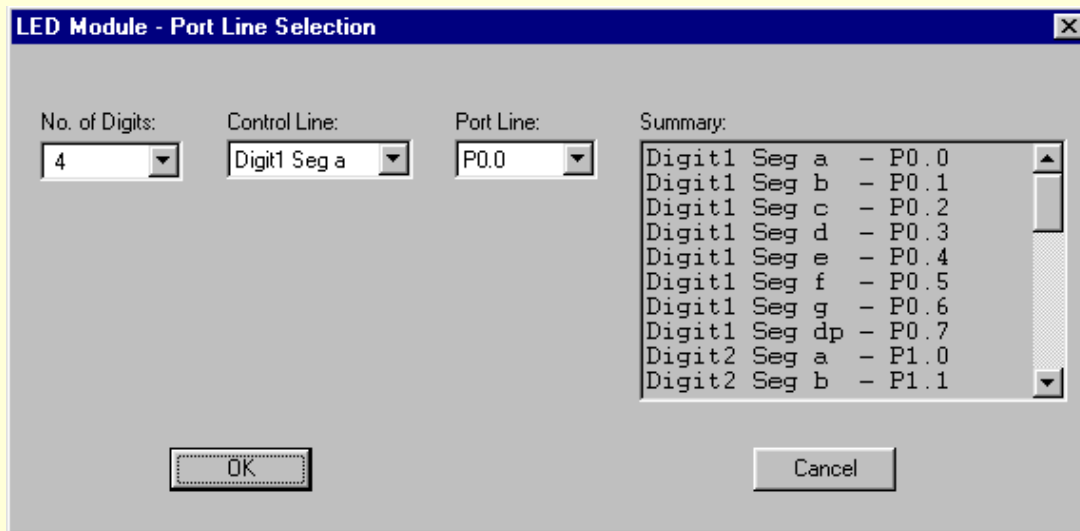
- 16 Digits of Seven Segment displays are graphically represented.
- Choice of display colors (Red/Green).
- Select Common Anode/Cathode displays.
- Seven Segment displays can be connected in any one of the following configurations:



- Non-Multiplexed displays with BCD inputs.
- Non-Multiplexed displays with Seven Segment Code inputs.
- Multiplexed displays with BCD inputs.
- Multiplexed displays with Seven Segment Code inputs.
- Multiplexed displays with BCD inputs using external multiplexer.
- Multiplexed displays with Seven Segment Code inputs using external multiplexer (upto 4 to 16 digits)

Maximum number of digits is defined by the configuration of the display module and the available port lines. There is no restriction on the port lines. Displays can be connected to available port lines of the microcontroller.

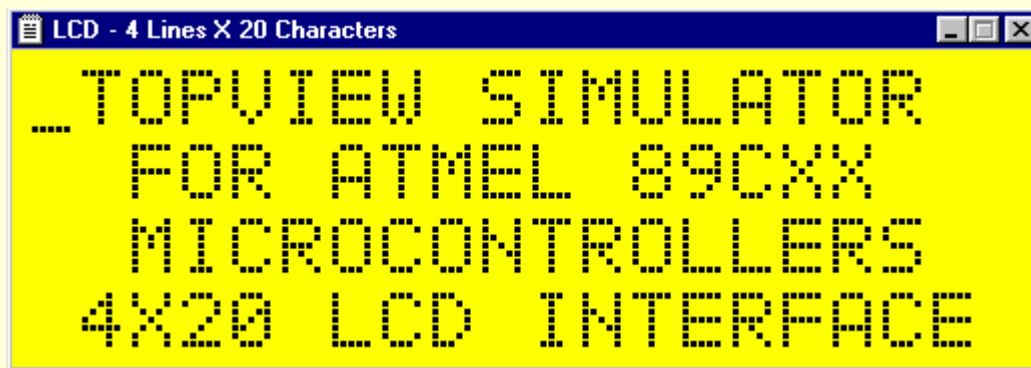
Seven Segment Displays



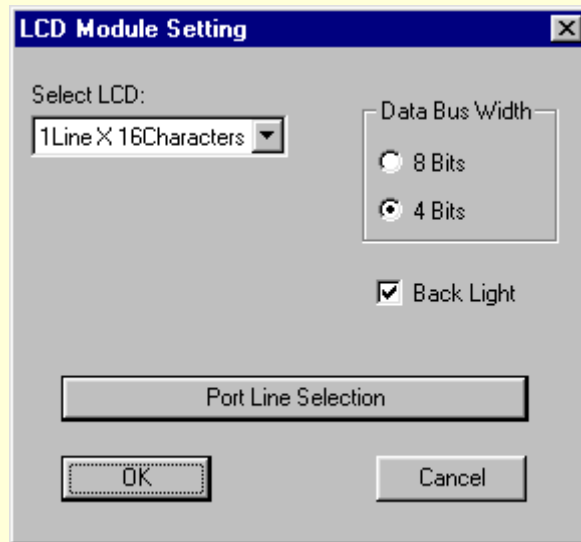
6.6.2 - LCD Module

The simulator gives actual and exact graphical view of the LCD modules similar to real life displays.

- Supported LCD modules: 1X16, 2X16, 4X16 and 4X20.



- Either 4 bit or 8 bit interface option can be selected.
- Facility to select the Back light of the LCD module.



6.6.3 - Keyboard Module

- A maximum of 32 Momentary ON type keys can be simulated.
- Active level can be selected (0 - 1 - 0 or 1 - 0 - 1)
- Key matrices of type 4X3, 4X4 and 4X8 can be simulated.



Keyboard Module

- Also toggle switches can be simulated. Maximum is 32.

Key matrix, toggle switches can be connected to any port lines as per your wish list.

Keyboard Module Setting

Keys - Momentary type

Select Port Lines

	7	6	5	4	3	2	1	0
Port 0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 1	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Check the checkbox to Connect a Momentary Switch

Activating Levels

	7	6	5	4	3	2	1	0
Port 0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 1	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Check the checkbox to activate the Switch at '1' level

Keys - Toggle Type

Select Port Lines

	7	6	5	4	3	2	1	0
Port 0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Port 1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Port 3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Check the checkbox to Connect a Toggle Switch

Matrix KeyPad

Select Keypad Type

4x3 4x8

4x4 None

Port Line Selection

OK Cancel

Matrix Keypad - Port Line Selection

Control Line: Row1

Port Line: P1.6

Summary:

- Row1 - P1.6
- Row2 - P1.7
- Row3 - P0.0
- Row4 - P0.0
- Column1 - P0.0
- Column2 - P0.0
- Column3 - P0.0

OK Cancel

6.6.4 - I²C Module

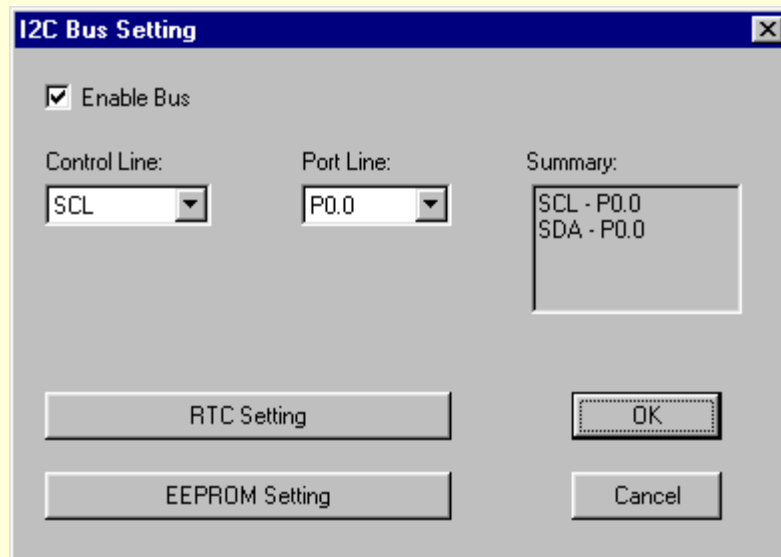
Complete external interface for two popular I²C devices can be selected:
I²C RTC, I²C EEPROM.

Devices that can be simulated : I²C RTC : Philips RTC - PCF 8583.
: I²C EEPROM : AT24C01, AT24C02, AT24C04,
AT24C08 and AT24C16.

The address lines, A0, A1 and A2 can be selected according to the device.

The device can be connected with any available I/O lines.

For EEPROM devices, write protect option can be enabled.



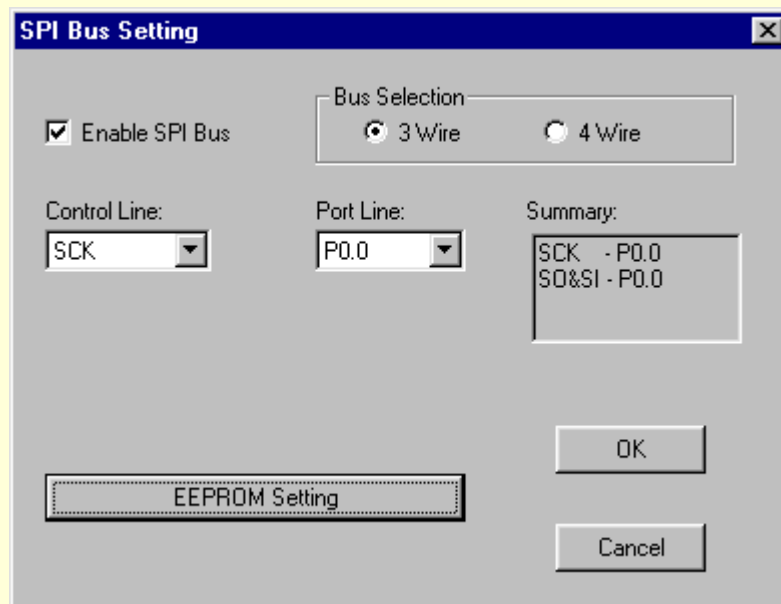
When the write protection is enabled the contents of the device will be displayed in different colour to indicate that the device is operating in ' Write Protect ' mode.

Special feature of I²C RTC simulation : You can use your computer's RTC timings as reference during RTC simulation.

Benefit: Handy feature when designing time dependent projects.

6.6.5 - SPI Modules

- All possible types of external SPI modules can be simulated and can be interfaced with any available I/O lines of the microcontroller.
- Either 3 or 4 wire configuration can be chosen.
- Any one of the EEPROM devices can be selected: AT 25010, AT 25020 and AT 25040 of Atmel.
- Write protection options can be enabled.



Both I²C and SPI EEPROM devices are simulated taking into account time required for data write cycle. This cycle may take upto about 10 milliseconds. You can't write any command or data into or read from the device until the write cycle comes to an end except when reading device status.

6.7 - Code Generation Facilities

Using this, you can generate required assembly code for all possible onchip peripheral functions and also many external peripheral functions. You can include these routines into your program.

You can do this in few simple steps. You keep open your editor and point the desired location using the cursor and then configure the peripheral function in the respective dialog box.

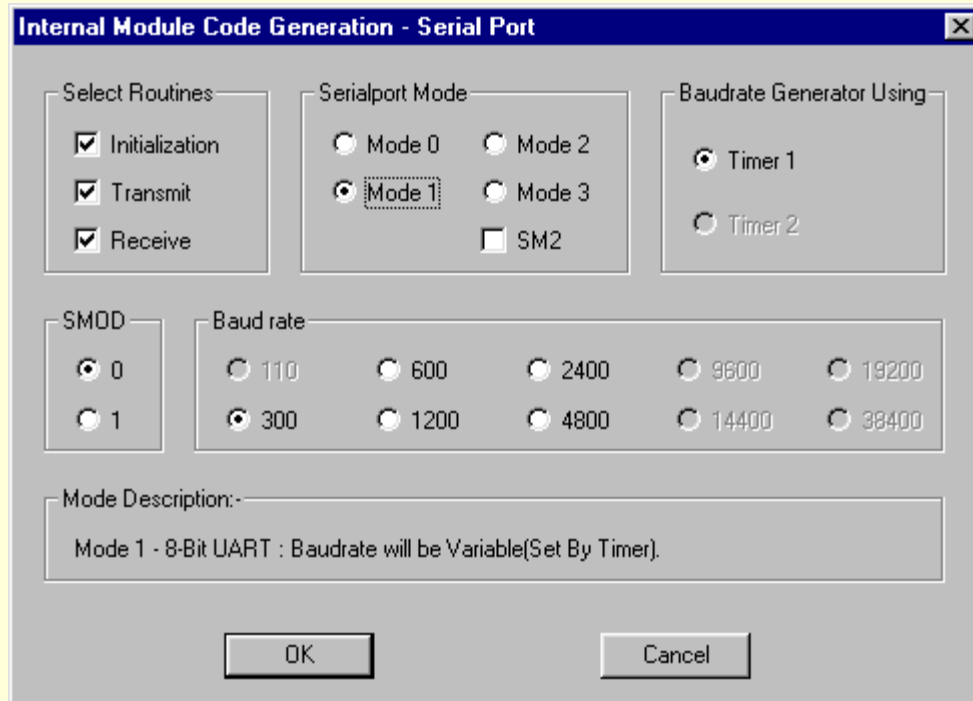
That's all. Now you press the code generate button to get the required assembly code at the defined place. Suitable comments are also made available for easy understanding. Then keep going with your target program flow.

Benefit: You are assured of fully tested and compact assembly codes which can be directly used in your target applications.

6.7.1 - Internal Peripheral Functions

6.7.1.1 - Serial Port

- Any one of the modes 0/1/2/3 can be selected.
- Depends upon the mode, crystal, the possible baud rates will be displayed for your convenience.
- Timer 1 or 2 can be selected for baud rate generator.
- Initialization, transmission and reception routines will be generated according to the given selection.



6.7.2 - External Peripheral Modules

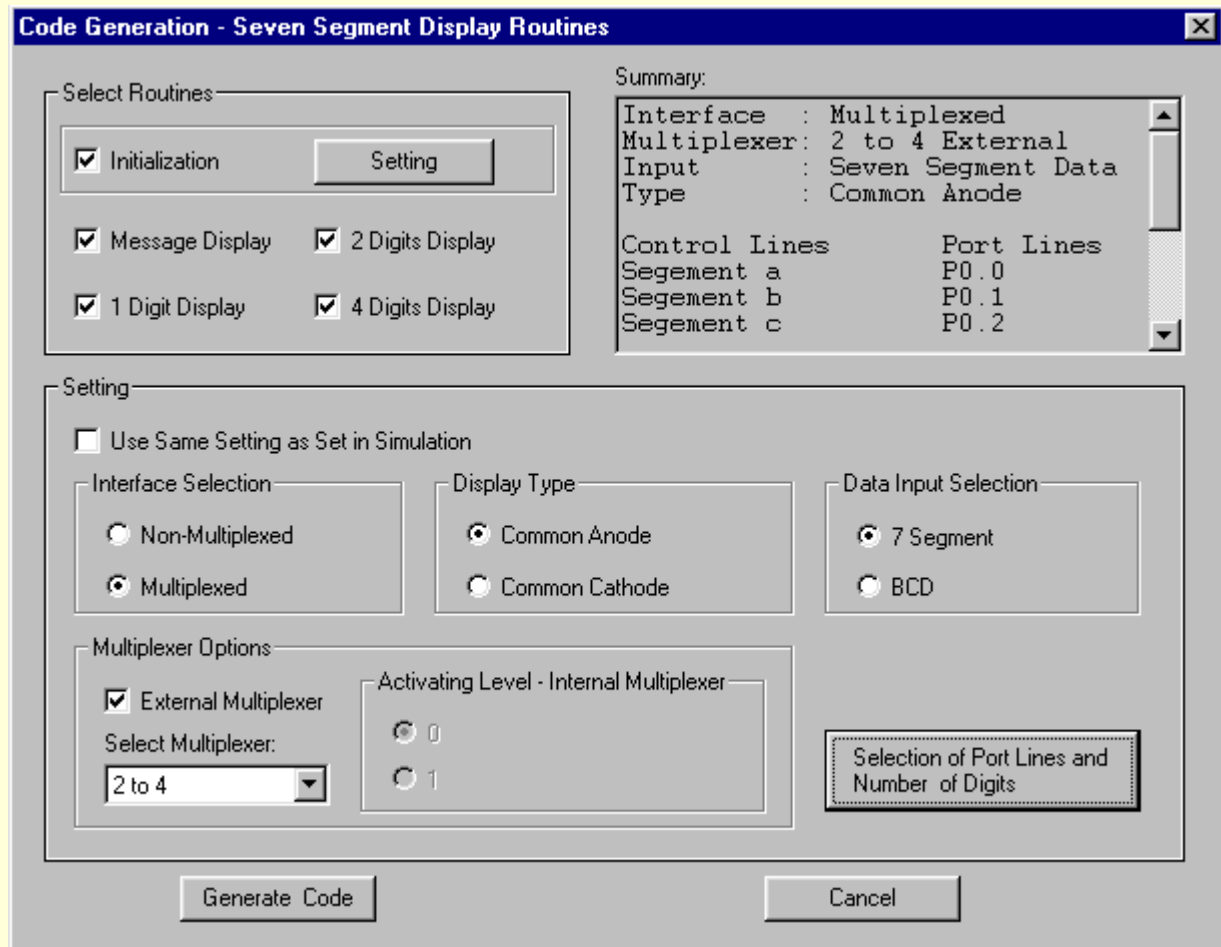
You can generate code snippets suitable for interfacing variety of external LED and LCD modules. You can activate a dialog box and define your exact needs by clicking relevant settings. You can use any available I/O lines for interfacing these modules. There is no restriction like using port lines (of same port) in order to get a LED/LCD function. Any I/O line from any port can be used to configure the required display function.

6.7.2.1 - LED Display Functions

- 1 / 2 / 4 Digits
- Multiplexed / Non Multiplexed.
- Common Anode / Cathode.
- Data Input options : BCD / 7 segment.
- Use of external multiplexer.

External Peripheral Modules

You can also notice the allotment of I/O lines to get the required LED function in the same dialog box.



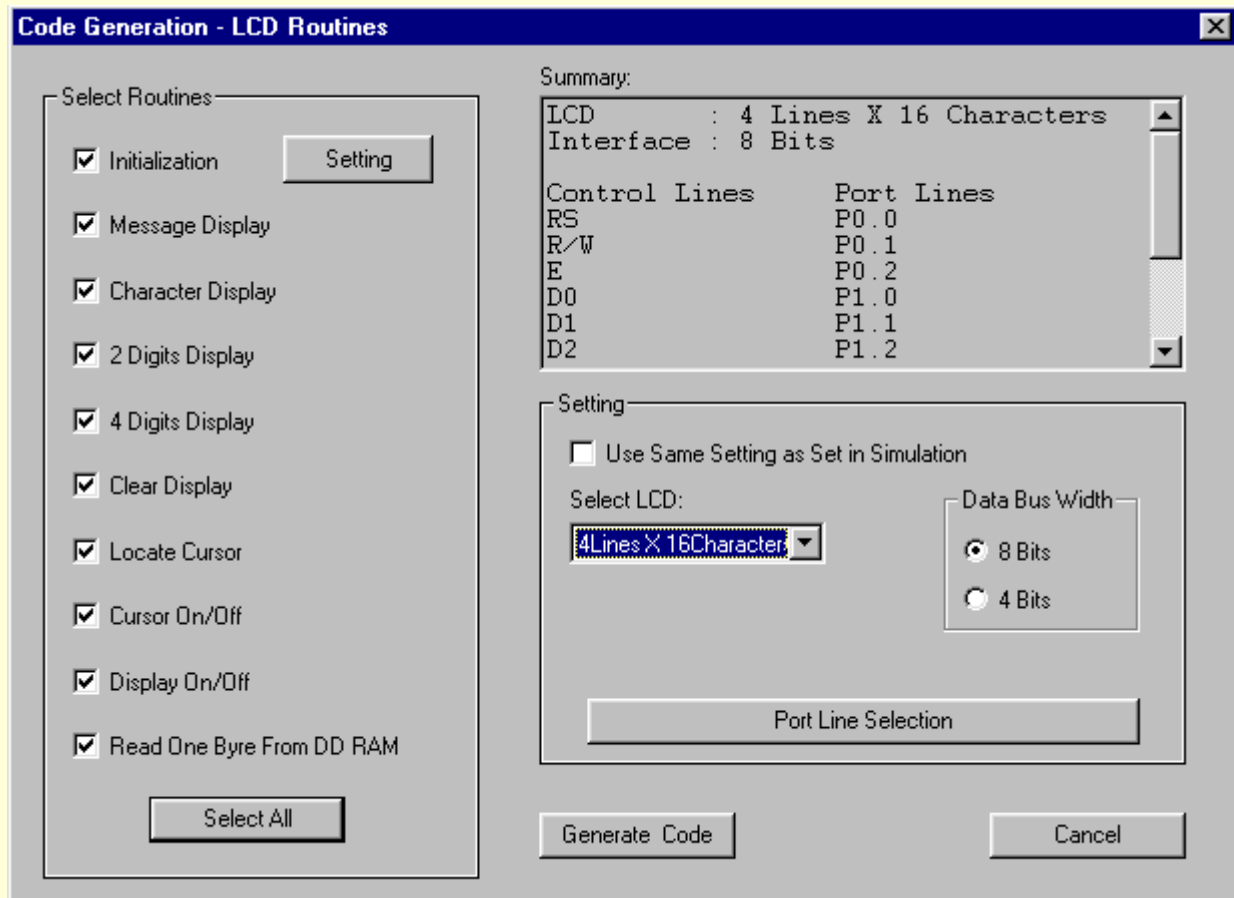
6.7.2.2 - LCD Module Selection

Just like LED options, you can make use of many possible modules suitable for LCD interfacing.

- LCD Type : 1 line X 16 Characters, 2 X 16 , 4 X 16 and 4 X 20.
- Interfacing options : 8 bits / 4 bits.
- Facility to keep same settings as you tried in simulator.
- All possible routines meant for LCD interfacing.

The dialog box gives all possible options. Just click your requirements and get the relevant assembly code.

LCD Module Selection



6.7.2.3 - Keyboard Interfacing

Using this facility, you can generate the required code for following key matrices:

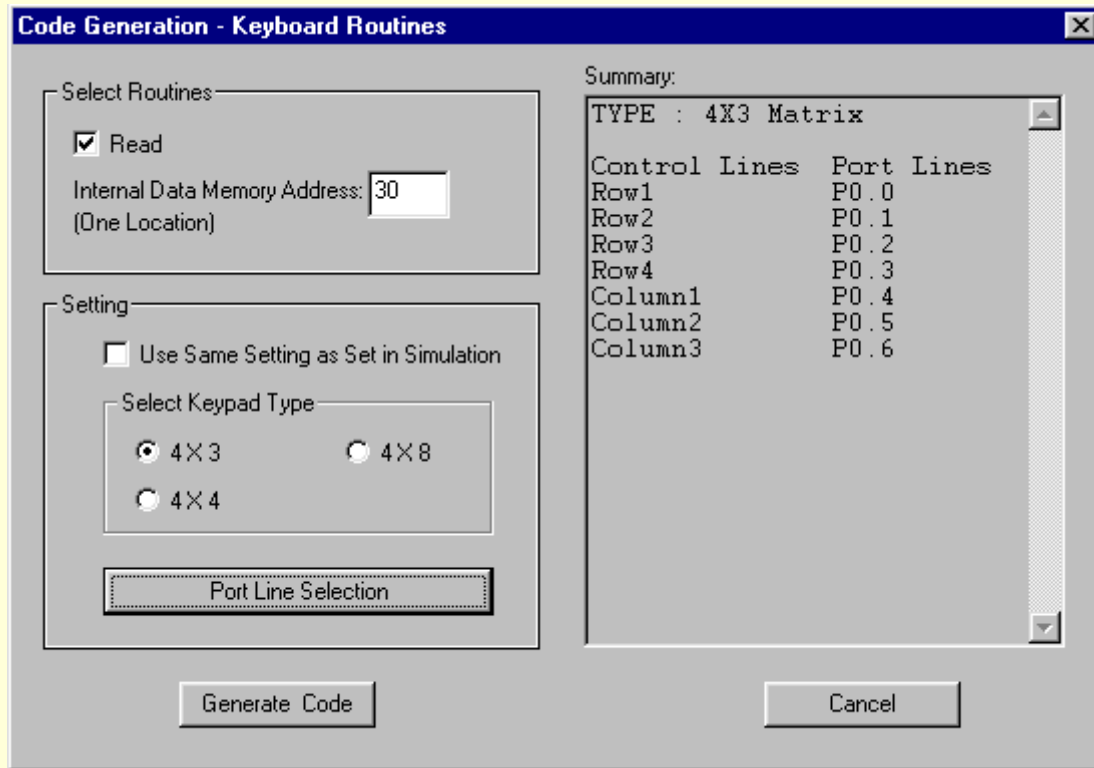
- 4 X 3
- 4 X 4
- 4 X 8

You also can define internal data memory location where you want to get the keyboard code.

You can use same simulator settings.

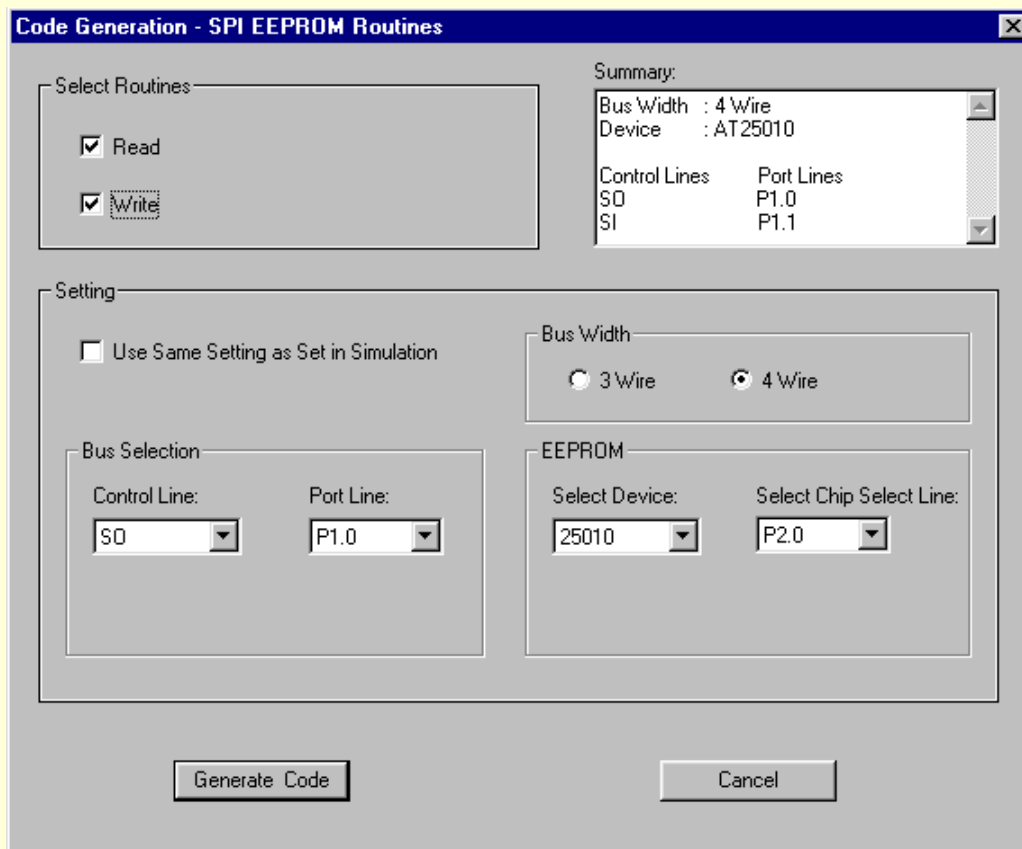
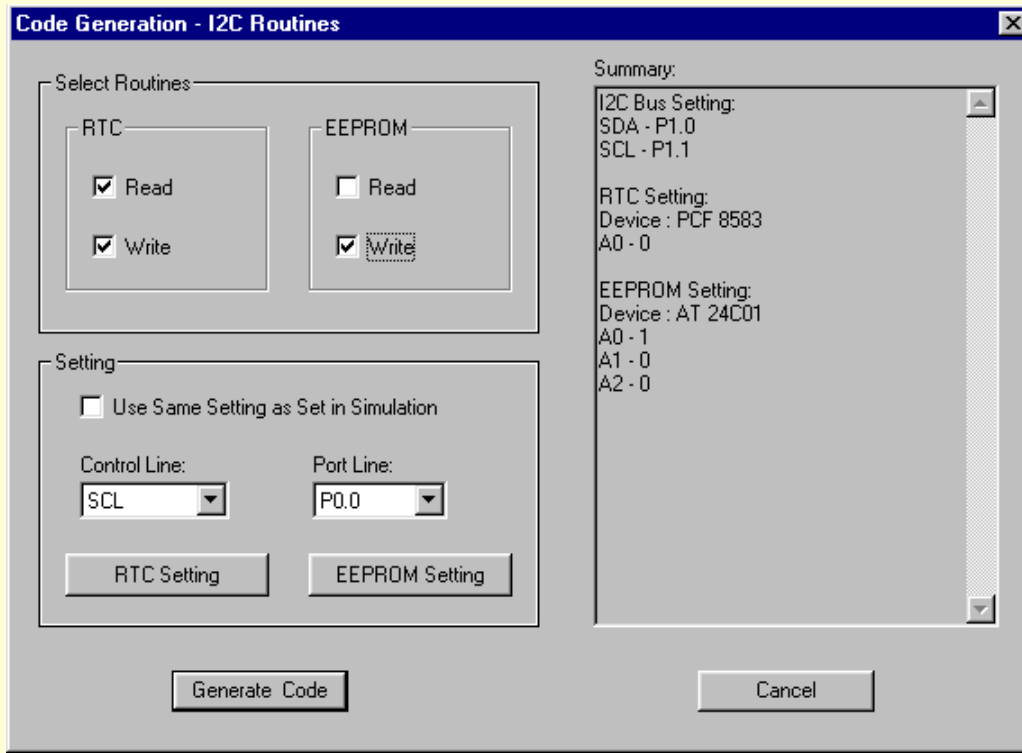
Use any port line to implement this key matrix.

Keyboard Interfacing



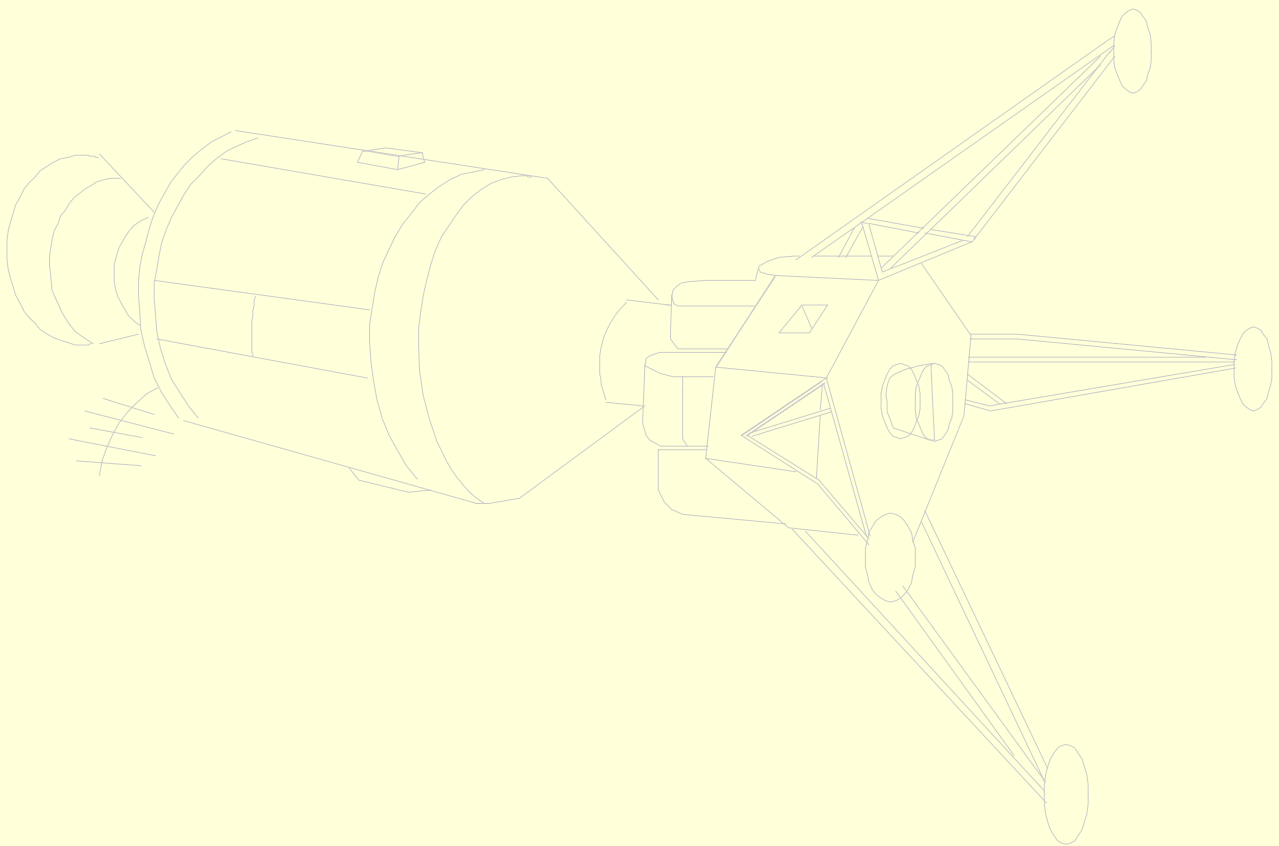
6.7.2.4 - I²C / SPI Buses

The Topview Simulator gives tight and right assembly code to implement all possible peripheral functions as shown in the following diagrams.



Chapter 7 - Topview Debugger

- Introduction



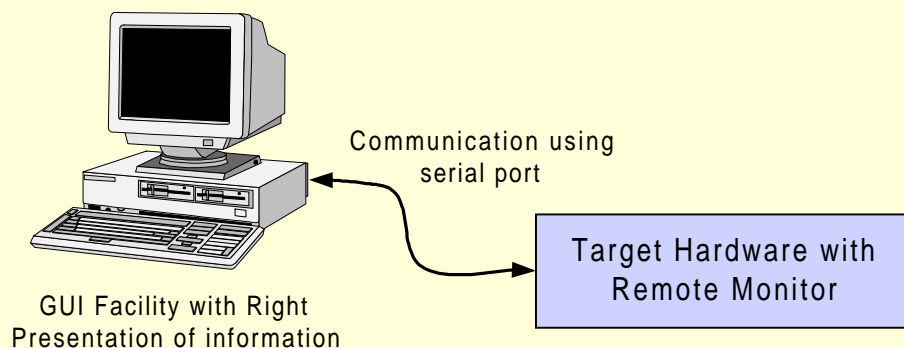
7.1 - Introduction

Topview Debugger is another important facility meant for developing 8031 Microcontroller based embedded solutions. Debugging is an inevitable part in any tool suite required to develop applications in real time. A right debugging tool may save a lot of development time in any product development process.

Topview debugger gives you the required development power to manage 8031 based projects. Topview supports generic 8031, 8032 devices and also Atmel's 89CXX family of microcontrollers.

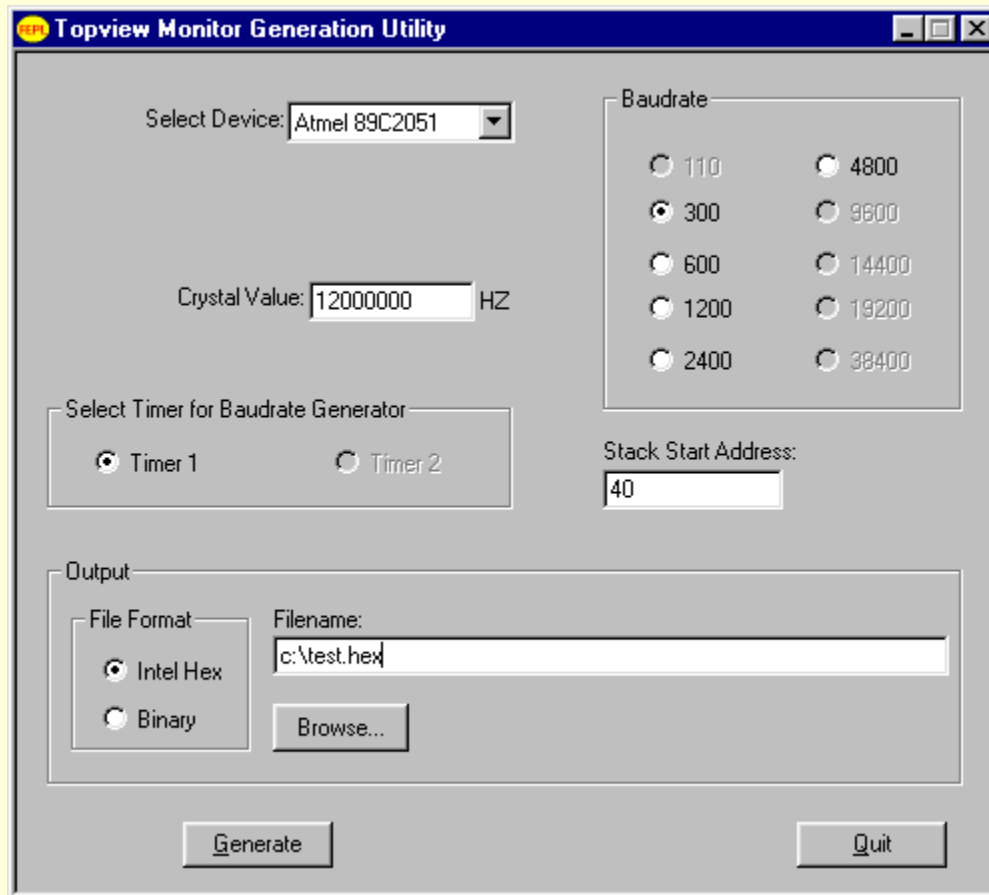
This debugger is a two part program in which major part stays inside of the target hardware and keeps track of internal operation of the 8031 microcontrollers. During program execution, it catches information on various register contents, internal/external memory areas and also various peripherals of the microcontroller. This information is later transferred to the host computer to which it is connected. Since it is residing in the target hardware, some times it is called ' Remote Monitor '.

Second part of the debugger operates in the host computer and is responsible for presenting the information received from the remote monitor in a most useful format using a GUI environment.



Basically you need to establish a reliable communication link between your target hardware and the host computer. You can make this happen in two simple steps.

First, you have to generate program code for the Remote Monitor that is going to sit in your target hardware. You activate the Topview Debugger in your host computer and select the monitor generating command and select the required options and then hit the code generate key. In a snap, you get the required hex code that is exclusively generated for your hardware.



Using a device programmer you program the target microcontroller in standalone design or an EPROM in the expanded version.

As the second step, you ensure in your hardware's PCB that all tracks between the connector and the serial port lines of the controller are staying good without any shorts or minute cuts along the way. Do anything to confirm yourself about this part of the deal.

Now connect both the target hardware and the serial port of the host computer using a suitable cable and activate the topview debugger and switch on the project hardware.

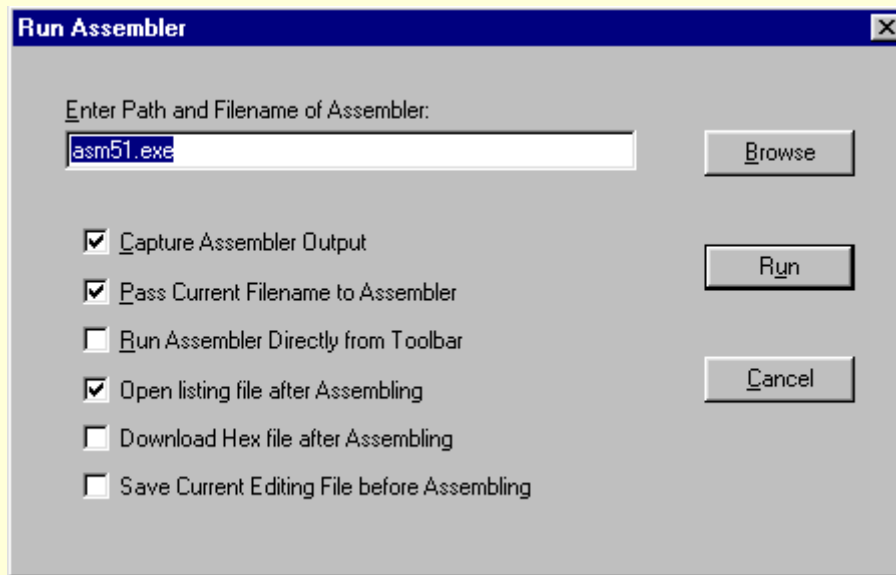
Now you are ready to debug your target embedded solution with Topview Debugger and you may not need to call your favorite gods often for the luck when using the debugger.

Here comes more information about the Topview Debugger:

- Topview Debugger can be used to debug the systems using generic 8031, 8051, 8032, 8052 and Atmel's AT89CXX family microcontrollers: 89C2051, 89C4051, 89C51, 89C52, 89C55, 89S53, 89S8252. Based on the selected controller, the GUI environment configures itself for exact amount of memory, peripheral functions.
- The target hardware may be either a single chip based design or an expanded one using external EPROM, SRAM and etc.
- Apart from making the target hardware compatible with the debugger, Topview generates a small size Remote Monitor that consumes about 1K byte in Program Memory area.
 - You can select either Timer1 or Timer 2 to generate baud rates for the serial port. Similarly there is a provision to select different baud rates based on the target's system clock.

Introduction

- The stack area can be predefined for the debugging purpose.
- The monitor can be generated in Intel hex or in Binary format.
- Then you should fuse the hardware's Microcontroller/EPROM with the generated monitor code starting from 0000H.
- Remaining area of the Program Memory from 2K onwards can be used to accommodate your application code. For the RAM based systems RAM area can also be used for program development, provided the Program and Data Memories overlap using external SRAM memory.
- Topview Debugger gives you all relevant information about all the microcontrollers to enable you to select a suitable device for the target application.
- The Debugger presents a GUI environment similar to Topview Simulator's windows arrangement. This clearview window structure gives facility for viewing/editing Program Memory, Internal Data Memory, External Data Memory, information on various internal registers, SFR bits and etc.
- A special feature of the debugger is the builtin text editing facility along with provision to run an external assembler for assembling the edited input program file.



- Any text file can be viewed/edited.
- The file size can go upto 640KB.
- Any third party assembler can be called in for assembling the edited input program.
- An option is there to download the hex file of the assembled program into the Program Memory of the target design at the predefined location.

Introduction

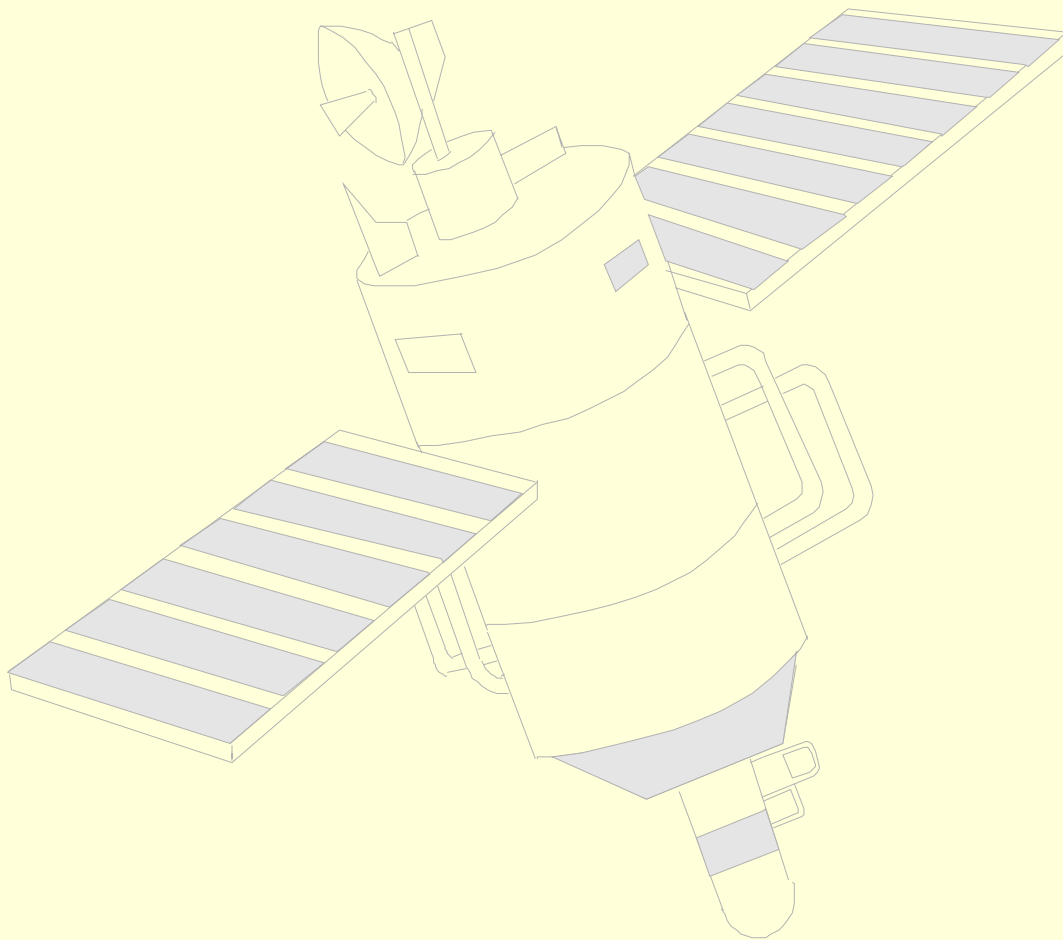
- In the Program Memory window, the address, opcodes and mnemonics with SFR names are displayed line by line with distinct colours.
- The Program Memory window sports a single line assembler. So you can simply enter your program line by line using mnemonics. The window also disassembles the hex input file and generates the actual assembly program.
- You can download a binary or hex file into RAM area of the target hardware reading from the disk.
- Similarly, you can store the contents of the Program Memory into the disk either in a hex or binary format.
- In the Internal Data Memory window, you can view/edit the memory contents. Facility is provided to FILL or Copy internal Data Memory contents.
- External Data Memory window also supports all the above mentioned and also sports the facility to test the external Data Memory area.
- In the Register and SFR windows, the names of all registers are clearly displayed to make your life easy during repeated debugging.
- You can execute your application program in many ways during trouble shooting operations. After every execution, the Debugger updates all the windows with the current information and the address at which the execution stops/breaks will be highlighted to grab your attention.
- You can execute the total program in a single shot and then check for desired results. You can also execute your program upto a Break point and then verify the contents of various registers and also the memory. The breakpoint can be defined using registers, internal Data Memory, SFR bits, external Data Memory or memory status.

There is another useful feature available to execute subroutines in a single step. You need not execute a subroutine block line by line.

Having described the debugger in detail, I am sure that you understand the clear advantage you get when implementing your next embedded solution using Topview Debugger. I once again remind you that a good debugging tool may save you from many hours of ' God-only-knows-what-happens-in-my-hardware '. You can also finish the project before the anticipated time.

Chapter 8 - **Topview Programmer**

- Introduction
- GUI Features

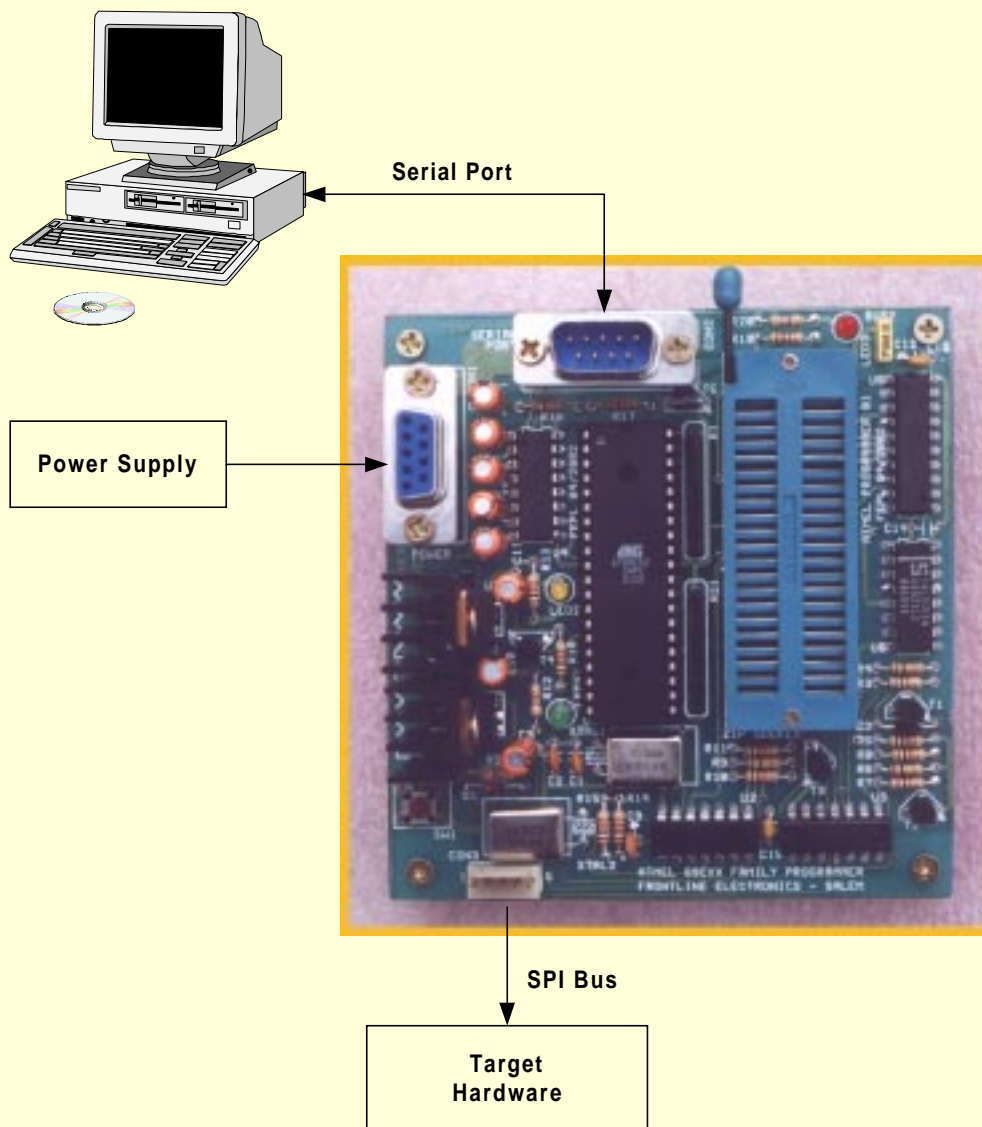


8.1 - Introduction

Now I have to introduce the most important part of your tool chest. Whether you use Simulator, Debugger or IDE or not, you definitely need a kind of programming facility to fuse your microcontroller with the target code. Only then you can watch how your design works. If you have an expanded design, then you may need to program the EPROM with the program code. Sometimes, you may keep a part of application code in the Flash/EPROM version of the 8031 and remaining part in an EPROM (in an expanded system).

Time has come to introduce another useful tool, Topview Programmer.

Topview Device Programmer supports the devices in the 8031 family of Atmel, AT89CXX. The programmer supports both parallel and SPI programming for your convenience. A powerful window's based GUI facility makes your programming task an easy one.



- **Supported Devices:**

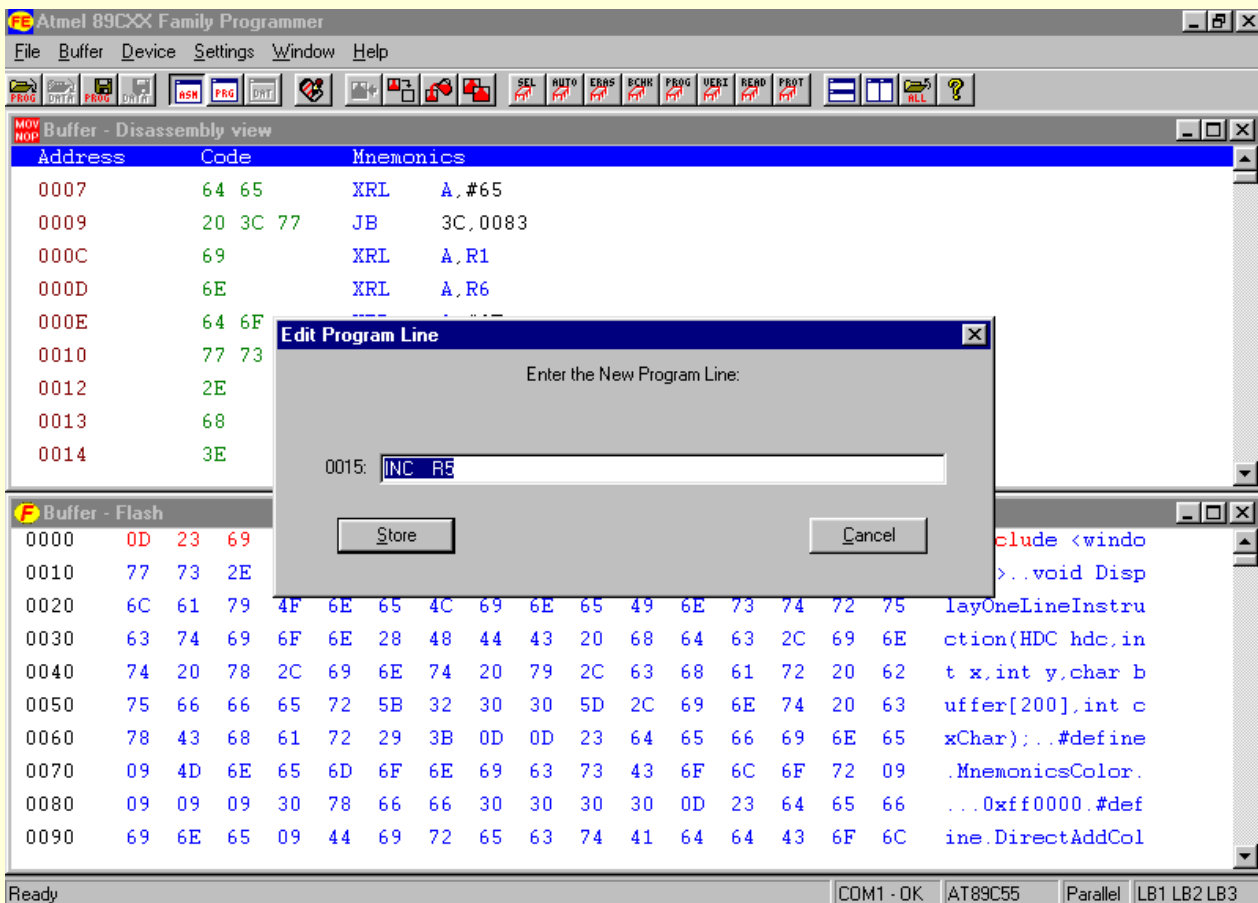
20 Pins: AT89C1051, AT89C1051U, AT89C2051, AT89C4051.

40 Pins: AT89C51, AT89S51, AT89LV51, AT89LS51, AT89C52, AT89S52, AT89LV52, AT89LS52, AT89C55, AT89LV55, AT89S53, AT89LS53, AT89S8252, AT89LS8252

- Programmer can be connected to the host PC through the serial port: COM1 to COM4.
- Supports both Parallel and SPI programming. Facility is available to program the device soldered in the target hardware. This facility is available in devices : AT89S8252, AT89LS8252, AT89S53 and AT89LS53.
- Separate connector and cables are provided for this purpose.
- A single 40 pin ZIF socket is there for programming both 20pin and 40pin devices.
- Programmer comes with suitable power supply and all relevant cables and is ready for usage immediately after opening the pack.

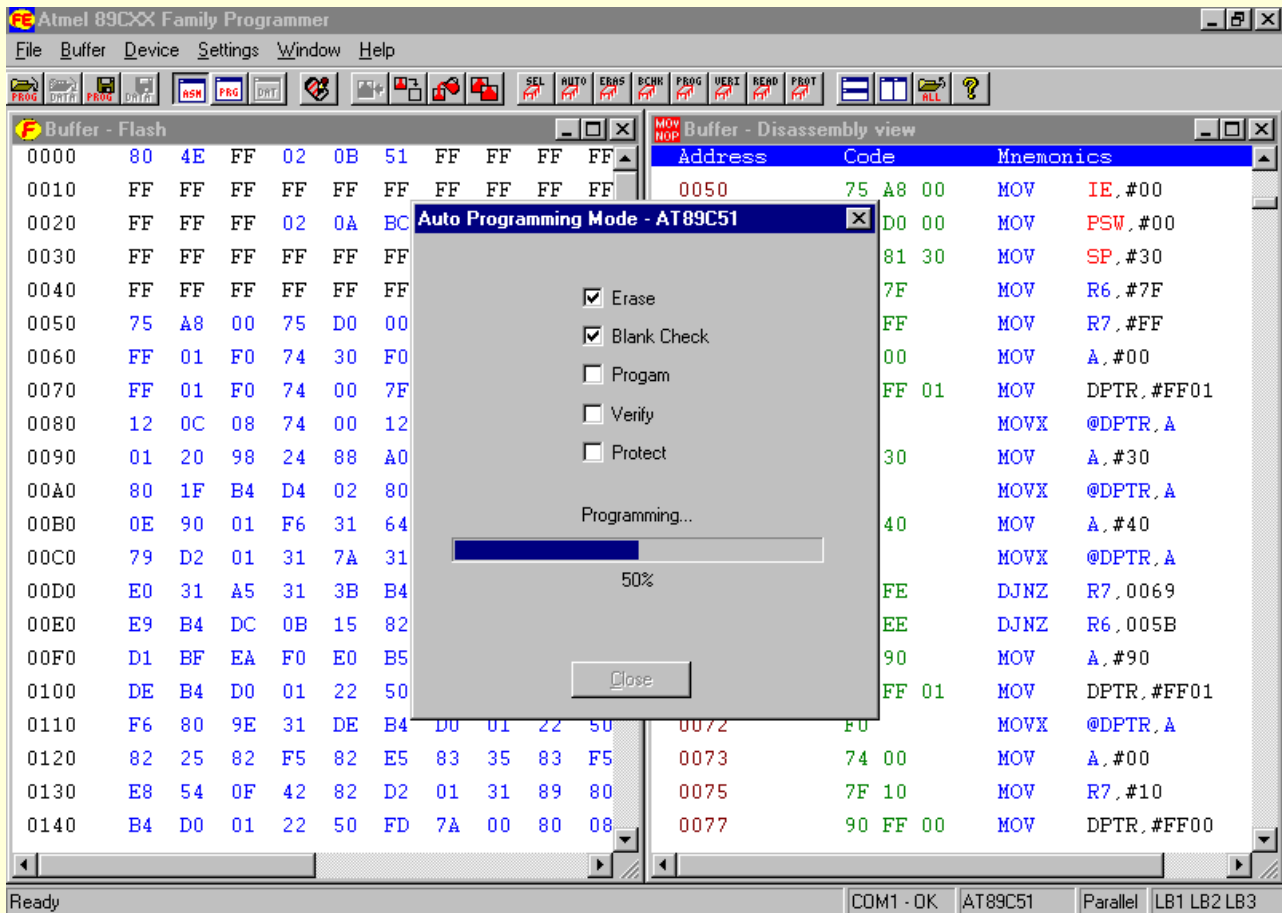
8.2 - GUI Features

- Software maintains a separate buffer for both Flash and EEPROM memory areas of the target Microcontroller.
- Hex or Binary file formats are supported in file access operations.
- You can also view the memory content as program lines.
- The software sports a built-in single line assembler to enable you to edit/modify program lines without returning to your development tools.



You may find it as an useful handy feature when debugging the code and also at field to modify the reference data or adjusting the critical part of the program.

All standard functions like Chip Erase, Blank Check, Program, Verify and Reading from the device and other file related operations: Fill, Copy, Checksum are available for all devices.



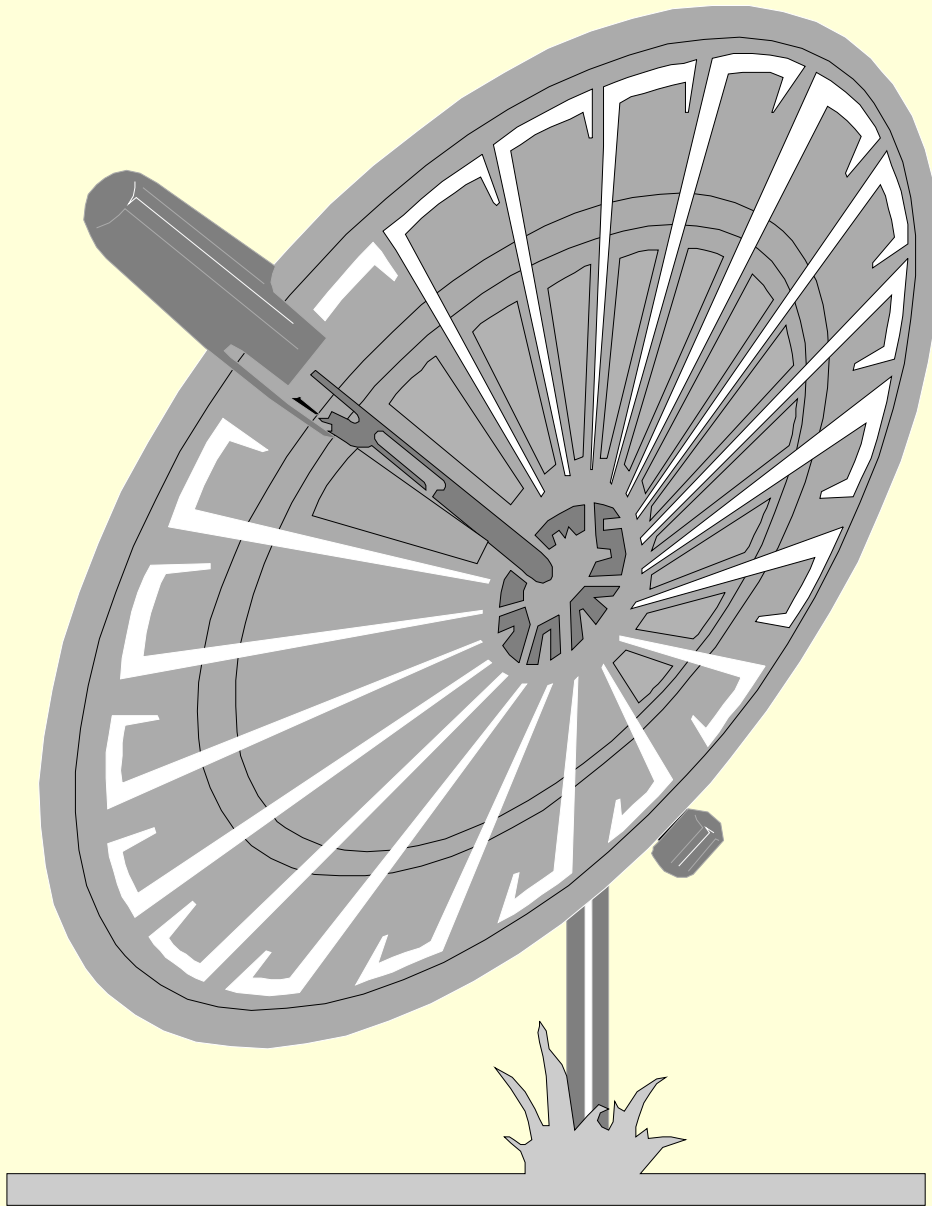
In the buffer operations, colouring facility is provided to indicate the source of the program/data.

Auto programming facility is provided to carryout all the programming tasks in a single shot to get fast programming for bulk production.

Software enables the users select the protection type taking into account of different protection options available for 20 pin and 40 pin device.

Chapter 9 - Contact Frontline Electronics

- Contact Frontline Electronics



9.1 - Contact Frontline Electronics

Frontline Electronics has few more development products like Topview Trainers, Proto boards meant for 8031 designers.

Topview Trainer is meant for learning Atmel's AT89CXX microcontrollers. Apart from the standard features, the trainer comes with an exclusive version of Topview Debugger. This facility should encourage the user to evaluate our Debugger and enable him/her to develop more complicated assembly language programs.

All the information is readily available in our website : www.Frontline-Electronics.com. You can also send email to feplslm@frontlinemail.com for any specific information/clarification.

Our address : [Frontline Electronics Pvt. Ltd.](#),
1/255C - Thatha Gounder St,
Kumaran Nagar
Alagapuram,
Salem - 636 016,
Tamilnadu,
India.
Phone : 0091 427 244 9238 / 243 1312.
Fax : 0091 427 244 9010.

Note : we are looking for Distributors / Country Representatives for our tools at every corner of the globe.