STANDALONE SCROLLING DISPLAY USING AT90S8515 AVR

■ **SHUBHIKA TANEJA, DEEPA CHAWLA**

Microcontrollers are being extensively used in many industrial and household applications. Here, we've used an AVR microcontroller (AT90S8515) from Atmel Corp. for controlling four 5x7 dot-matrix displays. The microcontroller is based on true reduced instruction set computer (RISC) architecture. Any message entered by the user through the keyboard of a PC scrolls elegantly through the displays even after disconnection of the circuit from the PC.

This display can be used in public places such as railway stations and restaurants to convey messages to the public. The microcontroller is interfaced to the PC keyboard through its serial port. The embedded system software is written in 'C.'

The circuit has the following features:

1. It accepts any message entered through the keyboard of the PC for display.

2. User interface is provided through the PC's RS-232 serial port (COM port).

3. The circuit derives power from 230V AC mains, which is converted into regulated 5V DC.

4. The string of characters entered through the keyboard is stored in the EEPROM. The stored message can be displayed on the dot-matrix display just by clicking the scud button on the terminal program while it is connected to the PC.

5. Any message entered from the PC's keyboard gets stored in the EEPROM of the AVR and can be scrolled at any time without the use of a PC, i.e. you just need to switch on the embedded system.

6. RXD and TXD pins of the microcontroller are used to communicate with the PC through MAX-232 IC and TX and RX pins of COM port. All the four ports (ports A, B, C and D) of the AVR are programmed as output ports.

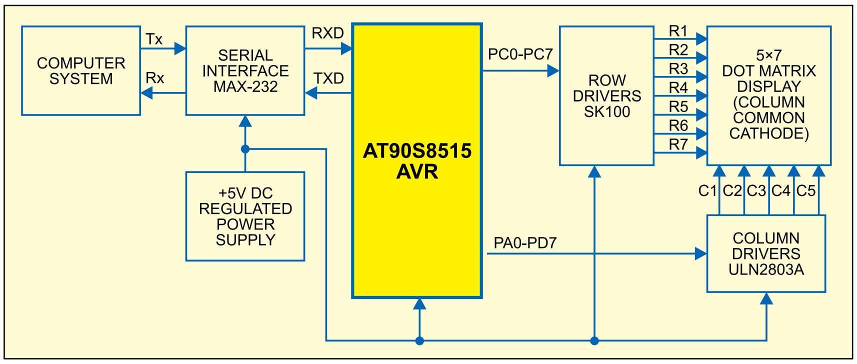


Fig. 1: Block diagram of standalone scrolling display using AT90S8515 AVR

Fig. 1 shows the block diagram of the AT90S8515-based standalone scrolling display system. It consists of an AVR microcontroller, row display drivers, column display drivers, four 5x7 dot-matrix displays and power supply section. The AVR compiler, in-system programmer (ISP) and terminal program are installed in the computer. The display control program, written in 'C' using AVR C compiler, is loaded into the microcontroller by using parallelport pins of the PC.

PARTS LIST

Semiconductors:		
IC1	-	AT90S8515 AVR micro-controller
IC2-IC6	-	ULN2803A Darlington array LED driver
IC7	-	MAX232 RS-232 serial interface
T1-T7	-	SK100B pnp transistor
Resistors (all ¼-watt, ±5% carbon):		
R1	-	220-ohm
R2-R8	-	1-kilo-ohm
R9-R15	-	220-ohm
R16	-	620-ohm
Capacitors:		
C1	-	100µF, 16V electrolytic capacitor
C2, C3	-	22pF ceramic capacitor
C4	-	0.1µF ceramic capacitor
C5-C9	-	1µF, 16V electrolytic capacitor
Miscellaneous:		
XTAL	-	8MHz crystal
DIS1-DIS4	-	5×7 dot-matrix (column common cathode) display
LED1	-	Red power indicator
S1	-	SPST on/off switch
S2, S3	-	Tactile switch

Circuit description

Fig. 2 shows the circuit of AVR AT90S8515-based scrolling display system.

AT90S8515 AVR microcontroller. AT90S8515 is a 40-pin, 8-bit microcontroller from Atmel. It has 512 bytes of SRAM, 512 bytes of EEPROM and 8kB Flash with 32 programmable input/output (I/O) lines. AVR microcontrollers are in-system programmable through RS-232C serial port (COM port) of the PC. The programmable Flash memory and EEPROM of the AVR can be programmed using a simple software and just four wires from parallel port of the PC to your target board containing AVR. Easy in-circuit programmability combined with Flash memory makes it easy to update the code during development. Since we require a minimum of 27 output pins (20 columns and 7 rows), AT90S8515 suits this application as it has 32 programmable I/O lines. Pin details of this AVR are shown in Fig. 4. The AVR marked on the IC with 8PI or 8PC indicates the value of the crystal to be used, which in this case is 8 MHz. The baud rate in the communication software should be selected as per the following relationship:

$$\text{Baud rate} = \frac{f_{CLK}}{16(V_{BRR}+1)}$$

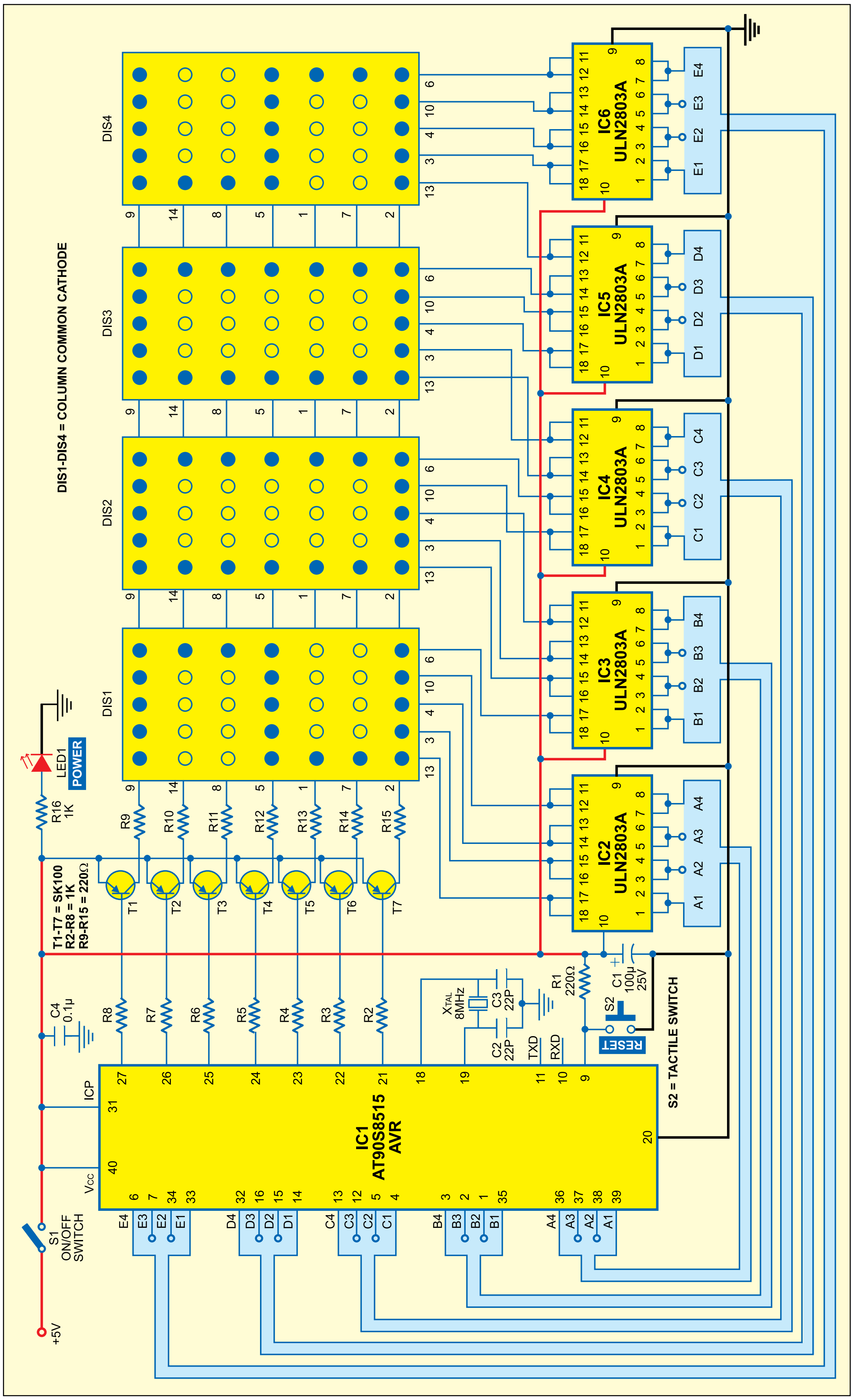


Fig. 2: Circuit for standalone scrolling display using AT90S8515

where f_{CLK} is crystal frequency and V_{BRR} is the value of contents of the UART baud rate register.

Serial interface. The serial interface comprises 9-pin D-type female connector, IC MAX-232, five 1μF electrolytic capacitors and 3-core cable as shown in Fig. 3.

Display drivers. Seven SK100B transistors along with 220-ohm (output current limitor) and 1k-ohm resistors (base current limiter) are used for controlling the rows of LED array, and five ULN2803 ICs (IC2 through IC6) are used for controlling the columns of dot-matrix displays.

Dot-matrix displays. Four 5x7 dot-matrix LEDs (with common cathodes as the columns) such as KLP2057 from Kwality Electronics (India) are used for the display. The displays need seven row drivers and 20 column drivers. These displays are identical, with cathodes shorted along the column and anodes shorted along the row (refer Fig. 5).

Since the human eye cannot perceive changes carried out at frequencies greater than 20 Hz, each column must be refreshed at a minimum rate of 20 Hz. Here, we have set the refresh rate (the rate at which the display from one column to the next) at about 400 Hz. In case only one LED glows in a particular col-

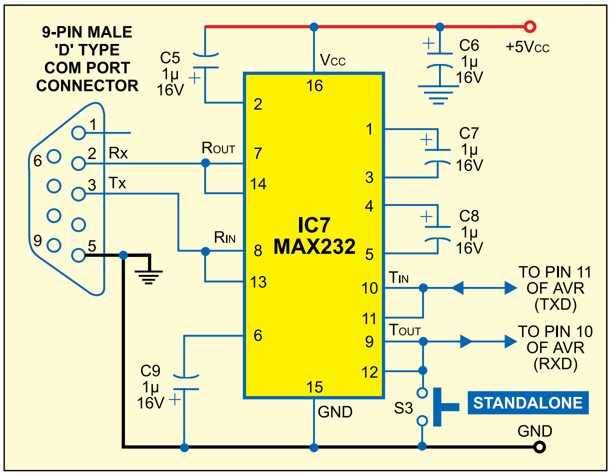


Fig. 3: RS-232 interface circuit

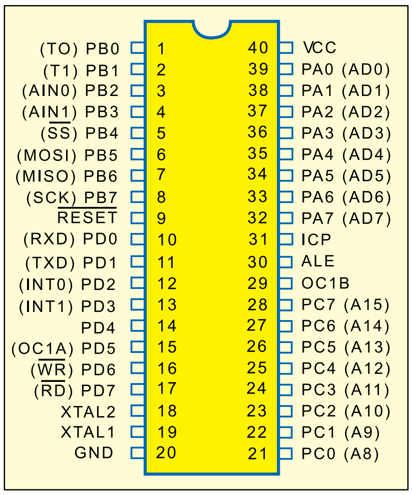


Fig. 4: Pin details of AT90

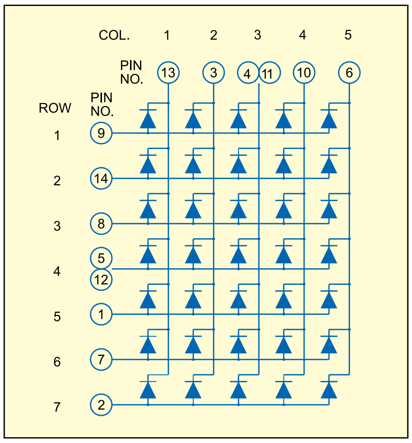


Fig. 5: Column common cathode

umn, that particular data line will have to handle 20mA current.

Since there are 20 LEDs in a row, 400mA current could flow through a particular column at a particular instant. The circuit has to be designed keeping the value of this peak current in mind. Since 400mA current cannot be sourced by the port pin of AVR (maximum current sourced or sinked by the AVR's I/O ports is 20 mA), the display cannot be directly connected to the AVR port. We thus use SK100B pnp transistors along with 220-ohm current-limiting resistors.

For obvious reason, we've used five ULN2803 ICs to increase the current sinking capacity. These ICs are connected to the columns of the displays. Each IC has eight Darlington pairs. Pairs of input and output pins of ULN 2803 are connected in parallel to increase the current sinking capability. The transistors are turned on by the TTL voltages applied by the input/output ports of the AVR to their bases through 1-kilo-ohm resistors.

Power source. A 5V DC regulated power supply is used in this circuit, which has to be supplied externally.

Connecting the AVR to the PC's serial port

The microcontroller needs to communicate with the PC's RS-232 port to scroll the string entered through the keyboard of the PC. AT90S8515 has a built-in serial port. The processor takes care of serialising and shifting out of the data on the output pin and assembling of the incoming data into a byte. Since the RS-232 signals are bipolar in nature, they cannot be fed directly to the controller. We have used a very popular RS-232 line driver and receiver MAX232 (IC7) for converting the PC's RS-232 compatible signals into TTL levels for AVR and vice versa. T_{IN} (TTLinput) and TOUT (TTL output) pins of MAX232 are connected to the transmitter (TXD) and receiver (RXD) pins of the AVR, respectively.

The transmitter (T_X) and receiver (R_X) pins of the PC's Com port are connected to the R_{IN} (RS-232 input) and R_{OUT} (RS-232 output) pins of MAX232, respectively. A 9-pin D-type male connector is attached to the PCB board, whose pins 2, 3 and 5 are soldered to R_{OUT}, R_{IN} and ground of IC7, respectively.

Two 9-pin D-type female connectors are required for connection between the PCB board and the PC's serial port. The communication between the PC and the circuit board for display is done through a terminal program software such as 'Terminal v1.9b,' which can be downloaded for free from the Website 'bray.velenje.cx/avr/terminal.' Using this software, up to 130 characters can be typed in at a time for transmission to the display circuit for the scrolling display.

Programming the AVR

Getting started with the AVR requires nothing more than the free assembler/compiler, a simple programmer such as the one by Jerry Meng (available on 'www.qsl.net/ba1fb/') and a target board. The target board can be as simple as a few parts since the AVR is highly integrated. Since it is easy to reprogram the flash memory, you can develop code and test without the need for an expensive in-circuit emulator. This is done by a built-in interface in the AVR chip, which enables you to write and read the contents of the programmed Flash and the built-in-EEPROM. This interface works serially and needs mainly three signal lines from the AVR to PC's printer port for programming:

1. SCK: A clock signal that shifts the bits to be written to the memory into an internal shift register, and that shifts out the bits to be read from another internal shift register.

2. MOSI: The data signal that sends the bits to be written to the AVR.

3. MISO: The data signal that receives the bits read from the AVR.

The connections for program-

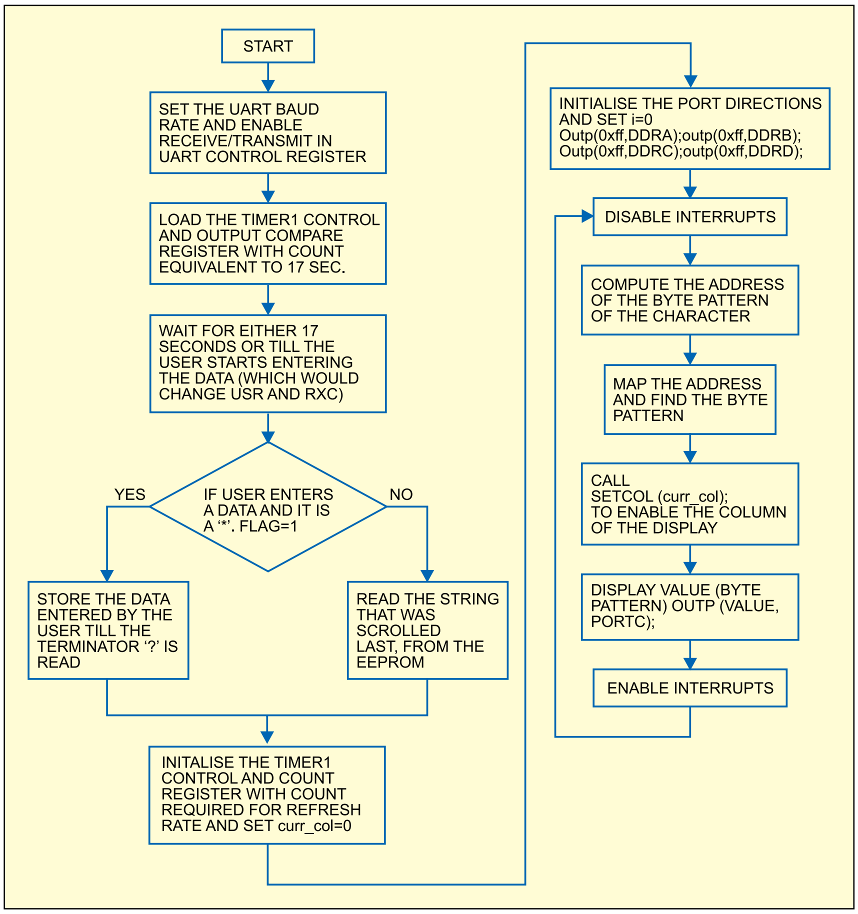

START
SET THE UART BAUD RATE AND ENABLE RECEIVE/TRANSMIT IN UART CONTROL REGISTER
LOAD THE TIMER1 CONTROL AND OUTPUT COMPARE REGISTER WITH COUNT EQUIVALENT TO 17 SEC.
WAIT FOR EITHER 17 SECONDS OR TILL THE USER STARTS ENTERING THE DATA (WHICH WOULD CHANGE USR AND RXC)
IF USER ENTERS A DATA AND IT IS A '*'. FLAG=1
YES
NO
STORE THE DATA ENTERED BY THE USER TILL THE TERMINATOR '?' IS READ
READ THE STRING THAT WAS SCROLLED LAST, FROM THE EEPROM
INITALISE THE TIMER1 CONTROL AND COUNT REGISTER WITH COUNT REQUIRED FOR REFRESH RATE AND SET curr_col=0
INITIALISE THE PORT DIRECTIONS AND SET i=0 Outp(0xff,DDRA);outp(0xff,DDRB); Outp(0xff,DDRC);outp(0xff,DDRD);
DISABLE INTERRUPTS
COMPUTE THE ADDRESS OF THE BYTE PATTERN OF THE CHARACTER
MAP THE ADDRESS AND FIND THE BYTE PATTERN
CALL SETCOL (curr_col); TO ENABLE THE COLUMN OF THE DISPLAY
DISPLAY VALUE (BYTE PATTERN) OUTP (VALUE, PORTC);
ENABLE INTERRUPTS

Fig. 6: Flow-chart of the program

ming are simple but there are various standards adopted by the industry. In this project, the ISP10 standard is used on the STK200 programmer board (from KANDA Systems) for programming. The STK200 board consists of the zif socket for the AVR and a 10-pin header box. The dongle is used to connect the port of the PC to the 10- in header connector on the STK200 board. Along with this STK200 board, you need a compiler/assembler such as AVREdit 3.5 and Atmel AVR ISP 2.65 software to be installed into your system for programming the AVR chip. The required software tools can be downloaded from the Website 'www.avrfreaks.net.' The STK200 dongle is available on the Website 'elm-chan.org/works/avrx/report_e.html.'

EFY note. A simple dongle circuit used in EFY Lab for programming the AVR will be published in the next issue.

Software Program

The software has the following features:

1. Initially waits for 17 seconds for the user to enter the string.

2. Receives data from UART sent through the serial port of the PC connected to MAX232 by a 9-pin connector.

3. Stores the string entered by the user. Else, retrieves the previously stored string from the EEPROM.

4. Stores the byte-patterns of characters 'A' through 'Z,' 'a' through 'z' and '0' through '9' in the 16-bit programmable flash memory.

5. Initialises the interrupts for refresh rate and scroll rate.

6. Maps the byte pattern of each character from the program memory as a function of the scroll parameter and then sends the values to the ports.

The flow-chart of the program is shown in Fig. 6.

The 8-bit timer/counter of the AVR is used to implement refreshing of the display. As the minimum refresh rate for flicker-free view is 20 Hz, we have chosen prescale as Clk/64, thus giving us the refresh rate in kilohertz, where 'Clk' is the oscillator clock frequency of the crystal used.

Wait interrupt has been implemented by the 16-bit timer/counter with clk/1024 as the pre-scaler and output-compare register (OCR). This gives us an initial wait period of 17 seconds.

Sub-modules of the code. During the 17-second waiting period, the program waits for the user to send data through the UART. Hence, the program waits in while loop 'While (! (USR&(1<<RXC))&& (q! =0));' and keeps checking the RXC bit (UART Receiver Complete) of the UART status register (USR) until either the user enters a data byte (RXC bit will be set) or the 16-bit timer/counter output compare interrupt is generated and the while loop terminates. The 16-bit timer/ counter is initialised as 'TCCR1B=5; OCR1AH=10;' which defines the prescaler of 'clk/64.'

To receive data from UART sent from the serial port of the PC, first the UART baud rate and UART control register (UCR) are set to enable the receiver and the transmitter as 'UBRR=25; UCR=(1<<RXEN)|(1<<TXEN);' where UBRR is the UART baud rate register.

If the user sends a new string, it will first be received from the UART data register (UDR) and stored in SRAM, then it will be written into the EEPROM, which, in turn, overwrites the previously stored string. The following lines enable storing of the string in SRAM:

```
While ((count1<100) && (str1 [k]! = 63))
{
if(USR & (1<<RXC))
flag=1;
```

If the string entered is in the correct format, the flag is set to '1.' Else, the flag remains '0' and the previously

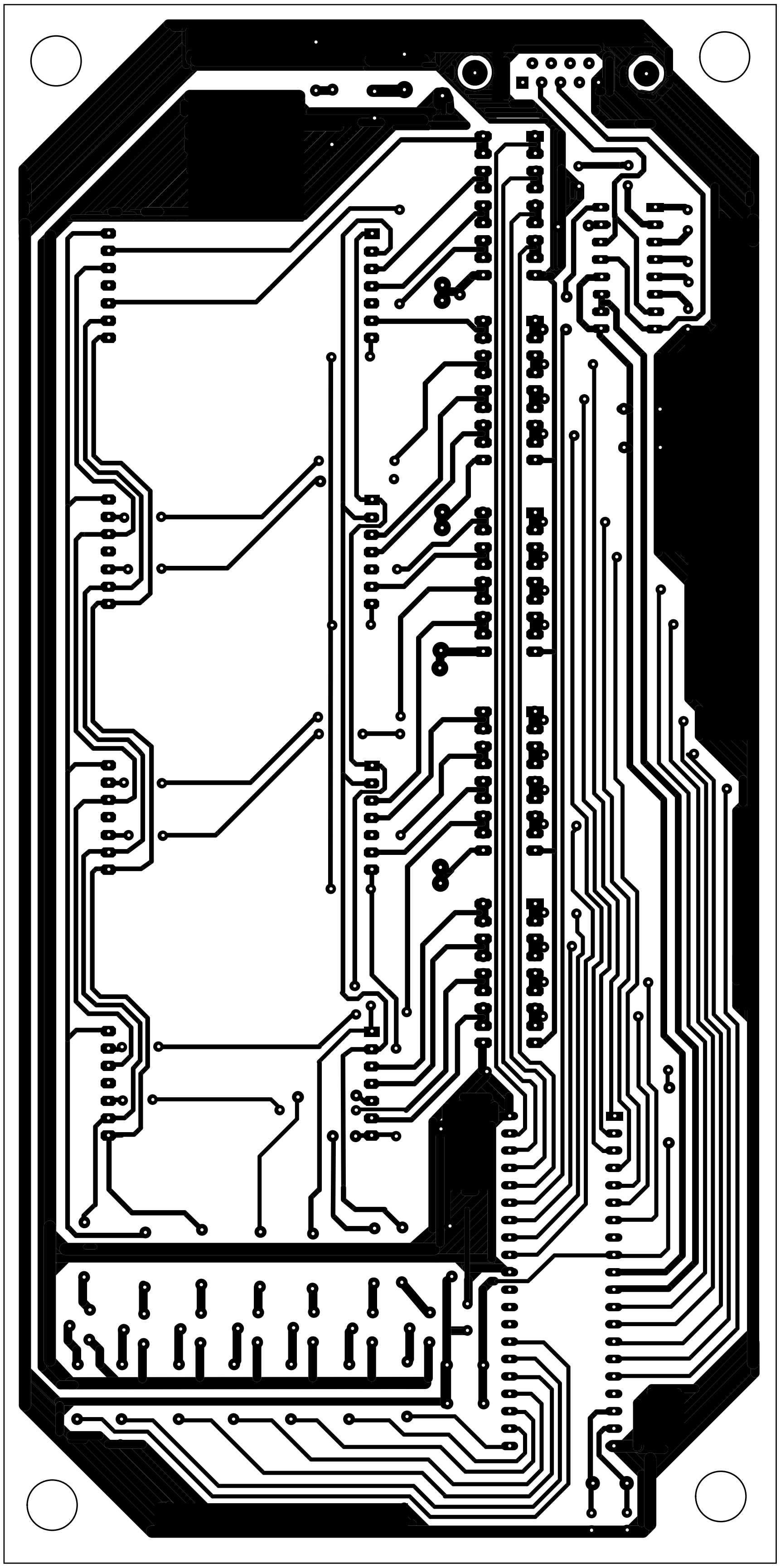


Fig. 7: Combined actual-size, single-side PCB layout for Figs 2 and 3

stored string will be displayed. To store the string in EEPROM, the string is written character-by-character in the EEPROM starting from location '0x0001.'

If the previously stored string is to be scrolled, the same routine is executed, except that data is only 'read from' instead of 'written to' the EEPROM. The following program lines perform these actions:

```
address = 0x0001;
EEREAD( address, str+x);
EEWRITE(address,str1[x]);
//Store the string
in EEPROM
```

To store the byte patterns of characters 'a' and 'b' in the 16-bit programmable flash memory, an extract from the program is reproduced below:

```
typedef unsigned char u08;
u08 __attribute__ ((progmem)) leds[]={
0xe0, 0xd7, 0xb7, 0xd7, 0xe0, //a
0x80, 0xb6, 0xb6, 0xb6, 0xc9, //b
```

The program lines "t = str[i]; addr = (t-'A')*5;" are used to retrieve the starting address of the byte-pattern of any character, where 'A' is the base address.

Initialisation of interrupts for refresh rate and scroll rate is as follows:

```
TCNT0 = 200;
TIMSK |= 1<<TOIE0 ;
TCCR0=3;//Timer/Counter Control Register
```

An 8-bit timer/counter (TCNT0) is used in the program, whose value can be changed to increase the intensity of the display. The scroll rate has been taken as a multiple of refresh rate. This multiple is taken as '2000.' When the string to be scrolled is known, first the input/output ports are set by the following instructions:

```
outp(0xff,DDRA);
outp(0xff,DDRB);
outp(0xff,DDRC);
outp(0xff,DDRD);
```

To map the byte pattern of each character of the string from the program memory as a function of the scroll parameter (named as offset here) and then send the values to the ports, the following section of the program is a critical section. As we don't want the interrupts to occur

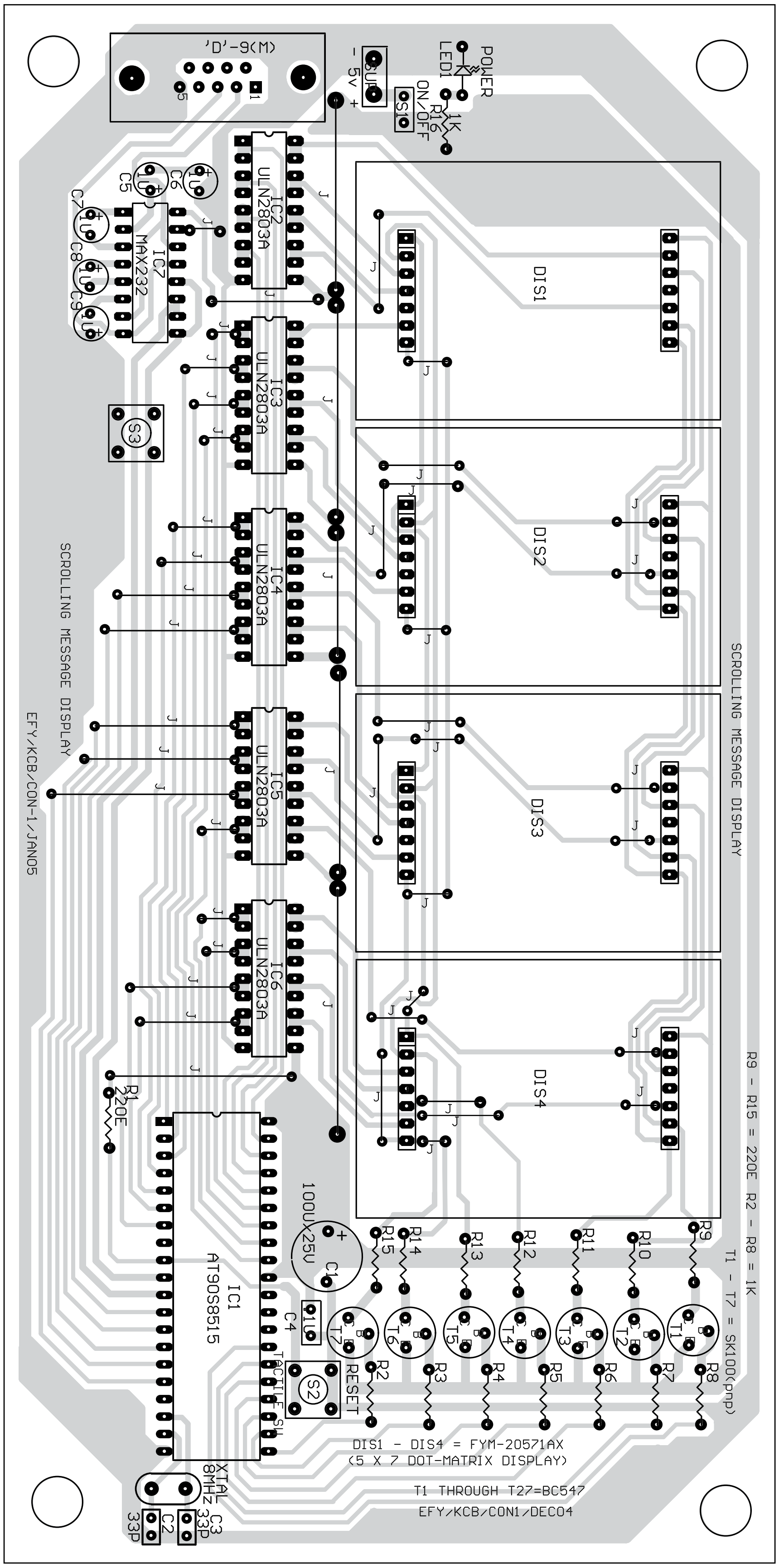


Fig. 8: Component layout for the PCB

during their execution, we use cli () and sei ():

```
"cli();//disable interrupt in
critical section
if( j == 2000
t = str[i];
if(t>=65&& t<=91);// Characters
between A
and Z
addr=(t-'A')*5;//i is being
incremented in interrupt
else if(t>=97&& t<='122);
// Characters between a and z
else if(t>=48&& t<=57);
// Characters between 0 and 9
curr_col_temp=(curr_col<5)?
curr_col:curr_col%5;
m = offset + curr_col_temp;
if(m>=5) m=m-5;
addr = addr + m;
value = PRG_RDB(&leds[addr]);
outp( value, PORTC);
setcol(curr_col);
sei();//enable interrupt"
```

The function 'setcol(int col)' is called to send appropriate values to the ports to drive the column LEDs.

Construction

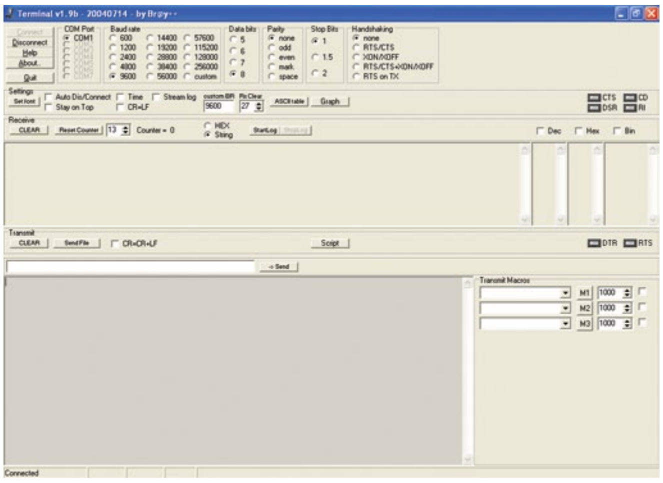
The circuit can be constructed on any general-purpose PCB. A 3-core serial cable is used for communication with the PC's keyboard. The 9-pin male connector is soldered on the PCB to interface with the cable. 5V DC regulated power supply is required for the circuit as well as programming the circuit, which can be constructed on a separate PCB.

An actual-size, solder-side combined PCB layout for the display and interface circuits (Figs 2 and 3) is shown in Fig. 7 and its component layout in Fig. 8.

Testing procedure

After having mounted all the components, except AVR on the PCB, you have to perform the initial test (optional) to check the connections of the 5x7 dot-matrix displays. The 'check.c' program given below can be programmed into the AVR for this checking. The various steps involved are:

1. Download the 'AvrEdit3.5' software and Atmel AVR ISP and


Terminal v1.9b - 20040714 - by Br@y++

COM Port: COM1 (selected) · Baud rate: 9600 (selected) · Data bits: 8 (selected) · Parity: none (selected) · Stop Bits: 1 (selected) · Handshaking: none (selected)

custom BR: 9600 · Rx Clear: 27 · Reset Counter: 13 · Counter = 0 · String (selected)

Connected

Screenshot of terminal program

install in your system. The 'AvrEdit' and 'Avrtools' folders automatically get created in the respective software.

2. Create another folder, say, 'Discheck,' under the 'AvrEdit' folder and copy the 'check.c' file into the 'Discheck' folder.

3. Run 'AvrEdit' from the desktop, open the 'check.c' program and click 'Run' in the menu bar for compilation. After compilation, the 'Check.Rom' file is automatically generated under the 'Discheck' folder.

4. Now, connect the STK200 (dongle) to the parallel port of the PC and insert the AVR into the zip socket of the STK200 board.

5. Run the Atmel AVR ISP from the desktop, select 'New Project' to load the 'Check. Rom' file from the 'AvrEdit' folder.

6. From 'Program' menu bar of the ISP, select 'Program Device' to program the AVR.

Remove the programmed AVR from the STK200 board. The AVR, when inserted into the populated PCB, will light up all the LEDs in the display devices if the circuit connections are correct.

Now, to program the main program 'ScrollD.c' into the AVR chip, create a folder, say, 'Scroll' under the 'AvrEdit' folder. Copy 'ScrollD.c' into the 'Scroll' folder, run 'AvrEdit' and follow steps 2 through 6 as mentioned above. After programming the AVR, remove it from the STK200 board and insert into the main circuit.

7. Connect the 9-pin D-type female connector from the main circuit to the COM port of your PC.

8. Download the 'Terminalv1.9b' communication software and install it in your PC. An application file icon named 'Terminal' will be created on the desktop.

9. Switch on the power to the circuit and run 'Terminal' from the desktop. Choose the baud rate of this application as 9600 and parity bit as none (refer to the screenshot).

10. Click 'Connect' button and type '*New Year 2005?' in the transmit box. Note that the message should always be enclosed between '*' and '?' before transmission.

11. Click 'Send' button to transmit the characters for display on the dot-matrix displays.

12. To enter new characters for display, click 'Disconnect' button, press reset switch S2 and type new message in the transmit/edit box. Click 'Connect' button followed by 'Send' button.

13. If a particular string is to be scrolled again and again, disconnect the circuit from the PC. Whenever the circuit is switched on, the display system will wait for 17 seconds and the previous string stored in the EEPROM will scroll on the displays without the need of serial cable, Terminal program and PC. This feature makes this embedded system a standalone system.

EFY note. 1. It was observed that a momentary low pulse is required to be provided at pin 10 (RXD) of the AVR through switch S3 to initiate the display without PC.

Download source code: http://www.efymag.com/admin/issuepdf/SCROLL%20DISPLAY.zip

SCROLLD.C

```
 // Code for AVR PROJECT of  Scrolling Display
 #include <eeprom.h>
 // Offset b/w 0 and 4
 #include <io.h>
 #include <progmem.h>
 #include<interrupt.h>
 #include<sig-avr.h>
 #include<ina90.h>
 //offset is the beginnig pointer
 // global varables
 int curr_col,i=0,j=0,offset=0,temp=0,q=1;
 unsigned char str[100], str1[100];
 int count=0, address,x,x1 ;
 void EEWRITE( int address,char value);
 void EEREAD( int address,char *val);
 void setcol(int col);
 SIGNAL(SIG_OUTPUT_COMPARE1A)
 {q=0;}
 SIGNAL(SIG_OVERFLOW0)
 {
 int k;
  setcol(-1);
 curr_col++;
j++;
if ( curr_col==20)
{
curr_col=0;
if( offset ==0)
 {
  if( i>=3) i=i-3;
 else i=i+count-3;
//offset++;
 }
else
if(offset==4 && j== 2000)
{i=temp+1;
temp=i;
}
 else
 { i--;
 k = 20 - offset;
 while( k>=5){ k=k-5; i--; if(i<0) i=i+count; }
 }
}

else
{
 int x = (curr_col<5)? curr_col: curr_col%5 ;
 if( (x!=0&&(x+offset)%5==0) ||(offset==0 && (
curr_col==5 || curr_col==10 ||curr_col==15 ||
curr_col==20)))
 i++;//char shift
 if(i==count ) i=0;
 }
 if(i==count)//added now
 i=0;
 TCNT0 = 230;
 }

 typedef unsigned char  u08;
 u08 __attribute__ ((progmem)) leds[]={
 0xe0, 0xd7, 0xb7, 0xd7, 0xe0,
 0x80, 0xb6, 0xb6, 0xb6, 0xc9,  //b
 0xc1, 0xbe ,0xbe, 0xbe, 0xdd,  //c
 0x80, 0xbe ,0xbe, 0xbe, 0xc1,  //d
 0x80, 0xb6, 0xb6, 0xb6, 0xbe, //e
 0x80, 0xb7, 0xb7, 0xb7, 0xbf,  //f
 0xc1, 0xbe, 0xba, 0xba, 0xd9,  //g
 0x80, 0xf7, 0xf7, 0xf7, 0x80,  //h
 0xbe, 0xbe, 0x80, 0xbe, 0xbe,  //i
0xb9, 0xbe, 0xbf, 0x81, 0xbf,  //j
0x80, 0xf7, 0xeb, 0xdd, 0xbe,  //k
0x00, 0xfe, 0xfe, 0xfe, 0xfe,  //l
0x80, 0xdf, 0xe7, 0xdf, 0x80,  //m
0x80, 0xef, 0xf7, 0xfb, 0x80,  //n
0xc1, 0xbe, 0xbe, 0xbe, 0xc1,  //o
0x80, 0xb7, 0xb7, 0xb7, 0xcf,  //p
0xc1, 0xbe, 0xba, 0xbc, 0xc0,  //q
0x80, 0xb7, 0xb3, 0xb5, 0xce,  //r
0xce, 0xb6, 0xb6, 0xb6, 0xd9,  //s
0xbf, 0xbf, 0x80, 0xbf, 0xbf,  //t
0x81, 0xfe, 0xfe, 0xfe, 0x81,  //u
0x83, 0xfd, 0xfe, 0xfd, 0x83,  //v
0x00, 0xfd, 0xfb, 0xfd, 0x00,  //w
0x1c, 0x6b, 0x77, 0x6b, 0x1c,  //x
0xbf, 0xdf, 0xe0, 0xdf, 0xbf,  //y
0xbc, 0xba, 0xb6, 0xae, 0x9e,
0xf0,0xee,0xee,0xf1,0xfe,//a
0x00,0xf6,0xf6,0xf6,0xf6,//b
0xf1,0xee,0xee,0xee,0xff,//c
0xf0,0xf6,0xf6,0xf6,0x00,//d
0xe1,0xd6,0xd6,0xd6,0xe6,//e
0xf7,0x00,0x37,0x37,0xdf,//
0xcf,0x37,0x31,0x36,0xc0,//g
0xff,0x00,0xf7,0xf7,0xf8,//h
0xff,0xff,0xd0,0xff,0xff,//i
0xfd,0xfa,0x20,0xff,0xff,//j
0xff,0x00,0xfb,0xf5,0xee,//k
0xfb,0x00,0x2e,0xdd,0xff,//l
0xf0,0xef,0xf0,0xef,0xf0,//m
0x6f ,0x70 ,0x6f ,0x6f ,0x70,//n
0xf9,0xf6,0xf6,0xf6,0xf9,//o
0x00, 0x6d, 0x6b, 0x77, 0x7f,//p
0x4f, 0x37, 0x37 ,0x00 ,0x7b,//q
0xf7 ,0xf4 ,0xfa ,0xf4 ,0xf7,//r
0xf5 ,0xea ,0xea ,0xf4 ,0xff,//s
0xf7 ,0xf7, 0x00, 0xf6, 0xf5,//t
0xf1, 0xfe ,0xfe, 0xfe, 0xf1,//u
0xef ,0xf1 ,0xfe ,0xf1 ,0xef,//v
0xe1, 0xfe, 0xf9 ,0xfe, 0xe1,//w
0xee, 0xed ,0xf3 ,0xed, 0xee,//x
0xcf ,0xf7, 0xf5, 0xf2, 0xc0,//y
0xee ,0xec ,0xea ,0xe6 ,0xee,//z
0x00,0x3e,0x3e,0x3e,0x00,//0
0xff,0xff,0x00,0xff,0xff,//1
0xb0,0xb6,0xb6,0xb6,0x86,//2
0xb6,0xb6,0xb6,0xb6,0x00,//3
0x87,0xf7,0xf7,0xf7,0x80,//4
0x06, 0x36 ,0x36 ,0x36 ,0x30,//5
0xf1, 0xee, 0xd6 ,0xb8, 0x7f,//6
 0xbd, 0xbb ,0xb7 ,0xaf ,0x9f,//7
0xc9 ,0xb6 ,0xb6 ,0xb6 ,0xc9,//8
 0xcd ,0xb6, 0xb6, 0xb6, 0xc1//9

 };

/* interrupts 1. refresh rate 2. scroll rate */
/* End of interrupts */
int main(void)
{
unsigned char first_byte,count1,k=0,flag=0;
count1=0;
UBRR=25;
UCR= (1<<RXEN)|(1<<TXEN);
TIFR=TIFR;
TIMSK=1<<OCIE1A;
TCCR1B=5;
OCR1AH=10;
// OCR1AL=0;
_SEI();
while( !(USR&(1<<RXC))&& (q!=0 ));//timer1
will count till 2^16-1
first_byte=UDR;
if(first_byte == 42) //is *
  {
     while((count1<100) && (str1[k] != 63))  //
enter not pressed
       {
       if(USR & (1<<RXC))
          {
            str1[count1]=UDR;
            k=count1;
            count1++;

          }
       }
 flag=1;//if string entered in correct format ok
else flag remains 0 & prevoiusly stored string will
be displayed
 }
 if(str1[k] == 63)
 str1[k]=’\0’;
 address = 0x0001;
 x=0;
 if(flag==1)
 {do
 {
 EEWRITE(address,str1[x]);
 EEREAD( address, str+x);
 address++;
 x1=x;
 x++;
 }
 while( str1[x1] !=’\0’);
 count = x;
 }//end of if flag==1

 if(flag==0)
 {do
 {EEREAD( address, str+x);
 address++;
 x1=x;
 x++;
 }
 while(str[x1]!=’\0’);
 count = x;
 }//end of flag==0

 TIFR = TIFR;
 TCNT0 = 230;
 TIMSK  |= 1<<TOIE0 ;
 TCCR0 = 3;

    int addr, curr_col_temp,m;
    u08  value;
    outp(0xff,DDRA);
    outp(0xff,DDRB);
    outp(0xff,DDRC);
    outp(0xff,DDRD);
    char t;
    curr_col=0;
  setcol(-1);
 while(1)
 {
    cli();
 if( j == 2000)
 {
 //if( offset == 4 ) temp= offset;
 offset++;
  j=0;
 }//multiple of refresh(19),make para 1900 or 2000

 if(offset >=5)
 {offset=0;
 //   temp++;
 if(temp>=count)
 temp=0;
 }
   t = str[i];
     if( t>=65 && t<=91)
      addr  = (t-’A’)*5;//i is being incremented in
interrupt
      else
  if( t>=97 && t<=122) // c b/w a and z
     addr = (t-71)*5;
  else
     if( t>=48 && t<=57)  // c b/w 0 and 9
     addr = (t-48+52)*5;

  else
  addr = -325;
  curr_col_temp=(curr_col<5)?curr_col:curr_col%5;
     m = offset + curr_col_temp;
     if(m>=5) m=m-5;
     addr = addr + m;
     value = PRG_RDB(&leds[addr]);
     outp( value, PORTC);
   // curr_col = curr_col+1;
     setcol(curr_col);
     sei();
  }
  }

  void setcol( int col)
  {

  //initially switch off all coloumns
  switch (col)
  {
  case -1: PORTA=0x00;PORTB=0x00;PORTC=0xF
F;PORTD=0x00;break;
  case 0:  PORTA = 0x01; break;
  case 1:  PORTA = 0x02; break;
  case 2:  PORTA = 0x04; break;
  case 3:  PORTA = 0x08; break;
  case 4:  PORTA = 0x10; break;
  case 5:  PORTB = 0x01; break;
  case 6:  PORTB = 0x02; break;
  case 7:  PORTB = 0x04; break;
  case 8:  PORTB = 0x08; break;
  case 9:  PORTB = 0x10; break;
  case 10: PORTD = 0x04; break;
  case 11: PORTD = 0x08; break;
  case 12: PORTD = 0x10; break;
  case 13: PORTD = 0x20; break;
  case 14: PORTD = 0x40; break;
  case 15: PORTA = 0x80; break;
  case 16: PORTA = 0x40; break;
  case 17: PORTA = 0x20; break;
  case 18: PORTB = 0x40; break;
  case 19: PORTB = 0x20; break;
  default : break;
  }
  }

  void EEWRITE(int address, char value)
  {
  while(EECR&(1<<EEWE));
  eeprom_wb(address, value);
   EECR  |=(1<<EEMWE);
   EECR|=(1<<EEWE);
  }

  void  EEREAD (int address,char *val)
  {
   while(EECR&(1<<EEWE));
   EEAR=address;
   EECR=(1<<EERE);
  *val= EEDR;
  }
```

CHECK.C

```
// Program for checking Dot matrix Display //
#include<io.h>
#include<sig-avr.h>
#include<ina90.h>
int main(void)
{
DDRA=0xFF;
DDRB=0xFF;
DDRC=0XFF;
DDRD=0xFF;
PORTA=0XFF;
PORTB=0XFF;
PORTD=0XFF;
PORTC=0X00;
for(; ;)
{
}
}
```