8052.com
Single Board Computer
(SBC)

Hardware rev. 1.3

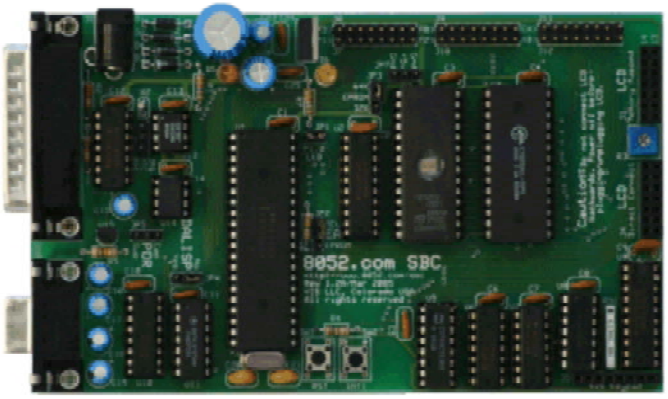
Manual revision 1.1
September 16, 2005

The contents of this document were written by Craig Steiner of 8052.com and Vault Information Services L.L.C.
Extracted from 8052.com and assembled into this file by Jon Ledbetter.

Revision History:

09/09/05 - Completed initial manual
09/13/05 - Added Information regarding the PS/2 Keyboard demo program
09/15/05 - Re-arranged topic order
09/15/05 - Fixed hyperlinks
09/15/05 - Initial Public Release Rev 1.0
09/16/05 - Added Information regarding the PS/2 Monitor and PS/2 Mouse demo programs
09/16/05 - Release Rev 1.1

Introduction to the 8052.com SBC



The 8052.com SBC is a single-board computer designed to be both useful and instructive in illustrating certain concepts commonly encountered in 8052-based development. Although many SBCs already exist, this SBC was designed specifically to form the basis of thorough technical discussions and tutorials consistent with those found on this website. Nothing was included in this SBC that does not serve a specific educational purpose and implementations that would confuse or unnecessarily complicate the design were intentionally omitted in favor of easy-to-understand approaches so that the important topics could be covered without being lost in the details.

This SBC was also designed with a worldwide market in mind. Recognizing that not all parts are easily available in all parts of the world, this SBC intentionally uses common parts that should not be difficult to acquire. In some cases the use of common parts may not have been the optimum design but in these cases the efficiency was sacrificed in favor of universality.

The board was developed with the Atmel AT89S8252/AT89S8253 and Dallas DS89C420 in mind but can be used with any 40-pin 8052 pin-compatible derivative including the traditional true-blue 8052, 8051, 8032, 8031, etc.; Additionally, this SBC will work with the new Atmel AT89S8253 that has been announced to replace the AT89S8252. Its feature-set is such that the user may use the SBC simply to learn and master the 8052 microcontroller but may also subsequently use it as a base for his or her own projects and designs.

8052.com SBC Features

Some notable features of the 8052.com SBC are:

1. **In-System Programming**. Atmel AT89S8252 and Dallas DS89C420 can be programmed in-system without having to remove the microcontroller and without need for a part programmer or special cables.
2. **Serial Port/UART**. Includes a single standard RS-232 compatible DB9 port that can be used to interface and communicate with external devices such as a standard PC. This is also useful in explaining the concept of serial communications.
3. **16x2 LCD**. Includes a 16 column by 2 row LCD that can be connected to the circuit either by direct connection (the lines being driven directly by port pins of the microcontroller on P1 and P3) or as a memory-mapped device (accessed with the MOVX instruction). This is useful in explaining communication with external devices by controlling individual I/O lines. It also is useful in explaining the concept of memory-mapped devices.
4. **4x4 Keypad**. A 4x4 matrix keypad which allows the user to input the numbers 0 through 9 plus provides 6 special function keys. This is useful in explaining the concept of key debounce.
5. **Real Time Clock**. The SBC includes a DS1307 Real-Time Clock. In addition to providing time-keeping capabilities this part is instrumental in demonstrating inter-chip communication using the I2C protocol.
6. **Serial EEPROM**. The SBC includes an Atmel AT25010A serial EEPROM which provides 128 bytes of non-volatile memory. More importantly it provides an opportunity to demonstrate inter-chip communication using the SPI protocol.
7. **EPROM**. The SBC may be configured to run code out of an EPROM inserted into the circuit. This is useful if the microcontroller being used does not have any on-chip code memory such as traditional 8032s, 8052s, etc. Those using a microcontroller with on-chip code memory may choose to omit the EPROM.
8. **Code RAM**. The SBC includes 32k of code RAM. This is RAM that has been wired such that the SBC will access it as RAM as well as code memory. This allows the user to download code into the RAM and execute it from RAM without having to burn a new EPROM or reprogram the microcontroller.
9. **Access to all I/O lines**. All data, address, and relevant signal lines are exposed on connector blocks such that the SBC may easily be expanded to circuits on external PCBs.
10. **Addressable LED**. P1.0 may be optionally connected to a LED for testing simple programs that cause the LED to flash.
11. **Dual reset circuits**. The SBC includes both a traditional resistor-capacitor (RC) network to provide the reset signal and also includes a more reliable MN13811 reset supervisor. Which solution is used to provide the reset signal is selectable with a jumper.

Why use an SBC?

SBCs were *very* popular before about the year 2000. Short of expensive in-circuit emulators (ICE), SBCs were one of the cheapest ways to quickly test 8052 firmware and/or experiment with the 8052 and attached devices. With the advent of low-priced evaluation boards from semiconductors companies such as Signal Laboratories, Texas Instruments, and Analog Devices, the demand for SBCs has dropped as many have found these evaluation boards to provide near-ICE functionality for under $200.

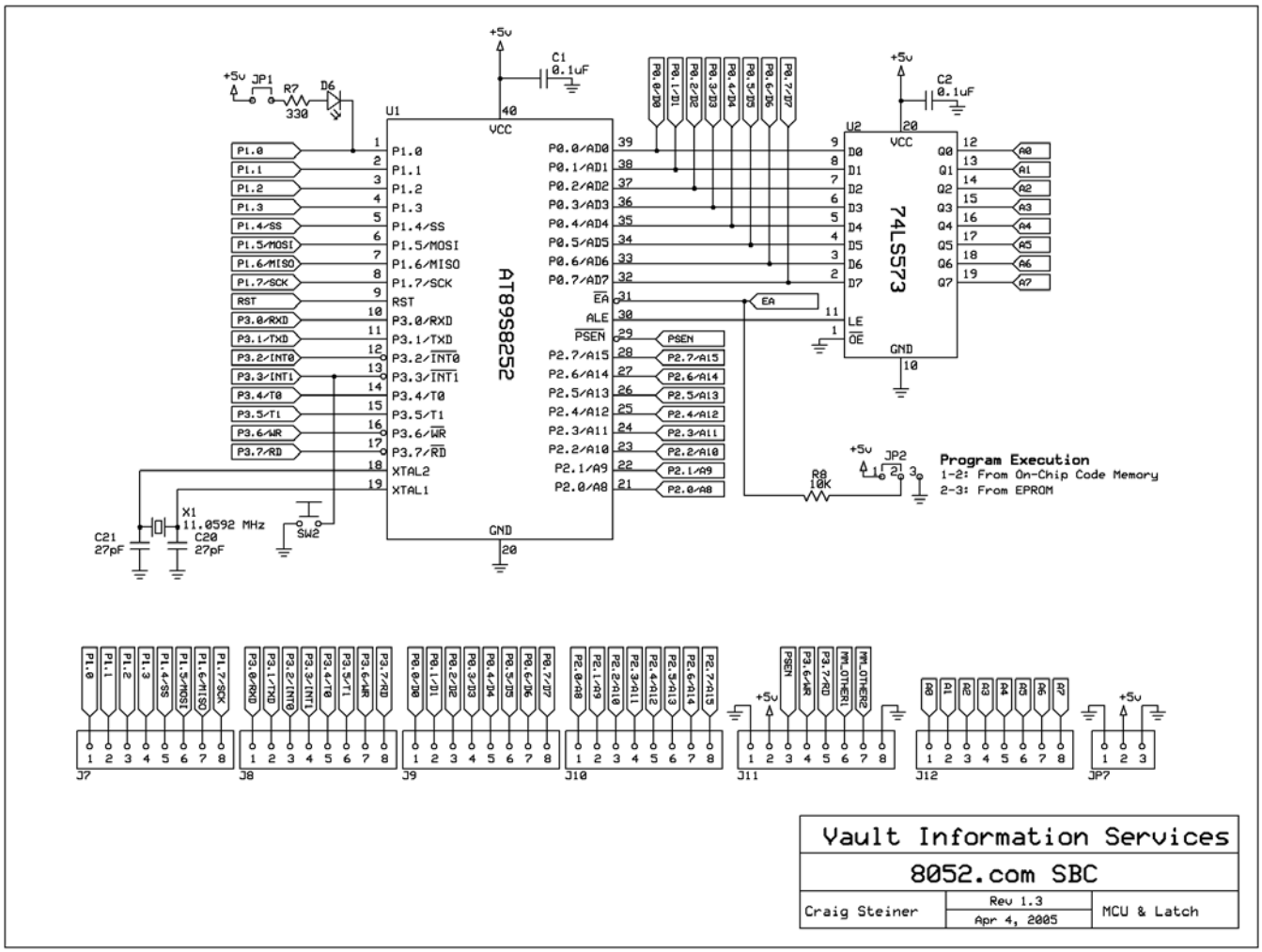
However, SBCs may still have a place in one's effort to learn and experiment with the 8052. An SBC provides the opportunity to understand what the 8052 is doing at the electrical level as it interfaces with external RAM, keypads, LCDs, serial ports, and buttons. Additionally, this SBC can be built by anyone with the parts and a soldering iron while most new evaluation boards must be purchased directly from the manufacturer and use surface mount technology that is not easily assembled by the novice or student. This SBC also utilizes parts that should be available in virtually any country without having to import parts from foreign sources. And, of course, this SBC can be used with virtually any 8052-compatible 40-pin DIP microcontroller so the user can actually build the SBC with almost any 8052 derivative that is available locally.

In short, using an SBC--and especially *building* one--forces an understanding of the 8052's architecture and external electrical connections that might be missed if the user immediately uses a modern evaluation board without understanding the underlying fundamentals.

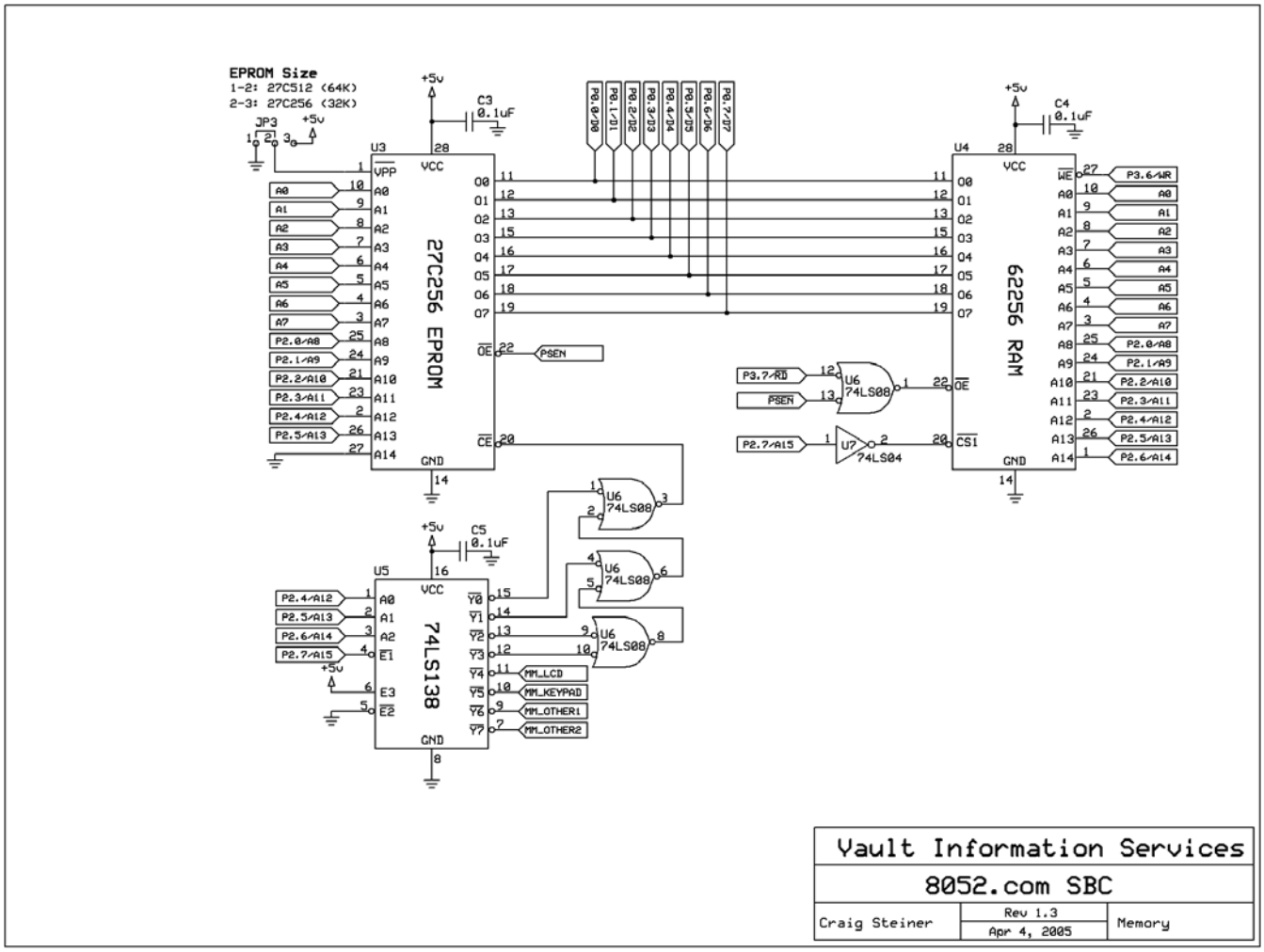
8052.com SBC Schematics

This page includes the schematics for the 8052.com SBC. You may build a functioning version of the SBC by building based on these schematics. Click on each schematic page for a full-size version.

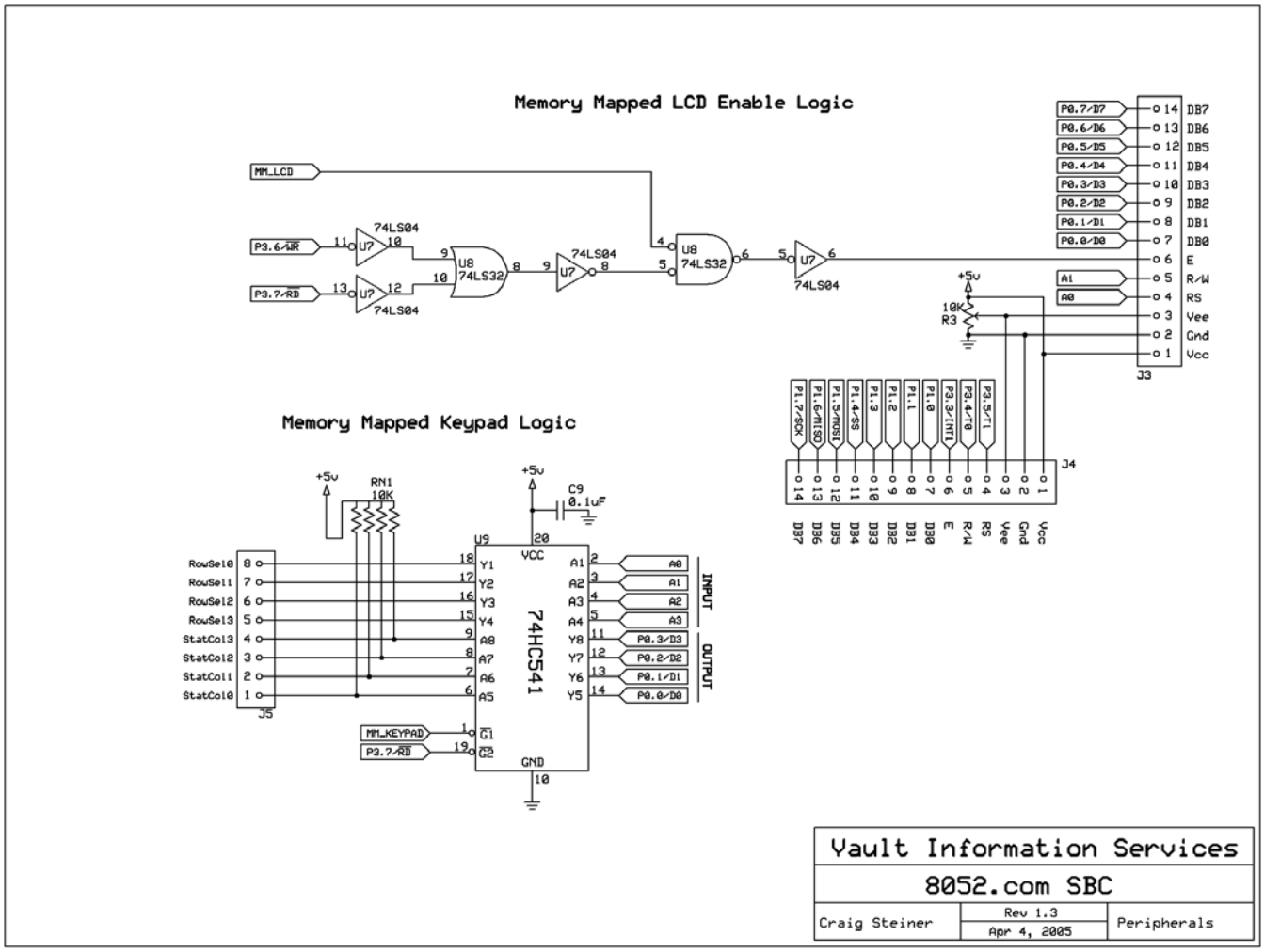
Microcontroller & Memory Latch



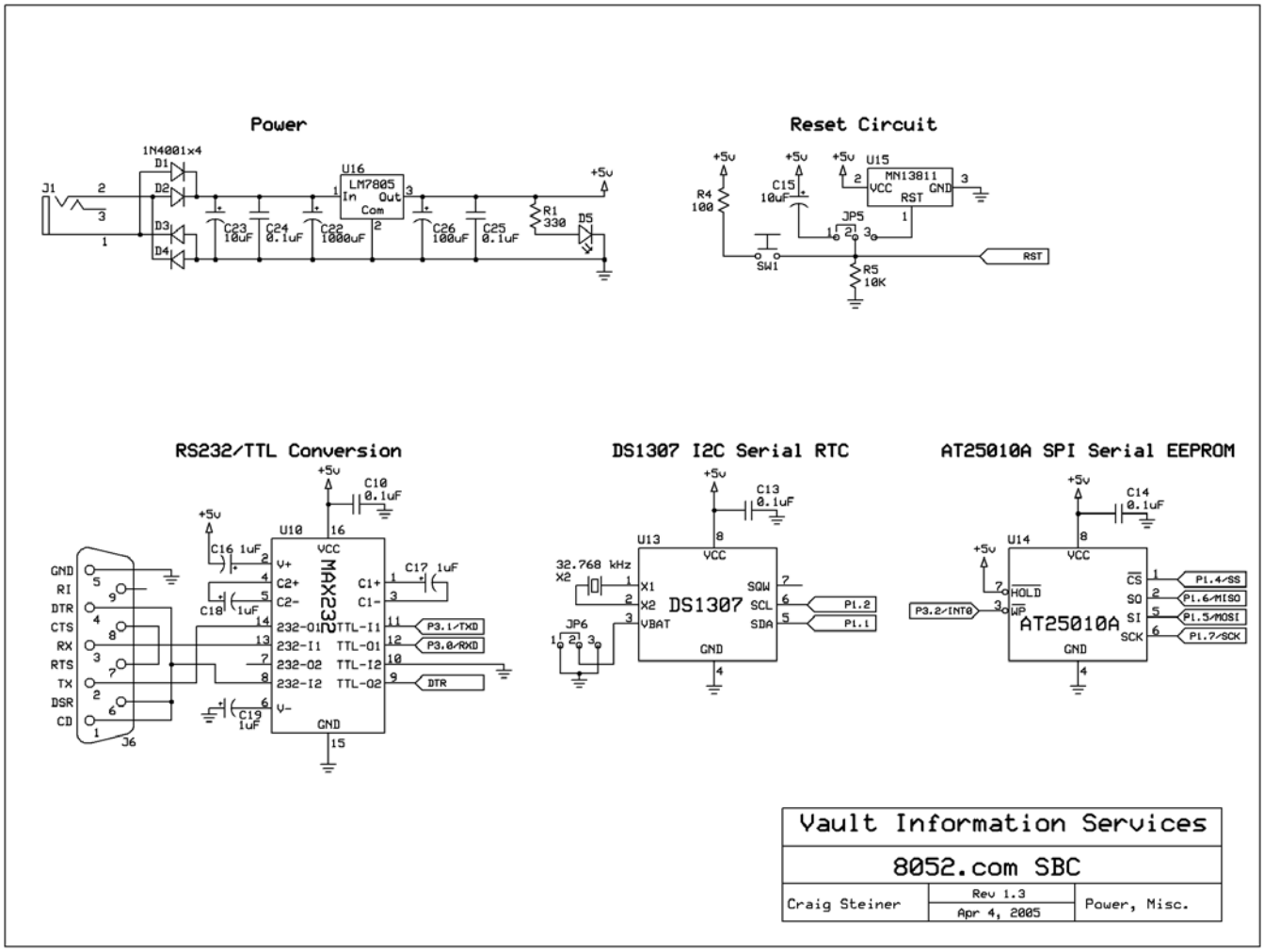
Memory (RAM & EPROM)



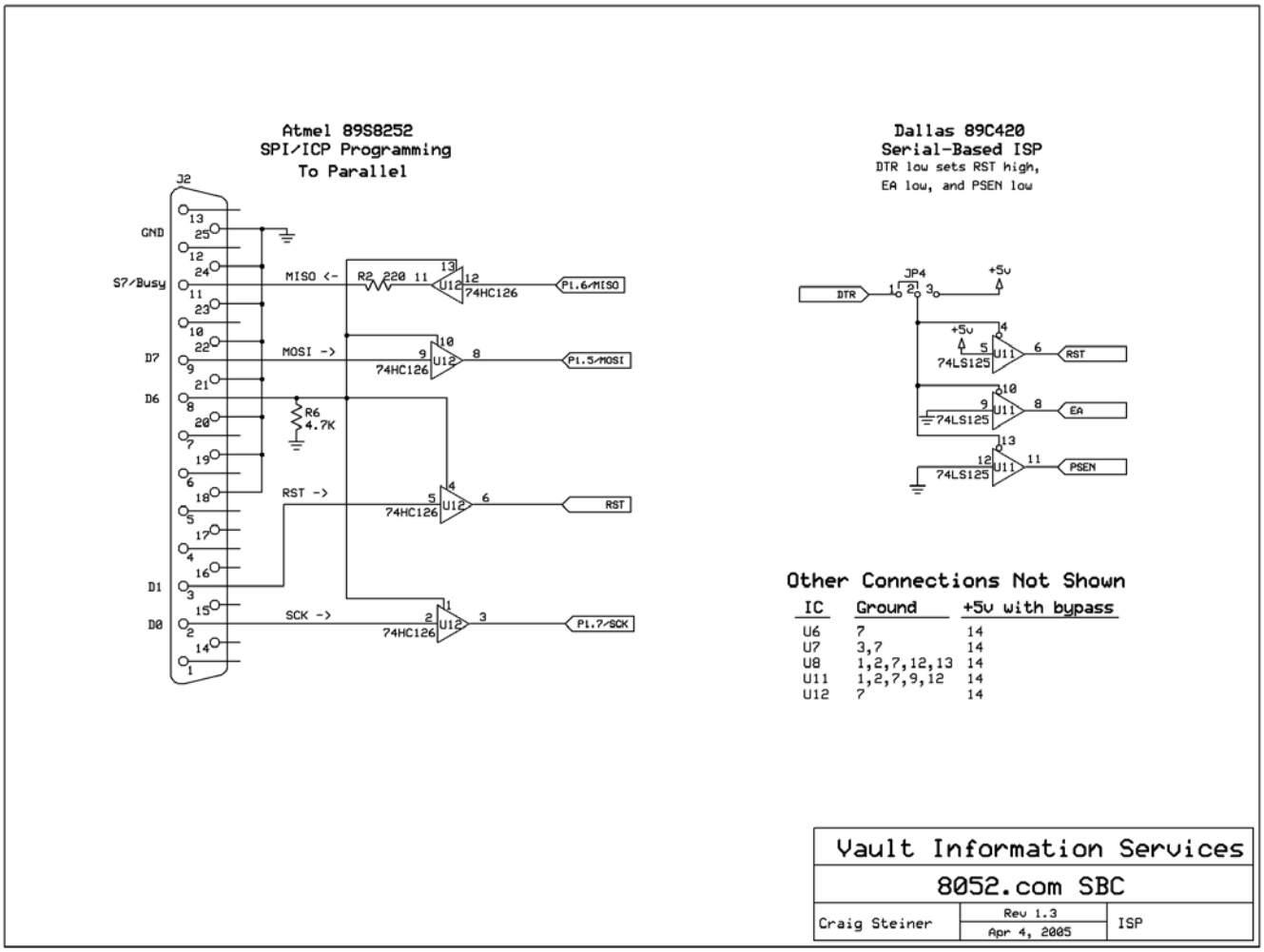
Memory-Mapped LCD & Keypad



Power & Misc.


Power
1N4001x4
D1
D2
D3
D4
J1
U16
LM7805
In
Out
Com
C23 10uF
C24 0.1uF
C22 1000uF
C26 100uF
C25 0.1uF
R1 330
D5
+5v
Reset Circuit
R4 100
C15 10uF
U15
MN13811
VCC
RST
GND
JP5
SW1
R5 10K
RST
RS232/TTL Conversion
U10
MAX232
VCC
GND
V+
V-
C1+
C1-
C2+
C2-
C10 0.1uF
C16 1uF
C17 1uF
C18 1uF
C19 1uF
232-O1
232-I1
232-O2
232-I2
TTL-I1
TTL-O1
TTL-I2
TTL-O2
P3.1/TXD
P3.0/RXD
DTR
GND
RI
DTR
CTS
RX
RTS
TX
DSR
CD
J6
DS1307 I2C Serial RTC
U13
DS1307
VCC
GND
X1
X2
VBAT
SQW
SCL
SDA
32.768 kHz
X2
JP6
C13 0.1uF
P1.2
P1.1
AT25010A SPI Serial EEPROM
U14
AT25010A
VCC
GND
HOLD
WP
CS
SO
SI
SCK
P3.2/INT0
P1.4/SS
P1.6/MISO
P1.5/MOSI
P1.7/SCK
C14 0.1uF
Vault Information Services
8052.com SBC
Craig Steiner
Rev 1.3
Apr 4, 2005
Power, Misc.

In-System Programming (ISP) Schematic



Bill of Materials

The following parts are needed to build the 8052.com SBC described above:

Reference	Quantity	Part	Digikey Part Number	Datasheet
C1-C14, C24,C25	16	0.1uF ceramic capacitor	1338PH-ND	
C15,C23	2	10uF electrolytic capacitor	P5134-ND	
C16-C19	4	1uF electrolytic capacitor	P10312-ND	
C20,C21	2	27pF ceramic capacitor	P10797-ND	
C22	1	1000uF electrolytic capacitor	P10253-ND	
C26	1	100uF capacitor	P11198-ND	
D1-D4	4	1N4001 diode	1N4001GICT-ND	
D5	1	Red LED	404-1090-ND	
D6	1	Yellow LED	404-1092-ND	
J1	1	2.1mm power connector	CP-202A-ND/CP-102AH-ND	
J2	1	DB25 male conector	A23285-ND	
J3,J4	2	14-pin Female Header	S4207-ND	
J5	1	8-pin Female Header	S4008-ND	
J6	1	DB6 DB9 female RS232	A23303-ND	
J7-J12	6	8-pin SIPs		
JP1	1	2-pin jumper		
JP2-JP6	5	3-pin jumper		
R1,R7	2	330	P330BACT-ND	
R2	1	220	P220BACT-ND	
R3	1	10K Variable Resistor	3362U-103-ND	
R5,R6,R8	3	10K	P10KBACT-ND	
R4	1	100	P100BACT-ND	
RN1	1	6.8k 6-pin resistor network	750-61-R6.8K-ND	
SW1,SW2	2	Momentary push button, normally open	P8006S-ND	
U1	1	Atmel AT89S8253	AT89S8253-24PI-ND	Datasheet
U2	1	74LS573 octal latch	296-1596-5-ND	Datasheet
U3	1	27C64 UV EPROM (optional)	497-1685-5-ND	Datasheet

U4	1	62256 8x32k RAM (or 6264)	428-1080-ND	Datasheet
U5	1	74LS138 3x8 decoder/MUX	296-1639-5-ND	Datasheet
U6	1	74LS08 Quad 2-Input AND Gate	296-1633-5-ND	Datasheet
U7	1	74LS32 Quad 2-Input OR Gate	296-1658-5-ND	Datasheet
U8	1	74LS04 Hex Inverter	296-1629-5-ND	Datasheet
U9	1	74HC541 Octal Buffer/Driver	296-12813-5-ND	Datasheet
U10	1	MAX232 Dual RS232 Driver/Receiver	296-1402-5-ND	Datasheet
U11	1	74LS125 Quad Tri-State Buffer Active Low	296-1638-5-ND	Datasheet
U12	1	74HC126 Quad Tri-State Buffer Active High	296-8221-5-ND	Datasheet
U13	1	DS1307 real time clock (RTC)	DS1307-ND	Datasheet
U14	1	AT25010A 128x8 Serial EEPROM	AT25010A-10PI-2.7-ND	Datasheet
U15	1	MN13811 4.6V Reset Supervisor	MN13811-U-ND	Datasheet
U16	1	LM7805 5V voltage regulator	296-1974-5-ND	Datasheet
X1	1	11.0592 MHz crystal	X139-ND	
X2	1	32.768 kHz crystal	300-1002-ND	
	1	40-pin IC Socket	A9440-ND	
	2	28-pin IC Socket	A9428-ND	
	2	20-pin IC Socket	A9420-ND	
	2	16-pin IC Socket	A9416-ND	
	5	14-pin IC Socket	A9414-ND	
	2	8-pin IC Socket	A9408-ND	
	3	36-pin breakaway headers for J7-J12, JP1-JP6	WM6436-ND	
	1	16x2 LCD (left side 2x7 connector)	67-1770-ND or 67-1780-ND	
	1	4x4 Keypad	GH5003-ND	
	1	5W Heatsink for 7805	HS121-ND	
	1	Wall transformer, 12VAC 750MA, 2.1mm	MT7119-ND	

	4	Bumpons Hemispheres .50X.14 Clear	SJ5312-7-ND	

Microcontroller Port Pin Usage

Name	Pin	Function/Connection
Port 1		
P1.0	**1**	**LED/Direct LCD DB0**. Used to control the D6 LED if JP1 is connected. Also used as DB0 when the LCD is connected as a direct-connection.
P1.1	**2**	**I2C SDA/Direct LCD DB1**. Used as the SDA line to access the DS1307 via I2C. Also used as DB1 when the LCD is connected as a direct-connection.
P1.2	**3**	**I2C SCL/Direct LCD DB2**. Used as the SCL line to access the DS1307 via I2C. Also used as DB2 when the LCD is connected as a direct-connection.
P1.3	**4**	**Direct LCD DB3**. Used as DB2 when the LCD is connected as a direct-connection.
P1.4	**5**	**SPI SS/Direct LCD DB4**. Used to select the AT25010A for SPI access. Also used as DB4 when the LCD is connected as a direct-connection.
P1.5	**6**	**SPI MOSI/Direct LCD DB5**. Used as MOSI when communicating with the AT25010A via SPI and when in AT89S8252 ISP mode. Also used as DB5 when the LCD is connected as a direct-connection.
P1.6	**7**	**SPI MISO/Direct LCD DB6**. Used as MISO when communicating with the AT25010A via SPI and when in AT89S8252 ISP mode. Also used as DB6 when the LCD is connected as a direct-connection.
P1.7	**8**	**SPI SCK/Direct LCD DB7**. Used as SCK when communicating with the AT25010A via SPI and when in AT89S8252 ISP mode. Also used as DB7 when the LCD is connected as a direct-connection.
Port 3		
P3.0	**10**	**RXD**. Connected to the MAX232 for RS-232 serial communication.
P3.1	**11**	**TXD**. Connected to the MAX232 for RS-232 serial communication.
P3.2	**12**	**AT25010A Write Protect**. Connected to the -WP pin of the AT25010A serial EEPROM, active low. When low the AT25010A will be write protected. When high the AT25010A will *not* be write protected.
P3.3	**13**	**Direct LCD E/Interrupt 1 Button**. Used to drive the LCD E line when the LCD is connected as a direct-connection. Also connected to momentary push-button. If button is pressed this line will be brought low which may activate an external 1 interrupt.
P3.4	**14**	**Direct LCD RW**. Used to drive the LCD RW line when the LCD is connected as a direct-connection. Not used if LCD is connected as memory-mapped device.
P3.5	**15**	**Direct LCD RS**. Used to drive the LCD RS line when the LCD is connected as a direct-connection.
P3.6	**16**	**WR**. Asserted automatically by the microcontroller to write to external memory/memory-mapped devices with the MOVX instruction.
P3.7	**17**	**RD**. Asserted automatically by the microcontroller to read from external memory/memory-mapped devices with the MOVX instruction.
Port 2 and Port 0		Both ports 2 and 0 are connected to the address bus and are driven automatically by the microcontroller. None of the pins are used by the SBC for any other functions.

Jumper Settings

Jumper	Function/Description
JP1	**P1.0 LED Enable**. If JP1 is connected then P1.0 will control LED D6.
JP2	**External Access**. If JP2 is jumpered to pin 1 (Vcc) then the microcontroller will execute code from the microcontroller's on-chip program memory. If JP2 is jumpered to pin 3 (ground) then the microcontroller will execute code from external EPROM. In this case be sure that a programmed EPROM is installed in the circuit.
JP3	**Eprom.** Jumper pins 1 and 2 for 64K or pins 2 and 3 for 32K
JP4	**Dallas ISP**. If JP4 is jumpered between pins 1 and 2, the DTR pin of the DB9 RS-232 port will drive enable pins of U11 and thereby permit Dallas DS89C420 ISP programming. If JP4 is jumpered between pins 2 and 3 then U11 is permanently disabled and Dallas ISP will not be possible.
JP5	**Reset Selection**. If JP5 is jumpered to pin 1 then the microcontroller's reset pin will be driven by a traditional reset-capacitor network with capacitor C15. If JP5 is jumpered to pin 3 then the MN13811 reset supervisor will be used to generate the reset signal.
JP6	**RTC Battery Backup**. JP6 may be used to attach a battery to the DS1307 RTC to drive the clock when the SBC's power is removed. The center pin is Vcc while the left and right pins are ground.
JP7	**Power.** A place to hook into the power supply. Pins 1 and 3 are ground. Pin 2 is +5V

Memory Map

Address Range	Function		
0000h - 3FFFh	**EPROM**, if JP2 selects EPROM code execution.		
4000h - 4FFFh	**LCD Memory-Mapped**. Specific addresses access memory-mapped LCD.		
	4000h	Write Command to LCD	
	4001h	Write Text to LCD	
	4002h	Read Command Register from LCD	
	4003h	Read Text from LCD	
5000h - 5FFFh	**Keypad Memory-Mapped**. Specific address access memory-mapped keypad.		
	500Eh	Read keypad row 0	
	500Dh	Read keypad row 1	
	500Bh	Read keypad row 2	
	5007h	Read keypad row 3	
6000h - 6FFFh	**Other 1**. Can be used to enable user memory-mapped devices connected via pin 6 of J11.		
7000h - 7FFFh	**Other 2**. Can be used to enable user memory-mapped devices connected via pin 7 of J11.		
8000h - FFFFh	**Program/RAM**. Can serve as XRAM or RAM-based program memory. If a 32k RAM IC is used (62256) then this RAM will occupy all of 8000h - FFFFh. If an 8k RAM IC is used (6264) then this RAM will occupy E000h - FFFFh.		

Notes:

EPROM. The EPROM socket is made available primarily for those that wish to use the SBC with a microcontroller that does not include on-chip code memory. In this case the user program or monitor must be loaded into an EPROM and JP2 must be configured to the 2-3 setting.

Non-EPROM Use. If a microcontroller with on-chip code memory is being used (such as an Atmel AT89S8252, Dallas DS89C420, etc.) then the EPROM may be completely omitted.

Programming the 8052.com SBC with ISP

The 8052.com SBC was designed to permit in-system programming (ISP) with the Atmel AT89S8252, AT89S8253, and the Dallas DS89C420 microcontrollers. This means that if the SBC is used with one of these microcontrollers, the on-chip code memory of the microcontroller may be programmed, read, and erased without anything more than a serial or parallel cable and appropriate software.

In-System Programming and Atmel Microcontrollers

If your 8052.com SBC has an Atmel AT89S8252 or an Atmel AT89S8253 then you may use the VisISP-52 software to perform in-system programming with the SBC. This Windows software was developed specifically to download firmware to the 8052.com SBC. VisISP-52 requires a standard 25-pin parallel cable to connect the PC to the 8052.com SBC.

In-System Programming and Dallas DS89C420

If your 8052.com SBC has a Dallas DS89C420 then you may use the Microcontroller Tool Kit (MTK) application from Maxim-Dallas Semiconductor to perform the in-system programming. A full explanation of how to do this is explained in this application note. MTK requires a standard serial cable to connect the PC to the 8052.com SBC.

Introduction to VisISP-52

VisISP-52 is a Windows application that will run on all modern versions of Windows (Win95, Win98, Win2000, WinME, and WinXP) and which will allow supported microcontrollers to be programmed "in-system". It was developed specifically to program the Atmel AT89S8252 and AT89S8253 installed in an 8052.com SBC but should work equally well for any other design that uses a compatible ISP and one of the supported microcontrollers.

Installing VisISP-52

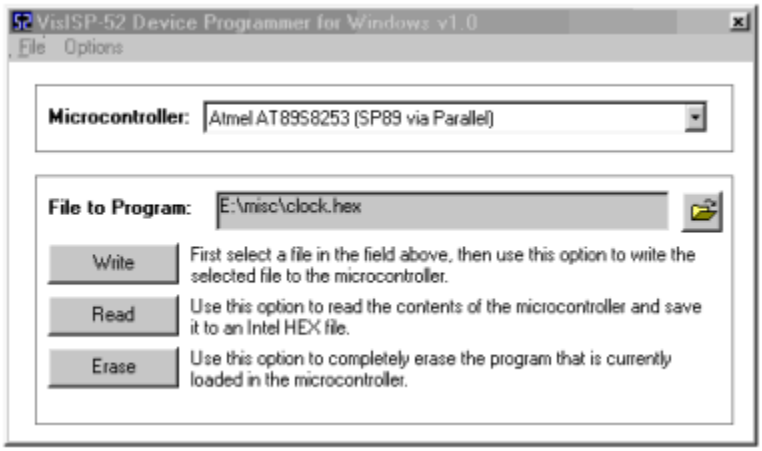
To install VisISP-52 on your computer, simply download the following installation program and run it. This will install the application on your computer and add the appropriate icons to your "Programs" menu. You may then launch the program normally.

Download VisISP-52 v1.0 Installation Program (3.4MB)

Note: You may need to be logged in as 'Administrator' in Win2000, WindowsXP, and Windows NT in order to install the program.

Using VisISP-52 to Load Programs on the Microcontroller

VisISP-52's interface is very straight-forward as is pictured below. A single window with minimal menu options provides access to the primary features of the program: Writing new firmware to the microcontroller, reading the firmware that is currently in the microcontroller, and erasing the microcontroller's memory.



Before proceeding, make sure you've selected the right microcontroller in the "Microcontroller" drop-down window. At this time, VisISP-52 only supports the Atmel AT89S8252 and AT89S8253.

To Write Firmware to the Microcontroller: In order to write firmware to the microcontroller, first select the HEX file you wish to download by pressing the "file" button to the right of the "File to Program" prompt. You will be allowed to select a HEX file which will be loaded into memory. Once you've selected the file, you may download it to the microcontroller by pressing the "Write" button. A progress window will be displayed as the file is transferred to the microcontroller.

To Read Firmware from the Microcontroller: You may read the contents of a microcontroller by pressing the "Read" button. This will download the program memory from the microcontroller to the PC. When the download is complete you will be prompted to enter a name for the HEX file that you wish to write the data to. Note that you will not be able to download firmware if the microcontroller has a security "lock bit" set to prohibit such downloading.

To Erase the Microcontroller: You may erase the microcontroller by pressing the "Erase" button. The erase process is very quick but is subsequently followed by a verification phase in which VisISP-52 will read the entire contents of the microcontroller's memory to verify that it was, in fact, erased. You may cancel this verification process at any time without affecting the erase process which is

completed almost immediately.

Options: The "Options" menu allows you to configure the ports (LPT port to which the SBC hardware is connected). You may also toggle the "Quick Program Mode." This option only applies to the AT89S8252. When the Quick Program Mode is selected, VisISP-52 will only download those bytes which are contained in the selected HEX file but will leave all other bytes alone. This means that if the HEX file only contains 2k of data, only 2k will be written to the AT89S8252 while the other 6k of the AT89S8252 will not be affected. If this option is *not* selected then the remaining 6k will be filled with FFh data.

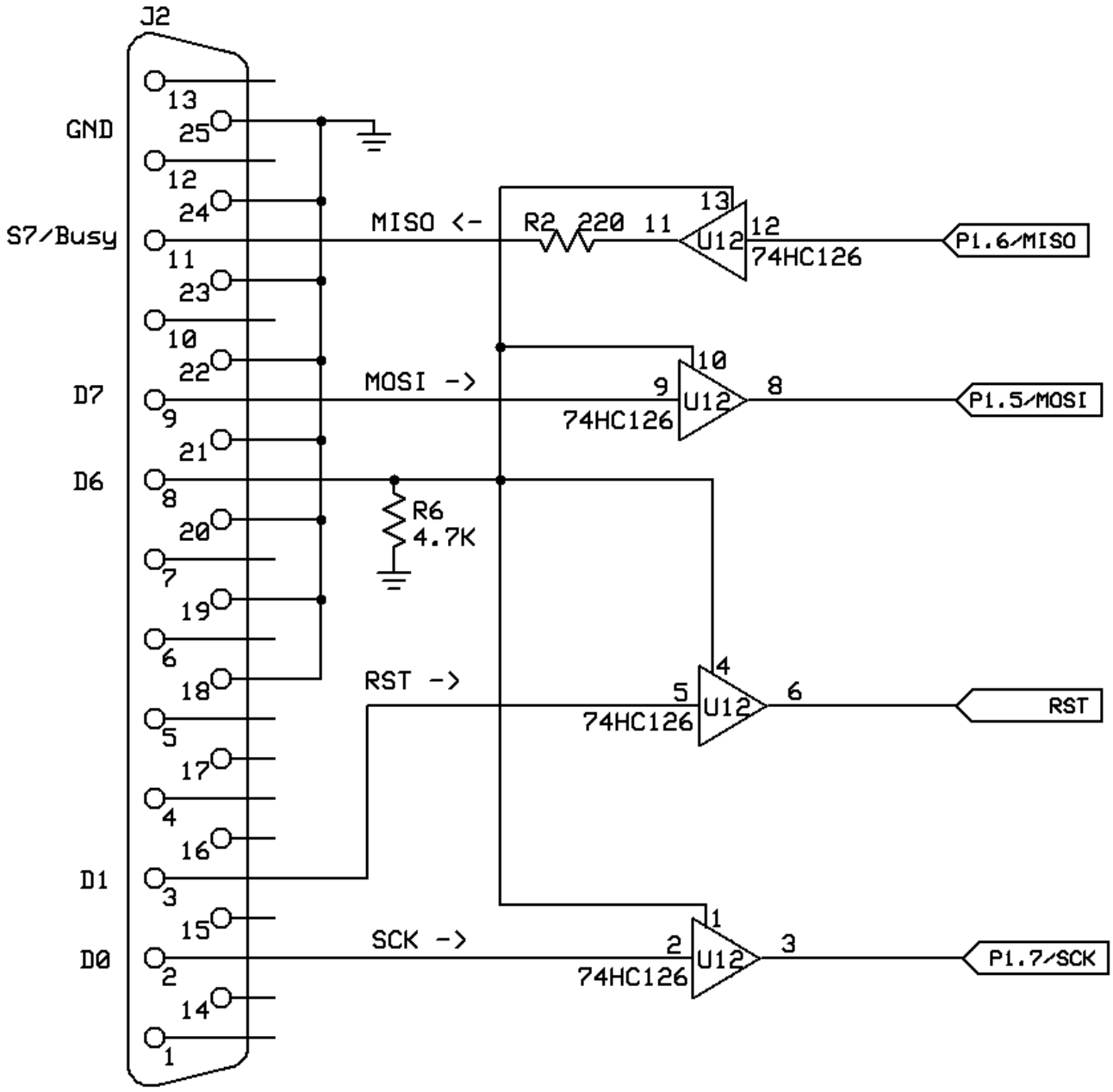
ISP Hardware

Since the ISP circuit of the 8052.com SBC was originally developed to operate with the AT89S8252 and "SP89", VisISP-52 will only work with ISP circuits that are compatible with the original SP89 parallel port pin-out. The 8052.com SBC meets this compatability requirement.
If you are going to include an ISP circuit in your own hardware and you wish it to be compatible with VisISP-52, use the following pin-out:

Parallel Port Pin	Microcontroller Connection/Function
2 (D0)	**SCK/P1.7**. The D0 pin of the parallel port drives the SCK/P1.7 pin on the microcontroller. SCK is the SPI clock which is generated by VisISP-52 and allows the PC to clock data into and out of the microcontroller.
3 (D1)	**RST/pin 9**. The D1 pin of the parallel port drives the RST pin of the microcontroller. This allows VisISP-52 to put the microcontroller into reset mode. This is necessary because ISP occurs with reset driven high.
8 (D6)	**ISP Enable**. The D6 pin of the parallel port should drive a buffer IC such as the 74HC126 such that the other input lines (D0, D1, and D7) are only connected to the microcontroller when this line is driven high by VisISP-52. When this line is low, the other three input lines should not be connected to their respective connections on the microcontroller. Note that a pull-down resistor between the parallel port and the 74HC126 may be necessary so that ISP is disabled when no parallel cable is connected to the ISP circuit.
9 (D7)	**MOSI/P1.5**. The D7 pin of the parallel port should drive the MOSI pin of the microcontroller. This is the pin that VisISP-52 uses to actually send data bits to the microcontroller in conjunction with the SCK clock line.
11 (S7/Busy)	**MISO/P1.6**. The S7/Busy pin of the parallel port should be driven by the MISO pin of the microcontroller. Data sent from the microcontroller to VisISP-52 is sent on this line. Note that a termination resistor of approximately 220 Ohms is recommended between the 74HC126 latch and the parallel port pin.
18-25 (Ground)	**Ground**. As many pins between 18 and 25 of the parallel port should be connected to ground on the microcontroller circuit. While a single ground line *may* work, connecting additional ground lines may decrease problems due to crosstalk between the data lines. This is especially true if the parallel cable is a flat ribbon-cable.

The following schematic describes in detail how such a VisISP-52-compatible ISP circuit should be designed. This is the designed that is implemented in the 8052.com SBC. Click on the schematic for a larger version.

ISP Circuit compatible with VisISP-52



VisISP-52 History/Background

VisISP-52 originated as a Windows application based on the design of "SP89" ISP software that was originally developed by Steven Bolt. That software works fine in Linux and probably works fine in Windows; however, the install process was not particularly friendly for Windows NT and Windows XP due to additional software that needed to be installed in order to access the parallel port which is protected in those operating systems.

Additionally, SP89 only supports the AT89S8252 microcontroller which has now been discontinued and replaced by the AT89S8253. Thus it is not possible to use SP89 with the newer AT89S8253 that is shipping with the 8052.com SBC.
VisISP-52 solves both of these problems by providing a single easy-to-install application that will work on any modern Windows operating system. It also supports both the AT89S8252 and the newer AT89S8253.

VisISP-52 does not use any code from SP89, although the code for VisISP-52 was developed with the benefit of looking at the SP89 source code to get a good idea of how the AT89S8252 communication worked. The AT89S8253 code was developed from scratch with some limited technical support from Atmel. The software also includes code based on the parallel port code from Logix4U.
This software is free and may be distributed without restriction. The source code is not currently available but will probably be made available in the future. The only the source code is not being made available at this time is because the code is not as presentable or as easy-to-follow as it should be and will probably confuse the curious developer rather than help him.

The SP89 In-System Programmer is a software package that can program many Atmel parts, including the AT89S8252, using a connection to the microcontroller via the PC's parallel port assuming the microcontroller is part of a functioning ISP-friendly circuit.

Software Downloads

- **Windows Version**: SP89 version 0.7.2 for Windows 95, 98, Windows 3.1, and MS-DOS
- **Windows NT/XP Version**: SP89 version 0.7.1 for Windows NT and XP
- **Windows NT/XP Add-On**: Windows NT/XP users also need this file
- **Linux Version**: SP89 version 0.7.2 for Linux

Credits & Disclaimers

The SP89 software was written and is maintained by Steven Bolt and is distributed under the GNU General Public License. The latest version of SP89 can be found here. An unmodified copy of the SP89 software is provided here free of charge for the convenience of the users of 8052.com in accordance with the GNU General Public License and having requested specific permission from the author. This software comes with no guarantee from its author nor from 8052.com or Vault Information Services LLC. Use of this software and/or the hardware described on this page is at the user's own risk.

Powering up SBCMON

When power is applied to the SBC loaded with SBCMON, the yellow LED (D6) will flash on and off three times. This serves two purposes:

1. It confirms that the SBCMON program is loaded correctly on the SBC and that the SBC is executing the monitor. This is useful since if there doesn't seem to be any communication between the PC and SBCMON, the flashing yellow LED will confirm that SBCMON is, in fact, executing. If the yellow LED flashes when the SBC is powered-up but nothing appears in the PC terminal program, the problem is almost certainly something to do with communicationsâ€”either the wrong type of cable or the PC's terminal configuration is incorrect.
2. It provides a delay of approximately 600 milliseconds. This is useful because the Atmel AT89S8252 has a published errata which indicates that problems may occur if a program accesses the SPI pins (P1.5, P1.6, and P1.7) during the first 500 milliseconds following an ISP process. The 600ms delay eliminates any possibility that these pins will be manipulated within this period.

Once power has been applied and the yellow LED has flashed on and off three times, SBCMON will send the message:

```
SBCMON v1.3.0 Initialized.
(Detected 32k of XRAM at 8000h)
```

This informs the user that the SBC has powered-up or has been reset. It also indicates whether or not SBCMON found XRAM that can be used to load programs. The monitor then enters the main menu mode.

SBCMON Main Menu

When SBCMON is ready to receive a command from the user it will display the following message:

```
SBC Ready>
```

At this point the user may enter any valid command. A command consists of a single letter and zero or more additional parameters. The command is executed when the user presses ENTER.
When commands include a parameter, the first parameter is always entered immediately after the command letter with no space between them. For example:

X8000 - Correct
X 8000 - Incorrect, there should not be a space between the 'X' and '2000'

The previously executed instruction may be executed again by hitting the ESCAPE key alone on a line. Commands are case-sensitive. This means the 'X' command is not the same as 'x'. All commands are upper-case except for one (the lower-case "w" command).
The following table lists all the commands that may be executed in SBCMON. Click on the link for each command for detailed syntax and additional information.

Command	Description
A[xxxx]	**Assembler**. Invokes mini-assembler to allow quick assembly of small programs. If address 'xxxx' is provided in hex, assembly occurs starting at that address. If no address is provided, assembly occurs starting at the next address.
C[ss nn hh w dd mm yy]	**Read/Set DS1307 RTC**. Reads the current date/time from the RTC, or sets it to the date/time specified in the parameters.
E[text]	**Read/Write Serial EEPROM**. If 'E' command is issued alone, display the current contents of the AT25010A serial EEPROM. If the 'text' parameter is given, writes that text to the serial EEPROM.
Ixx[,yy][=zz[,zz,etc]]	**Read/Write IRAM**. Displays or sets internal RAM.
K[D]	**Keypad Test**. Echos keypresses from the 4x4 keypad to the terminal. Keypad debounce is disabled if the 'D' parameter is given.
L	**Load Hex File**. Allows SBCMON to receive a HEX file and load it into XRAM, normally at 8000h.
M[0|1|2][I]	**Mode for LCD**. Configures whether 16x2 LCD is attached as direct connect in 4-bit or 8-bit mode, or attached in memory-mapped mode. The "I" parameter will send initialization commands to the LCD.
Q	**Quick Load & Run**. Receives a program in HEX format and immediately executes code at 8000h.
Rxxxx	**Run Code**. Causes SBCMON to execute code at the specified hex address 'xxxx'. The code that is executed should end with a final RET instruction to return to SBCMON.
V[F]	**Verify XRAM**. Executes a test of XRAM to verify that it is functioning reliably. The 'F' command will execute a much more thorough test that will take significantly longer to complete.
Wtext	**Write text to LCD**. Writes the specified text to the LCD.
w[aa bb cc]	**Write commands to LCD**. Writes the specified hex commands to the LCD.
Xxxxx[,yyyy][=zz[,zz,etc]]	**Read/Write XRAM**. Displays or sets external RAM.

SBCMON Library Routines

Programs which are run from within SBCMON by loading them with the "L" command and executing them with the "R" command may make calls to a number of library routines that are contained within SBCMON itself. Using these routines saves the developer from having to rewrite the routines and avoid using code memory for a routine that is already contained within SBCMON.
For example, a simple program to send the digits "45" to the serial port could be:

```
SendSerialHexByte EQU 0044h     ;Define SendSerialHexByte to address in SBCMON libraries
MOV A,#45h                      ;Load the accumulator with the byte that'll be sent to serial port
LCALL SendSerialHexByte         ;Call the SBCMON library routine
```

The program may define each routine that it uses with an EQU definition and subsequently make calls to that routine. Click on each routine below for the parameters that must be set or which are returned by each routine.

Address	Routine	Description
0041	SendSerial	Sends null-terminated string in code memory to serial port
0044	SendSerialHexByte	Sends value in ACC to serial port as two hexadecimal digits
0047	SendSerialByte	Sends character in ACC to the serial port
004A	GetSerialByte	Waits for character on serial port, returns it in ACC
004D	GetSerialLine	Receives an enter-terminated line of input from serial port
0050	SendLCDText	Sends a character to the LCD as text
0053	SendLCDCommand	Sends byte to the LCD as an LCD command
0056	I2C_ReadByte	Reads one byte from the I2C bus
0059	I2C_SendByte	Sends one byte to the I2C bus
005C	I2C_Reset	Resets the I2C bus
005F	I2C_Start	Starts an I2C conversation
0062	I2C_Stop	Ends an I2C conversation
0065	SPI_ReadByte	Reads one byte from the SPI bus
0068	SPI_SendByte	Sends one byte to the SPI bus
006B	ByteTo2Ascii	Converts accumulator to two ASCII digits
006E	HexToNibble	Converts ASCII value in accumulator to value 00h and 0Fh
0071	GetHexValue	Converts hex digits in input buffer to values
0074	GetSerialHex	Reads 2 hex digits from serial port, converts to value in ACC
0077	ToUpper	Converts the character in ACC to upper-case
007A	SendDecimalR4567	Sends value in R4-R7 as decimal value
007D	DivideBy10	Divides 32-bit value by 10
0080	Check4BytesForZero	Verifies that a 32-bit value is zero
0083	ShiftR765Left	Shifts registers R5 (MSB) through R7 (LSB) left1 bit
0086	AddR67to23	Adds 16-bit value in R6 and R7 to 16-bit value in R2 and R3
0089	AddR4567to0123	Adds 32-bit value R4 through R7 to R0 through R3
008C	InitializeLCD	Sends necessary commands to initialize LCD
008F	ReadLCDStatus	Reads the LCD status (cursor position and busy flag)

0092	ReadLCDText	Reads the LCD text/memory at current cursor position
0095	SetDptrMMKey1	Set DPTR to memory-mapped address for read keypad row 1
0098	SetDptrMMKey2	Set DPTR to memory-mapped address for read keypad row 2
009B	SetDptrMMKey3	Set DPTR to memory-mapped address for read keypad row 3
009E	SetDptrMMKey4	Set DPTR to memory-mapped address for read keypad row 4
00A1	SetDptrMMLCdWC	Set DPTR to memory-mapped address for write LCD cmd
00A4	SetDptrMMLcdWT	Set DPTR to memory-mapped address for write LCD text
00A7	SetDptrMMLcdRC	Set DPTR to memory-mapped address for read LCD cmd
00AA	SetDptrMMLcdRT	Set DPTR to memory-mapped address for read LCD text
00AD	GetSBCMONVersion	Returns the version of SBCMON
00B0	GetKeyDebounced	Return keypad key with debounce
00B3	WaitKeyReleased	Wait for keypad key to be released

SBCMON Library Routine: SendSerial

Subroutine: SendSerial

Entry Point: 0041h

Purpose: Send a null-terminated string to the serial port

Inputs Registers: None

Output Registers: None

Registers Destroyed: DPTR

The SendSerial subroutine will send to serial port the null-terminated string that immediately follows the LCALL SendSerial instruction that called it. It will then return to the instruction following the terminating null value of the message.

```
LCALL SendSerial
DB "Hello, World!", 13, 0
RET
```

The first instruction calls the SendSerial subroutine which will then send the contents of the string that follows ("Hello, World!') to the serial port. When the subroutine reaches the null character (”the zero byte”) it will return to the instruction immediately following; RET in this case.

NOTE: It is absolutely imperative that the string be terminated with the null character (0). This is the marker that tells the SendSerial subroutine to stop sending data to the serial port. If the null character is missing the routine will continue to send data to the serial port until it reaches some 00 character later in memory. This usually results in garbage characters being sent to the serial port and the program behaving in an unexpected manner.

This subroutine is very useful for writing messages to the serial port. Since the text immediately follows the LCALL, the text being written to the serial port is embedded with the related code.

SBCMON Library Routine: SendSerialHexByte

Subroutine:	SendSerialHexByte
Entry Point:	0044
Purpose:	Sends the value in the accumulator to the serial port as two hexadecimal digits.
Inputs Registers:	Accumulator: Value to send as two nibbles
Output Registers:	None
Registers Destroyed:	None

The SendSerialHexByte subroutine takes the value in the accumulator, converts it to two hexadecimal digits in ASCII format, and sends those two digits to the serial port. This is useful for displaying the contents of a register in a human-readable format.
For example:

```
MOV A,#41h
LCALL SendSerialHexByte
```

The above code will send the digit '4' followed by the digit '1' to the serial port. This differs from just sending the accumulator (41h) to the serial port which would result in the letter 'A' being sent since 41h is the ASCII value for 'A'.

SBCMON Library Routine: SerialSendByte

Subroutine:	SerialSendByte
Entry Point:	0047
Purpose:	Sends the value in the accumulator to the serial port
Inputs Registers:	Accumulator: Character to send to the serial port
Output Registers:	None
Registers Destroyed:	None

The SendSerialByte subroutine simply sends the character that is currently in the accumulator to the serial port and waits for it to be completely sent before returning.

```
MOV A,#41h
LCALL SendSerialByte
```

This code will send the letter 'A' to the serial port since 41h is the ASCII code for the letter 'A'.

SBCMON Library Routine: GetSerialByte

Subroutine:	GetSerialByte
Entry Point:	004A
Purpose:	Waits for a character to be received on the serial port, then returns it
Inputs Registers:	None
Output Registers:	Accumulator: The character that was received by the serial port
Registers Destroyed:	None

The GetSerialByte subroutine waits indefinitely for a character to be received by the serial port. When the character is received, the subroutine reads the value and returns it in the accumulator.

LCALL GetSerialByte

This instruction will wait for a character to be received by the serial port. If the user presses the 'A' key on the terminal's keyboard then this routine will return 41h, the ASCII code for 'A', in the accumulator.

SBCMON Library Routine: GetSerialLine

Subroutine:	GetSerialLine
Entry Point:	004D
Purpose:	Allows the user to enter a full line, waits for user to press RETURN
Inputs Registers:	None
Output Registers:	Accumulator: First character of the line entered by the user R0: Address of the first byte of internal RAM used for the buffer
Registers Destroyed:	PSW

The GetSerialLine subroutine allows the user to enter a full line of ASCII text into a buffer in internal RAM and terminates the string with a null character (00h) when the user presses the ENTER key. The subroutine correctly handles backspaces and ensures that the user does not enter more characters than there is room in the buffer.

The characters entered by the user are stored in internal RAM. When the subroutine returns the accumulator will hold the first character of the line entered by the user; this is useful for quickly branching based on the first character of the line. The subroutine returns the internal RAM address of the first byte of the buffer in R0. The buffer may be assumed to continue until a null character is reached.

Assume that GetSerialLine is called and the user enters the following line:

> **TEST123**

When the subroutine returns the accumulator would hold the value 54h—which is the ASCII value for 'T'. To read the buffer the program would look at the value of R0 returned by the subroutine. If R0 is returned as B0h, that would indicate that:
IRAM[B0] = 54h (T)
IRAM[B1] = 45h (E)
IRAM[B2] = 53h (S)
IRAM[B3] = 54h (T)
IRAM[B4] = 31h (1)
IRAM[B5] = 32h (2)
IRAM[B6] = 33h (3)
IRAM[B7] = 00h (Null inserted at end of line)

> **NOTE:** Be sure to use the value of R0 returned by GetSerialLine to determine where the line buffer is in internal RAM. While it will always be the same as long as the program is run on the same version of SBCMON, if SBCMON is ever upgraded to a new version it is entirely possible that the buffer will move somewhere else. If R0 is used to determine the location of the buffer then future upgrades to SBCMON will have no negative impact on the program.

SBCMON Library Routine: SendLCDText

Subroutine:	SendLCDText
Entry Point:	0050
Purpose:	Sends the character in the accumulator to the LCD as a text character for display
Inputs Registers:	Accumulator: Text character to send to the LCD
Output Registers:	None
Registers Destroyed:	None

The SendLCDText subroutine sends the ASCII character held in the accumulator to the LCD as a text character. Text characters sent to the LCD are normally those that will be displayed on the LCD itself.

```
MOV A,#'H'
LCALL SendLCDText
MOV A,#'E'
LCALL SendLCDText
MOV A,#'L'
LCALL SendLCDText
MOV A,#'L'
LCALL SendLCDText
MOV A,#'O'
LCALL SendLCDText
```

The code above sends the word HELLO to the LCD's display, assuming the LCD has previously been properly initialized.

SBCMON Library Routine: SendLCDCommand

Subroutine:	SendLCDCommand
Entry Point:	0053
Purpose:	Sends the value in the accumulator to the LCD as an LCD command
Inputs Registers:	Accumulator: Value to send to the LCD as a command
Output Registers:	None
Registers Destroyed:	None

The SendLCDCommand subroutine sends the value contained in the accumulator to the LCD as an LCD command. LCD commands perform special functions such as initializing the LCD, clearing the screen, or positioning the cursor.

The following instructions will initialize the LCD and clear the display screen by sending the appropriate commands to the LCD.

```
MOV A,#38h
LCALL SendLCDCommand
MOV A,#0Eh
LCALL SendLCDCommand
MOV A,#06h
LCALL SendLCDCommand
MOV A,#01h
LCALL SendLCDCommand
```

SBCMON Library Routine: I2C_ReadByte

Subroutine:	I2C_ReadByte
Entry Point:	0056
Purpose:	Reads one byte from the I2C bus
Inputs Registers:	Carry: 0=Send ACK, 1=Send NAK
Output Registers:	Accumulator: Byte received from I2C bus
Registers Destroyed:	None

The I2C_ReadByte subroutine reads one byte from the I2C bus and returns its value in the accumulator. If the carry bit is clear then the subroutine sends an ACK, otherwise it sends a NAK.

SBCMON Library Routine: I2C_SendByte

Subroutine:	I2C_SendByte
Entry Point:	0059
Purpose:	Sends the byte in the accumulator to the I2C bus
Inputs Registers:	Accumulator: Byte to send to the I2C bus
Output Registers:	Carry: 0=No error, 1=Error-no ACK received
Registers Destroyed:	None

The I2C_SendByte subroutine sends the byte in the accumulator to the I2C bus. When it returns the carry bit indicates whether the byte was sent successfully or not. If the carry bit is clear, the send was successful. If the carry bit is set then no ACK was received from the receiving I2C device.

SBCMON Library Routine: I2C_Reset

Subroutine:	I2C_Reset
Entry Point:	005C
Purpose:	Resets the I2C bus
Inputs Registers:	None
Output Registers:	Carry: 0=No error, 1=Error, couldn't reset bus
Registers Destroyed:	None

The I2C_Reset subroutine sends the I2C "stop" condition nine times to ensure that every device has aborted any ongoing transfers and released the I2C bus. The routine returns the carry cleared if the reset was successful and sets the carry if it appears that some device hasn't released the I2C bus.

SBCMON Library Routine: I2C_Start

Subroutine:	I2C_Start
Entry Point:	005F
Purpose:	Starts an I2C conversation
Inputs Registers:	None
Output Registers:	None
Registers Destroyed:	None

The I2C_Start subroutine begins an I2C conversation by bringing the SDA line low while SCL is high. This should be executed before sending bytes to any I2C device.

SBCMON Library Routine: I2C_Stop

Subroutine:	I2C_Stop
Entry Point:	0062
Purpose:	Terminates an I2C conversation
Inputs Registers:	None
Output Registers:	None
Registers Destroyed:	None

The I2C_Stop subroutine ends an I2C conversation by bringing the SDA line high while SCL is high. This should be executed when I2C communication is complete.

SBCMON Library Routine: SPI_ReadByte

Subroutine:	SPI_ReadByte
Entry Point:	0065
Purpose:	Reads a single byte from the SPI bus into the accumulator
Inputs Registers:	None
Output Registers:	Accumulator: Byte received from the SPI bus
Registers Destroyed:	None

The SPI_ReadByte subroutine reads a single byte from the SPI bus and returns it in the accumulator.

SBCMON Library Routine: SPI_SendByte

Subroutine:	SPI_SendByte
Entry Point:	0068
Purpose:	Sends the byte in the accumulator to the SPI bus
Inputs Registers:	Accumulator: Byte to send to the SPI bus
Output Registers:	None
Registers Destroyed:	None

The SPI_SendByte subroutine sends the byte in the accumulator to the SPI bus.

SBCMON Library Routine: ByteTo2Ascii

Subroutine:	ByteTo2Ascii
Entry Point:	006B
Purpose:	Converts the value in the accumulator to two ASCII digits
Inputs Registers:	Accumulator: Byte to convert to ASCII digits
Output Registers:	Accumulator: High nibble of ASCII pair (30h-39h, 40h-46h) R0: Low nibble of ASCII pair (30h-39h, 40h-46h)
Registers Destroyed:	None

The ByteTo2Ascii subroutine converts the value in the accumulator to two ASCII values that represent the two digits of the original value. This is useful in converting a numeric value in the accumulator to two ASCII values which can be displayed on the LCD or sent to the serial port.

For example:

```
MOV A,#41h
LCALL SendSerialByte     ;Sends letter 'A' to serial port
MOV A,#41h
LCALL ByteTo2Ascii       ;Convert 41h to component ASCII digits
LCALL SendSerialByte     ;Sends 34h (digit '4') to the serial port
MOV A,R0
LCALL SendSerialByte     ;Sends 31h (digit '1') to the serial port
```

The above code illustrates the difference between sending the value 41h to the serial port—which will display the letter 'A' since 41h is its ASCII code—and converting it to two ASCII characters using the ByteToBCD routine. Upon returning from ByteToBCD the accumulator will hold 34h—which is the ASCII value for '4'—and R0 will hold 31h—which is the ASCII value for '1'.

SBCMON Library Routine: HexToNibble

Subroutine:	HexToNibble
Entry Point:	006E
Purpose:	Convert ASCII value in accumulator to a value between 0 and 15 (00h and 0Fh)
Inputs Registers:	Accumulator: Hexadecimal ASCII character to convert to a value
Output Registers:	Accumulator: Converted value between 0 and 15 (00h and 0Fh) Carry: 0=No error, 1=ASCII value wasn't hexadecimal digit
Registers Destroyed:	None

The HexToByte subroutine converts a ASCII character representing a hexadecimal digit to a value between 0 and 15 (00h and 0Fh). This is useful when receiving hexadecimal data from the user and needing to convert it to a numeric value for subsequent processing.

Original Hex Digit	Original ASCII Value	HexToNibble Result
'0'	30h	00h
'1'	31h	01h
'2'	32h	02h
'3'	33h	03h
'4'	34h	04h
'5'	35h	05h
'6'	36h	06h
'7'	37h	07h
'8'	38h	08h
'9'	39h	09h
'A', 'a'	41h, 61h	0Ah
'B', 'b'	42h, 62h	0Bh
'C', 'c'	43h, 63h	0Ch
'D', 'd'	44h, 64h	0Dh
'E', 'e'	45h, 65h	0Eh
'F', 'f'	46h, 66h	0Fh

The following code converts the ASCII value 'A' (41h) to 0Ah:

```
MOV A,#'A'
LCALL HexToByte
```

SBCMON Library Routine: GetHexValue

Subroutine:	GetHexValue
Entry Point:	0071
Purpose:	Converts hexadecimal digits in input buffer to numeric values
Inputs Registers:	R0: Points to first internal RAM address of hexadecimal digits R1: Points to first internal RAM address to store results
Output Registers:	Accumulator: The first non-hex character that terminated conversion R0: Points to the internal RAM address after the non-hex character Carry: 0=Successful conversion, 1=Nothing was converted
Registers Destroyed:	None

The GetHexValue subroutine converts between one and four hexadecimal ASCII digits contained within a buffer in internal RAM to their 16-bit numeric value. It is normally used to convert hexadecimal values that were received in a character buffer using the GetSerialLine subroutine.

Before calling GetHexValue, the program must initialize R0 and R1. The R0 register must be set to point to where the first hexadecimal character to be decoded is in internal RAM. R1 must be set to point to where in internal RAM the decoded value will be placed. The decoded value is a 16-bit value which requires two bytes: The high byte will be stored at the address pointed to by R1 and the low byte will be stored in the byte immediately following that address. The subroutine will then scan the buffer and convert the hexadecimal values to the 16-bit value until it reaches a character that is not a valid hexadecimal digit. It will then load the accumulator with that character and set R0 to the address following that non-hexadecimal character.

For example:

```
LCALL GetSerialLine     ;Get line, R0 returns pointing to buffer
MOV R1,#50h             ;Set address for decoded 16-bit value
LCALL GetHexValue       ;Convert hex to 16-bit value in 50h & 51h
```

This code will first get a line of input from the serial port; for the sake of example, the user enters the ASCII string "681FXRS" and GetSerialLine returns with R0 set to B0h as the address used by SBCMON for the serial input buffer. The second line sets the decode address to 50h so that the high byte of the decoded value will be stored at internal RAM address 50h and the low byte of the decoded value will be stored at internal RAM address 51h. The program then calls GetHexValue.

Upon return, internal RAM address 50h will contain the value 68h (the high byte of the hexadecimal value entered by the user), address 51h will contain 1Fh (the low byte of the hexadecimal value entered by the user), the accumulator will contain the character 'X' (58h), and R0 will contain B5h which is the address of the character in the buffer following the letter that terminated the hexadecimal decoding.

The fact that R0 is always advanced by GetHexValue to point to the character after the non-hexadecimal character means that it can be called repeatedly to decode multiple hexadecimal values contained within a single line. For example, if the line entered by the user was "123A B981 C912" the following lines could decode it:

```
LCALL GetSerialLine     ;Get line, R0 returns pointing to buffer
MOV R1,#50h             ;Set address to decode 1st 16-bit value
LCALL GetHexValue       ;Convert 123A, store in 50h and 51h
MOV R1,#52h             ;Set address to decode 2nd 16-bit value
```

```
LCALL GetHexValue          ;Convert B981, store in 52h and 53h
MOV R1,#54h                ;Set address to decode 3rd 16-bit value
LCALL GetHexValue          ;Convert C912, store in 54h and 55h
```

SBCMON Library Routine: GetSerialHex

Subroutine:	GetSerialHex
Entry Point:	0074
Purpose:	Reads 2 hex digits from serial port, converts to value in accumulator
Inputs Registers:	None
Output Registers:	Accumulator: Converted value of the 2 hex digits received Carry: 0=No error, 1=Error, characters weren't hexadecimal digits
Registers Destroyed:	R1

The GetSerialHex subroutine waits for two characters on the serial port and treats them as hexadecimal digits. It converts each character to the numeric value it represents and combines them to form an 8-bit value that is returned in the accumulator. For example, if this subroutine is called and the serial report receives the character 'F' followed by the character '8', this routine will return the value F8h in the accumulator.

SBCMON Library Routine: ToUpper

Subroutine:	ToUpper
Entry Point:	0077
Purpose:	Converts the character in the accumulator to upper-case
Inputs Registers:	Accumulator: Character to be made upper-case
Output Registers:	Accumulator: Character to be made upper-case
Registers Destroyed:	None

This routine does nothing more than make a lower-case character into an upper-case character. If the character in the accumulator is not a letter or is already upper-case then the value is left unchanged.

SBCMON Library Routine: SendDecimalR4567

Subroutine:	SendDecimalR4567
Entry Point:	007A
Purpose:	Sends the value in R4-R7 to the serial port as a decimal value
Inputs Registers:	R4 (high) – R7 (low): Value to be sent to serial port as decimal value
Output Registers:	None
Registers Destroyed:	R0, R1, Accumulator

This routine takes the 32-bit value contained in registers R4 (high) through R7 (low) and sends it to the serial port as a decimal value. For example, if R4=01h, R5=25h, R6=30h, and R7=15h, that represents the value 1,9214,357. Thus the SendDecimalR4567 routine would send the string "19214357" to the serial port.

This function is useful for displaying decimal values from within SBCMON programs.

SBCMON Library Routine: DivideBy10

Subroutine:	DivideBy10
Entry Point:	007D
Purpose:	Divides the value pointed to by @R0 through @R0+3 by 10
Inputs Registers:	R0: Points to high byte of value to be divided by 10, remaining three bytes of value are contained in next three bytes of memory.
Output Registers:	Accumulator: Remainder of the division by 10.
Registers Destroyed:	None

This routine takes the 32-bit value that is contained in the internal RAM addresses pointed to by R0 (and the subsequent three bytes of memory) and divides it by 10 (decimal). The result is left in the same four bytes of memory and the remainder is left in the accumulator.

While the DIV AB instruction can be used to divide values less than 256 by 10, this routine is useful to divide large 32-bit values by 10.

SBCMON Library Routine: Check4BytesForZero

Subroutine:	Check4BytesForZero
Entry Point:	0080
Purpose:	Checks that value pointed to by @R0 through @R0+3 is zero.
Inputs Registers:	R0: Points to high byte of value to be checked for zero, remaining three bytes of value are contained in next three bytes of memory.
Output Registers:	Accumulator: 0=All four bytes are zero, Non-0: 1 or more bytes are non-zero
Registers Destroyed:	None

This routine analyzes the 32-bit value that is contained in the internal RAM addresses pointed to by R0 (and the subsequent three bytes of memory) and determines if all four bytes are zero. If all four bytes are zero, the routine returns zero in the accumulator. Otherwise, the accumulator is non-zero.

SBCMON Library Routine: ShiftR765Left

Subroutine:	ShiftR765Left
Entry Point:	0083
Purpose:	Checks that value pointed to by @R0 through @R0+3 is zero.
Inputs Registers:	Shifts the value in R5 (high) through R7 (low) left one bit
Output Registers:	R5 (high) – R7 (low): Shifted value
Registers Destroyed:	Accumulator

This routine takes the value contained in R5 (high byte) through R7 (low byte) and shifts the entire value left by one bit, propagating through the three registers as necessary. This is effectively a RL A instruction that acts on R5 through R7 rather than the accumulator.

SBCMON Library Routine: AddR67ToR23

Subroutine:	AddR67ToR23
Entry Point:	0086
Purpose:	Adds R6 (high) and R7 (low) to R2 (high) and R3 (low)
Inputs Registers:	R6 (high), R7 (low): 16-bit value to add
Output Registers:	R2 (high), R3 (low): 16-bit value to be added to, and result
Registers Destroyed:	Accumulator

This routine takes the 16-bit value contained in R6 (high byte) and R7 (low byte) and adds it to the 16-bit value contained in R2 (high byte) and R3 (low byte). The result is stored in R2 and R3.

SBCMON Library Routine: AddR4567ToR0123

Subroutine:	AddR4567ToR0123
Entry Point:	0089
Purpose:	Adds R4 (high) through R7 (low) to R0 (high) through R3 (low)
Inputs Registers:	R4 (high) - R7 (low): 32-bit value to add
Output Registers:	R0 (high) - R3 (low): 32-bit value to be added to, and result
Registers Destroyed:	Accumulator

This routine takes the 32-bit value contained in R4 (high byte) through R7 (low byte) and adds it to the 32-bit value contained in R0 (high byte) through R3 (low byte). The result is stored in R0 through R3.

SBCMON Library Routine: InitializeLCD

Subroutine:	InitializeLCD
Entry Point:	008C
Purpose:	Initializes and configures the LCD
Inputs Registers:	Bit LCDDirect4: 0=8-bit mode, 1=4-bit mode Bit LCDMMMode: 1=Direct mode, 0=Memory-mapped mode
Output Registers:	None
Registers Destroyed:	Accumulator

This routine initializes the LCD by sending it one of the following sequence of commands: 38h, 0Eh, 06h (for 8-bit mode) or 28h, 0Eh, 06h (for 4-bit mode). The commands are automatically sent to the right bus (direct or memory-mapped) depending on the setting of the LCDMMMode bit.

SBCMON Library Routine: ReadLCDStatus

Subroutine:	ReadLCDStatus
Entry Point:	008F
Purpose:	Read LCD Status
Inputs Registers:	None
Output Registers:	Accumulator: Returned status of the LCD
Registers Destroyed:	None

This routine waits for the LCD to return its status register. This can be used for waiting for the LCD to become not busy and, upon becoming not busy, to return the status register which includes the LCD cursor position.

SBCMON Library Routine: ReadLCDText

Subroutine:	ReadLCDText
Entry Point:	0092
Purpose:	Read LCD text
Inputs Registers:	None
Output Registers:	Accumulator: Returned text from the LCD
Registers Destroyed:	None

This routine reads the value contained in the LCD at the current cursor position and returns it in the accumulator. This is useful for reading what the LCD is currently displaying which, in turn, is useful in many scrolling applications.

SBCMON Library Routine: SetDPTRmmKey1

Subroutine:	SetDPTRmmKey1
Entry Point:	0095
Purpose:	Set DPTR to address of memory-mapped keypad row #1
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of keypad row #1
Registers	None

This routine sets DPTR to the memory-mapped address of keypad row #1. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for keypad row #1 can always be obtained by calling this routine.

SBCMON Library Routine: SetDPTRmmKey2

Subroutine:	SetDPTRmmKey2
Entry Point:	0098
Purpose:	Set DPTR to address of memory-mapped keypad row #2
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of keypad row #2
Registers	None

This routine sets DPTR to the memory-mapped address of keypad row #2. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for keypad row #2 can always be obtained by calling this routine.

SBCMON Library Routine: SetDPTRmmKey3

Subroutine:	SetDPTRmmKey3
Entry Point:	009B
Purpose:	Set DPTR to address of memory-mapped keypad row #3
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of keypad row #3
Registers	None

This routine sets DPTR to the memory-mapped address of keypad row #3. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for keypad row #3 can always be obtained by calling this routine.

SBCMON Library Routine: SetDPTRmmKey4

Subroutine:	SetDPTRmmKey4
Entry Point:	009E
Purpose:	Set DPTR to address of memory-mapped keypad row #4
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of keypad row #4
Registers	None

This routine sets DPTR to the memory-mapped address of keypad row #4. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for keypad row #4 can always be obtained by calling this routine.

SBCMON Library Routine: SetDptrMMLcdWC

Subroutine:	SetDptrMMLcdWC
Entry Point:	00A1
Purpose:	Set DPTR to address of memory-mapped LCD for write command
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of LCD for write command
Registers Destroyed:	None

This routine sets DPTR to the memory-mapped address of the LCD for writing a command. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for the LCD can always be obtained by calling this routine.

SBCMON Library Routine: SetDptrMMLcdWT

Subroutine:	SetDptrMMLcdWT
Entry Point:	00A4
Purpose:	Set DPTR to address of memory-mapped LCD for write text
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of LCD for write text
Registers Destroyed:	None

This routine sets DPTR to the memory-mapped address of the LCD for writing text. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for the LCD can always be obtained by calling this routine.

SBCMON Library Routine: SetDptrMMLcdRC

Subroutine:	SetDptrMMLcdRC
Entry Point:	00A8
Purpose:	Set DPTR to address of memory-mapped LCD for reading status/command
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of LCD for read status
Registers Destroyed:	None

This routine sets DPTR to the memory-mapped address of the LCD for reading status/command. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for the LCD can always be obtained by calling this routine.

SBCMON Library Routine: SetDptrMMLcdRT

Subroutine:	SetDptrMMLcdRT
Entry Point:	00AA
Purpose:	Set DPTR to address of memory-mapped LCD for read text
Inputs Registers:	None
Output Registers:	DPTR: Set to the memory-mapped address of LCD for read text
Registers Destroyed:	None

This routine sets DPTR to the memory-mapped address of the LCD for reading text. By calling this routine instead of setting DPTR to a specific value, a program can operate under SBCMON secure in the knowledge that even if the underlying hardware that SBCMON runs on is changed, the correct address for the LCD can always be obtained by calling this routine.

SBCMON Library Routine: GetSBCMONVersion

Subroutine:	GetSBCMONVersion
Entry Point:	00AD
Purpose:	Returns the version of the SBCMON program
Inputs Registers:	None
Output Registers:	Accumulator: Major version and point relase B: Minor version
Registers Destroyed:	None

Gets the version of SBCMON and returns it in accumulator and B. Accumulator holds the major version number * 10 plus the minor version number. The B register holds the minor version number.

For example, if the SBCMON version is 1.3.25, the accumulator would return with the value 13h and the B register would return with the value 25 (19h).

SBCMON Library Routine: GetKeyDebounced

Subroutine:	GetKeyDebounced
Entry Point:	00B0
Purpose:	Reads one keypress from the keypad, debounces it, and returns it
Inputs Registers:	None
Output Registers:	Accumulator: The value of the key pressed
Registers Destroyed:	R3, R6, and R7

This routine waits for the user to press a key on the keypad. When a key is detected, it is properly debounced to protect against false readings. Once debounced, the value of the key is returned so that the calling program may process it. Once the key is processed by the main program, the main program should also call the WaitKeyReleased routine to verify that the key has been released so it doesn't automatically re-read the same keypress.

SBCMON Library Routine: WaitKeyReleased

Subroutine:	WaitKeyReleased
Entry Point:	00B3
Purpose:	Waits for the key of the keypad to be released
Inputs Registers:	None
Output Registers:	None
Registers Destroyed:	R0, Accumulator

This routine is called after the GetKeyDebounced key is used to obtain a keypress. Once the key is obtained and processed, the program should call this routine to make sure the key is released so that the program doesn't re-interpret the keypress as a new keypress.

Download 8052.com SBCMON Monitor Program

You may download both the source code or the ready-to-use Intel-Hex version of the monitor program. If you already have an 8052.com SBC you may download the HEX program directly into your SBC via ISP. Use the source ASM program if you'd like to see the code that makes SBCMON work.

Current Version: 1.3.2 - 28/Aug/2005
8052.com SBC Monitor Program Source (sbcmon13.asm)
8052.com SBC Monitor Program Intel HEX (sbcmon13.hex)

SBCMON Manual

Full instructions for SBCMON are available here.

What is a Monitor Program?

A "monitor program" is a program which is loaded into a system--in this case the 8052.com SBC--and which provides a certain number of features and control over the system. It provides a base with which the SBC can be tested and controlled without writing an entire program from scratch and, once the hardware is known to be working, can facilitate subsequent software development.
For example, when first developing code for a piece of hardware which has not been tested it is sometimes difficult to know for certain whether something isn't working because the software is broken or because something in the hardware is broken. If your program writes to the LCD and nothing appears on the LCD is it because your code is wrong or because one of the pins of the LCD was poorly soldered?
A monitor program solves that problem by providing a piece of software that is known to work which may be loaded onto the SBC. If the SBC's features can be successfully operated by the monitor program the user may proceed with the knowledge that the hardware is sound. Additionally, the monitor program may provide additional features which make software development easier and faster.

Features of the 8052.com SBC Monitor Program

The 8052.com SBC Monitor Program includes the following features:

1. **Test XRAM**. Perform a test of XRAM to verify correct functionality.
2. **LCD Access**. Read/write from the LCD either as memory-mapped device or by direct connection.
3. **DS1307 RTC Access**. Set and read the DS1307 Real Time Clock via I2C.
4. **AT25010A Access**. Write to and read from the AT25010A serial EEPROM via SPI.
5. **Load Hex Files**. Load Intel-standard HEX files into the Program RAM area from 2000h-3FFFh.
6. **Execute Code**. Execute code from anywhere in memory.
7. **Display XRAM**. Display the contents of any range of XRAM in hex format.
8. **Modify XRAM**. Modify the contents of any address of external RAM.
9. **Mini-Assembler**. Provides a functional mini-assembler that can take standard 8052 assembly language instructions and assemble them into machine code within Program RAM for subsequent execution.

Download Keypad Demo for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the keypad demonstration program. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes the keypad demo work.

Keypad Demo Program Source (keypad.asm)
Keypad Demo Intel HEX (keypad.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

This is a 4x4 keypad demonstration program which will read keypresses on the keypad and echo them to the serial port where they may be viewed with the PC terminal program. The code includes simple debounce logic to filter phantom keypresses that would otherwise be observed due to the mechanical bounce of the key itself.

The routine will exit and return to SBCMON's main menu when the user presses any key in the PC terminal program.

Concepts Demonstrated by Program

This program demonstrates the following concepts:

1. Reading a 4x4 keypad.
2. Memory-mapped device & memory-mapped I/O.
3. Key debounce.
4. Code look-up table.
5. Delay based on instruction loops (1/20th of a second).

Download LCD Echo Demo for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the keypad demonstration program. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes the LCD demo work.

LCD Echo Demo Program Source (lcdecho.asm)
LCD Echo Demo Intel HEX (lcdecho.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

This program receives input from the PC terminal program via the serial port and echos everything it receives to the LCD. Although in theory this sounds simple, the code implements some features that are not automatically handled by the LCD.

First, keep in mind that line 1 involves LCD memory addresses 00h through 0Fh while line 2 involves addresses 40h through 4Fh. Thus if a character is displayed at the last character position of line 1, the next character sent to the LCD will be not be displayed since it will be written to address 10h which is not visible. Thus a "wrap" feature must be implemented that makes sure that when the cursor goes off the end of line 1 that it is automatically positioned at the beginning of line 2.

Further, if the user types beyond the visible area of line 2 then the contents of the LCD must be scrolled. This is accomplished by copying the contents of line 2 to line 1, blanking out line 2, and positioning the cursor at the beginning of line 2.

The carriage return character is also a special case. If the user is typing on line 1 then the carriage return simply positions the cursor at the beginning of line 2. If the user is on line 2 and press a carriage return, however, then the program executes the scrolling code that was just described--effectively causing the LCD to scroll and leaving the cursor at the beginning of line 2.

The backspace character is handled by reading the current cursor position, decrementing it by one, and erasing the character found at that position and then leaving the cursor at that position. The code will ignore the backspace character if the cursor is already in the upper-left hand corner. Also, if the cursor is at the first character of line 2 then the backspace character must go to the last position of line 1--if it were not for this check then a backspace from line 2 character 1 (address 40h) would go to address 3Fh which is not visible on line 1. Instead, 40h must back up to position 0Fh.

All of these aspects are handled by the demo program. Thus you will find that typing into the PC terminal program will cause text to be displayed on the LCD in a very natural and logical manner.

NOTE: Before running this program, you must first use the M0, M1, or M2 command in SBCMON to configure how the LCD is connected to the SBC (direct-connect, memory-mapped, etc.)

Concepts Demonstrated by Program

This program demonstrates the following concepts:

1. Receiving characters from serial port.
2. Initializing LCD.
3. Clearing LCD screen.
4. Setting LCD cursor position.
5. Reading LCD cursor position.
6. Reading contents of LCD at cursor position.
7. LCD vertical text scrolling.
8. Handling backspace on multi-line display.

Download Digital Clock for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the keypad demonstration program. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes SBCMON work.

SBCMON Digital Clock Program Source (clock.asm)
SBCMON Digital Clock Intel HEX (clock.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

This program is a simple digital clock that constantly displays the current time and date retrieved from the DS1307 RTC and displays it on the LCD. A typical display for the digital clock is:

Fri Feb 25, 2005
10:31:47

Note that the clock must first be set (with the 'C' command, for example) prior to running this program or the clock will be wrong and/or will not be updated. When setting the clock, the day of the week code (1-7) should be 1=Monday, 2=Tuesday, etc.
You may press any key in the PC's terminal program in order to exit the clock.

Concepts Demonstrated by Program

This program demonstrates the following concepts:

1. I^2C communication.
2. Reading the time from the DS1307 RTC.
3. Clearing LCD screen.
4. Setting LCD cursor position.
5. Writing characters to the LCD.

Download Software-Based Digital Clock for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the keypad demonstration program. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes SBCMON work.

SBCMON Software-Based Digital Clock Program Source (sftclock.asm)
SBCMON Software-Based Digital Clock Intel HEX (sftclock.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

This program is a simple digital clock that constantly displays the current time and date using an interrupt-driven time-keeping system that is based on the concept explained in this tutorial. This program will keep accurate count of the current time using the timer 0 interrupt and display the time (hour:minute:second) to the LCD each second.

The clock may be set at any time by pressing the asterisk ("*") key on the 4x4 keypad. The program will then prompt for the 2-digit hour. The hour should be entered and the asterisk key pressed as an "enter" key. The hour will be validated to verify it is a valid number between 00 and 23. The application will then prompt the user to enter the minute and second, both of which will be validated for being a number between 00 and 59. Once the second is entered, the current time will be set to the requested hour:minute:second. The clock program may be exited by pressing the "A" button on the 4x4 keypad when the program is displaying the clock.

This program is capable of keeping accurate time with a precision that is limited only by the oscillator connected to the microcontroller. Since such oscillators are usually accurate within +/-100ppm, the maximum deviation of this clock should be +/- 8.6 seconds per day. In reality, oscillators are usually more accurate than their maximum specified deviation and testing of this application on a typical 8052.com SBC has yielded no discernible error over a period of 24 hours.

Concepts Demonstrated by Program

This program demonstrates the following concepts:

1. Interrupts.
2. Timer 0 as a time-keeping mechanism.
3. Clearing LCD screen.
4. Setting LCD cursor position.
5. Writing characters to the LCD.
6. Reading and debouncing the 4x4 keypad.

Download PS/2 Communication Monitor for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the PS/2 communication monitor. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes the program work.

SBCMON PS/2 Communication Monitor Program Source (ps2.asm)
SBCMON PS/2 Communication Monitor Program Intel HEX (ps2.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

This program allows a standard PS/2-style keyboard or mouse to be connected to the SBC. All data that is returned by the keyboard or mouse is displayed to the terminal in the form of hexadecimal values. Absolutely no interpretation or modification to the data is made by the program so the data that is displayed is the "raw" data as returned by the keyboard or mouse.

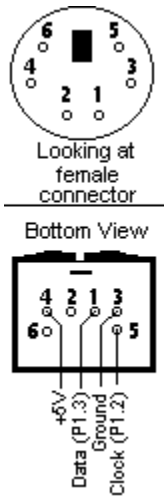
You may press any key in the PC's terminal program in order to exit the program and return to SBCMON.

Concepts Demonstrated by Program

This program demonstrates the following concepts:

Communicating with keyboard or mouse using clock and bidirectional data line.
Utilizing the PS/2 communication protocol.

Additional Required Hardware



This program requires a PS/2 keyboard or mouse to be connected to the SBC. The keyboard requires a +5V power supply, ground, and the program assumes that the keyboard data line is attached to P1.3 and the clock line is attached to P1.2.

I would recommend using an external board to mount a DIN-6 keyboard connector (Digikey part #275-1043-ND). On this board,

mount two 8-pin SIP headers. The first SIP header will allow you to attach a ribbon cable from port 1 (J7 on the SBC) and the second SIP header will allow you to attach a ribbon cable from the EX connector (J11). The J7 connector will expose P1.2 and P1.3 while the J11 connector will provide +5V and ground.

Connecting the PS/2 Connector

The PS/2 connector is a female DIN-6 connector. The pinout is pictured to the right. The top graphic shows the pinout when looking at the PS/2 female connector. The bottom portion of the graphic shows the normal pinout of the pins that are connected at a right angle to the PCB. The pins use 0.1" spacing so are very appropriate for a thru-hole circuit board.

Theory of Operation

This program operates on the basis that both the PS/2 keyboard and the PS/2 mouse use the same communication protocol--the only difference is the format of the data returned by the device.

Additionally, the absolute minimum initialization code necessary to make both the keyboard and the mouse work is exactly the same: The 0xFF initialization command must be sent followed by the 0xF4 command which enables reporting from the device. When these two commands are sent to either a PS/2 keyboard or mouse, the device will immediately start returning data via the PS/2 protocol.

As a result, this program simply sends this absolutely minimal initialization sequence and then displays all data that is returned. The program need not know or care whether the device is a keyboard or a mouse since it is only interested in displaying the raw data.

Credits

This example program was based on Gabriel Lour's "keyboard routine" code which was contributed to the 8052.com code library. Modifications were made to his code to make the example program more closely conform to the style of the other SBC example programs. Additionally, the routines were modified so as not to require any specific addresses in internal RAM. Rather, "R" registers are used for temporary variable storage.

Additionally, Gabriel's code deals with the keyboard's scan codes. This example program, on the other hand, takes the 1-3 byte keyboard scan codes and converts them to a a single-byte unique scan code. This makes it much easier to deal with the resulting data in an assembly language program.

Information on the PS/2 connector pin-out and the keyboard scan codes returned by the keyboard were obtained from Adam Chapweske's site.

Download PS/2 Keyboard for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the keyboard demonstration program. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes SBCMON work.

SBCMON PS/2 Keyboard Program Source (ps2key.asm)
SBCMON PS/2 Keyboard Intel HEX (ps2key.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

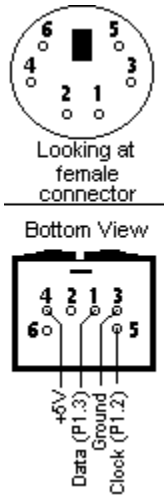
This program allows a standard PS/2-style keyboard to be connected to the SBC. Keypresses on the keyboard are detected by the program, the keyboard scan code of 1-3 bytes is converted to a single-byte scancode, and the byte is displayed on the terminal. You may press any key in the PC's terminal program in order to exit the program and return to SBCMON.

Concepts Demonstrated by Program

This program demonstrates the following concepts:

Communicating with keyboard using clock and bidirectional data line.
Utilizing the keyboard's communication protocol.
Translating a multi-byte keyboard sequence to a single byte

Additional Required Hardware



This program requires a PS/2 keyboard to be connected to the SBC. The keyboard requires a +5V power supply, ground, and the program assumes that the keyboard data line is attached to P1.3 and the clock line is attached to P1.2.

I would recommend using an external board to mount a DIN-6 keyboard connector (Digikey part #275-1043-ND). On this board, mount two 8-pin SIP headers. The first SIP header will allow you to attach a ribbon cable from port 1 (J7 on the SBC) and the second SIP header will allow you to attach a ribbon cable from the EX connector (J11). The J7 connector will expose P1.2 and P1.3 while the J11 connector will provide +5V and ground.

Connecting the PS/2 Connector

The PS/2 connector is a female DIN-6 connector. The pinout is pictured to the right. The top graphic shows the pinout when looking at the PS/2 female connector. The bottom portion of the graphic shows the normal pinout of the pins that are connected at a right angle to the PCB. The pins use 0.1" spacing so are very appropriate for a thru-hole circuit board.

Theory of Operation

The routines used in this example program can best be summarized as follows:

PS2_Init: Initializes the keyboard, sets the repeat rate to 30 characters/second, and turns off all keyboard LEDs.

PS2_SetLeds: Turns the three keyboard LEDs on or off. Bits 0, 1, and 2 control the LEDs accordingly.

PS2_GetByte: This reads a single byte from the keyboard. If the keyboard has no data pending, it clears the carry which indicates no data was read. If the keyboard has data pending, it is returned in the accumulator and the carry bit is set. Note that this will return one *byte* and that a single keypress may generate multiple bytes.

PS2_GetScanCode: Like PS2_GetByte, this routine will clear the carry if there is no data waiting and set the carry flag if data was returned by the keyboard. However, while PS2_GetByte will return a single byte (which may not be all the data required to process a keypress), PS2_GetScanCode will read 1, 2, or 3 bytes of data from the keyboard as necessary to determine the full keypress. It then translates the data received into a *single* byte to represent that keypress. For example, pressing the 'A' key will return the character 'A', pressing the 'C' key will return the character 'C'. Non-printable characters (escape, tab, function keys, etc.) are assigned a unique value--the values themselves can be viewed in the source code represented by the "KS_" equates. For example, KS_RIGHTCTRL is equated to 'b' so pressing the right control key would return the character 'b'.

If these routines were used in a more significant, functional program, the program would be responsible for recognizing when the caps key is pressed, caps lock is pressed, etc. These routines simply return a "raw" scan code representing the key that is actually pressed--it does not attempt to process these keys or interpret them.

Credits

This example program was based on Gabriel Lour's "keyboard routine" code which was contributed to the 8052.com code library. Modifications were made to his code to make the example program more closely conform to the style of the other SBC example programs. Additionally, the routines were modified so as not to require any specific addresses in internal RAM. Rather, "R" registers are used for temporary variable storage.

Additionally, Gabriel's code deals with the keyboard's scan codes. This example program, on the other hand, takes the 1-3 byte keyboard scan codes and converts them to a a single-byte unique scan code. This makes it much easier to deal with the resulting data in an assembly language program.

Information on the PS/2 connector pin-out and the keyboard scan codes returned by the keyboard were obtained from Adam Chapweske's site.

Download PS/2 Mouse for SBCMON

You may download either the source code or the ready-to-use Intel-Hex version of the mouse demonstration program. The Intel-Hex version may be loaded into SBCMON's memory using the **L** or **Q** commands. Download the source ASM program if you'd like to see the code that makes SBCMON work.

SBCMON PS/2 Mouse Program Source (ps2mouse.asm)
SBCMON PS/2 Mouse Intel HEX (ps2mouse.hex)

NOTE: This demo program requires SBCMON monitor program to run. Be sure to download SBCMON first, install on your SBC, and then run this demo from within SBCMON itself. This program may require additional modification in order to run without SBCMON.

What Does this Program Do?

This program allows a standard PS/2-style mouse to be connected to the SBC. Movements of the mouse and mouse clicks are detected by the program and the information is displayed to the terminal.

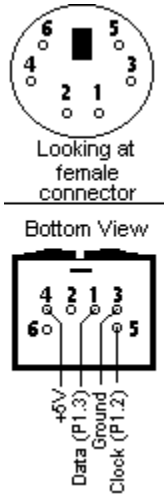
You may press any key in the PC's terminal program in order to exit the program and return to SBCMON.

Concepts Demonstrated by Program

This program demonstrates the following concepts:

Communicating with mouse using clock and bidirectional data line.
Utilizing the mouse's communication protocol.

Additional Required Hardware



This program requires a PS/2 keyboard or mouse to be connected to the SBC. The keyboard requires a +5V power supply, ground, and the program assumes that the keyboard data line is attached to P1.3 and the clock line is attached to P1.2.

I would recommend using an external board to mount a DIN-6 keyboard connector (Digikey part #275-1043-ND). On this board, mount two 8-pin SIP headers. The first SIP header will allow you to attach a ribbon cable from port 1 (J7 on the SBC) and the second

SIP header will allow you to attach a ribbon cable from the EX connector (J11). The J7 connector will expose P1.2 and P1.3 while the J11 connector will provide +5V and ground.

Connecting the PS/2 Connector

The PS/2 connector is a female DIN-6 connector. The pinout is pictured to the right. The top graphic shows the pinout when looking at the PS/2 female connector. The bottom portion of the graphic shows the normal pinout of the pins that are connected at a right angle to the PCB. The pins use 0.1" spacing so are very appropriate for a thru-hole circuit board.

Theory of Operation

The routines used in this example program can best be summarized as follows:

PS2_MouseInit: Initializes the mouse.

PS2_GetByte: This reads a single byte from the mouse. If the mouse has no data pending, it clears the carry which indicates no data was read. If the mouse has data pending, it is returned in the accumulator and the carry bit is set. Note that this will return one *byte* and that a single mouse movement generates *three* bytes of data.

PS2_GetMouse: Like PS2_GetByte, this routine will clear the carry if there is no data waiting and set the carry flag if data was returned by the mouse. However, while PS2_GetByte will return a single byte (which is not all the data required to process a single mouse movement), PS2_GetMouse will read three bytes of data as necessary to obtain a full data packet from the mouse.

If these routines were used in a more significant, functional program, the program would be responsible for keeping track of the mouse pointer position on the screen, adjusting it positively and negatively as the mouse is moved, and keeping track of the button positions of the mouse buttons.

Credits

This example program was based on Gabriel Lour's "keyboard routine" code which was contributed to the 8052.com code library. Modifications were made to his code to make the example program more closely conform to the style of the other SBC example programs. Additionally, the routines were modified so as not to require any specific addresses in internal RAM. Rather, "R" registers are used for temporary variable storage.

Additionally, Gabriel's code deals with the keyboard's scan codes. This example program, on the other hand, takes the 1-3 byte keyboard scan codes and converts them to a a single-byte unique scan code. This makes it much easier to deal with the resulting data in an assembly language program.

Information on the PS/2 connector pin-out and the keyboard scan codes returned by the keyboard were obtained from Adam Chapweske's site.

Product & Vendor Information 8052.com Single Board Computer

The 8052.com SBC is a single-board computer designed to be both useful and instructive in illustrating certain concepts commonly encountered in 8052-based development. The SBC was designed specifically to form the basis of thorough technical discussions and tutorials consistent with those found on this website. Nothing was included in this SBC that does not serve a specific educational purpose and implementations that would confuse or unnecessarily complicate the design were intentionally omitted in favor of easy-to understand approaches so that the important topics could be covered without being lost in the details.

It is anticipated that in the near future additional projects, tutorials, and software will be made available at 8052.com which will be built specifically to operate with this SBC.

The board was developed with the Atmel AT89S8252 and Dallas DS89C420 in mind but can be used with any 40-pin 8052 pin-compatible derivative including traditional true-blue 8052s, 8032s, etc; Additionally, this SBC will work with the new Atmel AT89S8253 that has been announced to replace the AT89S8252. Its feature-set is such that the user may use the SBC simply to learn and master the 8052 microcontroller but may also subsequently use it as a base for his or her own projects and designs.

Please see the 8052.com SBC Page here at 8052.com for full information on this SBC including schematics, technical features, memory map, and software designed and tested to operate on this SBC.

SBC Features

- **SBCMON Monitor Program**. Microcontroller comes pre-installed with SBCMON monitor program which was specially developed to demonstrate the features of the SBC and facilitate development.
- **In-System Programming**. Atmel AT89S8252/AT89S8253 and Dallas DS89C420 can be programmed in-system without having to remove the microcontroller and without need for a part programmer or special cables.
- **Serial Port/UART**. Includes a single standard RS-232 compatible DB9 port that can be used to interface and communicate with external devices such as a standard PC. This is also useful in explaining the concept of serial communications.
- **16x2 LCD**. Includes a 16 column by 2 row LCD that can be connected to the circuit either by direct connection (the lines being driven directly by port pins of the microcontroller on P1 and P3) or as a memory-mapped device (accessed with the MOVX instruction). This is useful in explaining communication with external devices by controlling individual I/O lines. It also is useful in explaining the concept of memory-mapped devices.
- **4x4 Keypad**. A 4x4 matrix keypad which allows the user to input the numbers 0 through 9 plus provides 6 special function keys. This is useful in explaining the concept of key debounce.
- **Real Time Clock**. The SBC includes a DS1307 Real-Time Clock. In addition to providing time-keeping capabilities this part is instrumental in demonstrating inter-chip communication using the I2C protocol.
- **Serial EEPROM**. The SBC includes an Atmel AT25010A serial EEPROM which provides 128 bytes of non-volatile memory. More importantly it provides an opportunity to demonstrate inter-chip communication using the SPI protocol.
- **EPROM**. The SBC may be configured to run code out of an EPROM inserted into the circuit. This is useful if the microcontroller being used does not have any on-chip code memory such as traditional 8032s, 8052s, etc. Those using a microcontroller with on-chip code memory may choose to omit the EPROM.
- **Code RAM**. The SBC includes 32k of code RAM. This is RAM that has been wired such that the SBC will access it as RAM as well as code memory. This allows the user to download code into the RAM and execute it from RAM without having to burn a new EPROM or reprogram the microcontroller.
- **Access to all I/O lines**. All data, address, and relevant signal lines are exposed on connector blocks such that the SBC may easily be expanded to circuits on external PCBs.
- **Addressable LED**. P1.0 may be optionally connected to a LED for testing simple programs that cause the LED to flash.

- **Dual reset circuits.** The SBC includes both a traditional resistor-capacitor (RC) network to provide the reset signal and also includes a more reliable MN13811 reset supervisor. Which solution is used to provide the reset signal is selectable with a jumper.

SBC Options

The SBC is offered in three different versions.

- **Bare PCB**. The bare PCB is nothing but the printed circuit board. The customer must subsequently obtain all necessary parts and solder them onto the board.
- **SBC Kit**. This includes the bare PCB and all necessary ICs, capacitors, etc. necessary to build the SBC. The customer must solder the parts onto the PCB. Also includes keypad, LCD, and 120 VAC power supply.
- **Fully built and tested SBC**. This is a fully tested and functional SBC that is ready to be used by the customer. It need only be plugged in and it's ready to use. Includes keypad, LCD, and 120 VAC power supply.

Notes about the SBC

- Both the kit and the fully-built SBC will be shipped with a socketed Atmel AT89S8253. The customer may replace this microcontroller with any 40-pin DIP 8052-compatible microcontroller.
- Since the Atmel part that ships with the SBC has 16k of on-chip flash code memory, the kit and the fully-built SBC do not ship with an EPROM IC. A socket is, however, included in case the customer needs to use an EPROM in the future.
- PCB is 2-layer, silk-screen, green solder-masked and is rated to a maximum operating temperature of 125C (individual parts may not be rated to operate at this temperature).
- Dimensions of PCB are 6" x 3.5" (15.3cm x 8.9cm).
- Shipping weight of PCB alone is 1.5 oz, ships in envelope.
- Shipping weight of kit or fully-built SBC is 1.5 pounds, ships in small box.

Notes about shipping

- Shipping prices have been calculated for the U.S., North America, South America, and Europe. All other locations have been quoted as "rest of world" and shipping will be calculated as very expensive. If you are uncomfortable with the amount being charged for shipping or you believe you live in an area that doesn't have a specific shipping quote, please contact us and let us know your country and we will update the ordering system with precise shipping charges for your location.
- Orders for U.S. customers will be shipped from the United States. Orders for non-U.S. customers will be shipped from Mexico or the United States.
- For international orders, the value declared for customs will be the price paid for the product. We cannot declare lower values.

Product Options and Pricing

SBC8052-Built 8052.com SBC fully built and tested **$ 249.00**

SBC8052-KIT 8052.com SBC Kit: PCB, ICs, and parts **$ 159.00**

SBC8052-PCB Bare PCB for 8052.com SBC **$ 25.00**

(Quantity Discounts starting at 5 units)

Distributor Information

Vault Information Services (*www.8052.com*)
8174 S. Holly PMB 272
Littleton, CO 80122 U.S.A.
Phone: +1 (303) 439-0909
Fax: +1 (303) 465-6217

Vendor Information

Vault Information Services
8174 S. Holly PMB 272
Littleton Colorado 80122 U.S.A.
Phone: +1 (303) 439-0909
Fax: +1 (303) 465-6217