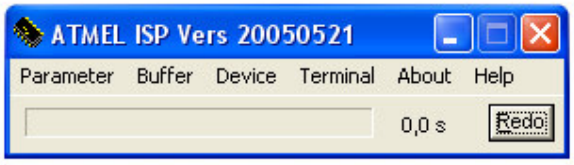
ATMELISP



A utility for In System Programming the

AT89S8252, AT89S52, AT89S53, AT89S8253,
AT89LP2052 and AT89LP4052

By Ulrich Bangert, DF6JB

History

Some time ago I needed to put some bytes of code to an AT89S8252's **code** memory as well as some bytes of data to its flash **data** memory. While my device programmer would nicely handle the **code** memory part of the story, the same would not hold true for the processor's flash **data** memory.

It seemed that putting some data into the processor's *data memory* **without the interaction** of a program that had been loaded to the processor's *code memory* before, was kind of problem. This was the moment when I started to scan the internet for a utility program that would do the job using the processor's ISP features. Result: None… or I to foolish to find one (supply whatever is more likely).

This made me wondering whether my problem was really such a challenging task. Let's see: ATMEL says ISP is really easy, just bring the processor in the reset condition, clock some bits to a port pin using a second port pin as a shift clock input and… that's all. Together with some additional "dos" and some "don'ts" there seemed to be nothing mysterious about ISP at all and I became even more curious about the question why the internet was not full of utilities for that purpose.

Nevertheless, I was in a hurry and decided to write a utility of my own. Please take my word that I did not mean that I could do really perform better than someone else! It was just the fact, that I found nothing at all, that put me to work.

However, in the meantime ATMELISP has grown from a very simple tool with only basic functionality to an instrument with a number of interesting features. I hope you can agree…

Features

- Uses a standard PC RS232 port for ISP
- Works under Win 95/98/ME/NT/2000/XP
- No limitation concerning the serial port number, i.e. Com1: up to Com255: may be used
- Very flexible scheme to map RS232 pins to ISP function pins handles a lot of different hardware arrangements

- Supports AT89S8252 (8K flash code memory + 2 K flash data memory) as well as AT89S53 (12 K flash code memory), AT89S52 (8K flash code memory), AT89S8253 (12K flash code memory and 2K flash data memory), AT89LP2052 (2K flash code memory) and AT89LP4052 (4K flash code memory)

- Reads BINARY as well as INTEL HEX files into a internal buffer

- Buffer may be edited by a built in combined HEX / ASCII editor featuring an almost unlimited undo function

- Code memory (AT89S8252 + AT89S53) & data memory (AT89S8252 only) may be read back from the processor into local buffer for inspection and editing

- Buffer may be stored on disk as a binary file

- Can program the processor's lock bits 1 - 3

- Can perform a "chip erase", which is necessary if you want to further program a processor when the lock bits have been set once

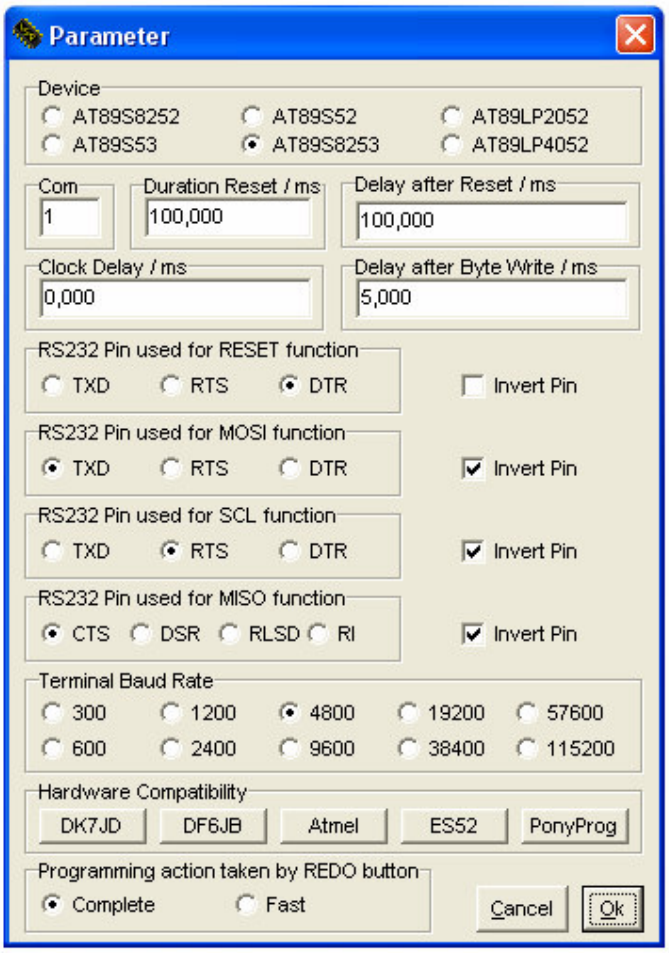
- Supplies a built in terminal, that is: You may communicate with your micro-controller application using the same serial port that is used for ISP downloading. Very effective for debugging

- Has a redo feature to ease the typical program development turnaround scheme

Using ATMELISP

Most of the program is very straightforward to use, at least if you are familiar with micro-controller programming. However, due to its universal pin mapping scheme, you may find ATMELISP more difficult to setup than other programs.

Setting up the Parameters



The setting of the **Com**: port and the type of **Device** is a self explanatory matter.

Duration Reset / **ms** This is the time in ms that the program waits after it has raised the reset line to let the reset really take place. If your hardware arrangement has kind of resistor-capacitor combination connected to the processors reset input, the time constant of the resistor-capacitor combination may limit the minimal length of an external supplied reset pulse. This value should be set to at least 2 times greater than the resistor-capacitor time constant. If you are sure, that all other parameters are set ok, but still ATMELISP does not work, it may be a good idea to increase this value.

Delay after Reset / **ms** This is similar to above, however for a change of the reset line into the opposite direction. It is the time that the program waits between releasing the reset line and any other further action. It is always safe to set this to the same value as **Duration Reset** / **ms**.

Clock Delay / **ms** ATMEL states, that the maximum shift clock frequency for ISP is **CPU Clock** / **40**. For a 11.0592 MHz clocked processor this makes 276 KHz clock frequency or about 4 µs clock period. My 400 MHz Pentium II does by far not reach up to this value, so an additional delay is not needed in my case. However, with faster PCs coming up every day, it might be necessary to induce a additional delay! If necessary adjust this value very careful in 0.1 ms increments. Otherwise you may end up with a very slow download performance. Remember: This is a delay **after every single bit** send to the processor.

Delay after Byte Write / **ms** Programming flash memory involves a wait time after each programmed byte. ATMEL states that their devices need not more than 3 ms wait after a byte write. So it is a good value to start with. Perhaps you will never have to change this value. With the AT89S8253 a minimum of 4 ms seems to be necessary.

RS232 pin used for RESET function
RS232 pin used for MOSI function
RS232 pin used for SCL function
RS232 pin used for MISO function

Communication with the processor via ISP involves the physical handling of four processor pins:

- RESET
- MOSI
- SCL
- MISO

RESET is self explanatory and is a output concerned the PC which is to program the device and a input concerned the device itself.

MOSI stands for "**M**aster **O**ut **S**lave **I**n" which means: A output concerned the PC which is to program the device and a input concerned the device itself. This is the pin used to clock information **into** the device.

MISO stands for "**M**aster **I**n **S**lave **O**ut" which means: An input concerned the PC which is to program the device and a output concerned the device itself. This is the pin used to clock information **out of** the device.

SCL is the shift clock input for clocking information in and out via MOSI and MISO.

Because RESET is an output concerned the PC, any of the RS232 output pins (TXD, RTS, DTR) may in principal be used to serve as the RESET signal. That is the reason, why some hardware designers have chosen to do it this way and others have chosen to do it that way. To be compatible with every hardware scheme, you may choose free which hardware scheme is used on your board. For a reason which is to be discussed later, it is not a good idea to use TXD for the RESET function. However, if your hardware is made that way, you have to configure the program according to that!

Because MOSI and SCL are outputs concerned the PC, what has been said about RESET holds true for them, too. Note, that not two or more functions may share the same RS232 pin!

Because MISO is an input concerned the PC, any of the RS232 input pins (CTS, DSR, RLSD sometimes called CD, RI) may serve for the MISO function.

You may have heard that micro-controller pins usually use CMOS-TTL levels of voltage to indicate their logical condition while RS232 pins use RS232 levels.

To say it in very simple terms: micro-controller pins like voltages **between 0 and 5 Volts**, with 0 Volts standing for a **0** or **Low** condition and +5 Volts standing for a **1** or **High** condition.

In contrast, RS232 pins like voltages **below -5 Volts** to represent the **one** logical condition and voltages **above +5 Volts** to represent the **opposite** condition. Note, that I did not say which is which until yet. That comes from the following problem: Since the levels used by micro-controller and RS232 pins are that different, one cannot simply connect them to each other. Instead one has to use kind of "level conversion circuit" between a micro-controller pin and a RS232 pin.

Unfortunately it is possible to built level conversion circuits **with** or **without** level **inversion**. A class of well known integrated TTL <-> RS232 converters, the MAX2XX from MAXIM (you may have read about the MAX232 anywhere else before) uses level inversion. That is: a -5 Volts on a RS232 pin translates into a +5 Volts at the TTL pin and a +5 Volts on a RS232 pin translates into a 0 Volts at the TTL pin and vice versa.

A lot of guys will insist that this is kind of "standard" but believe me: ISP programming a micro-controller via a RS232 port is basically thus far off any standard (concerned RS232 communication) that we do not need to talk about standards any more. Instead, let me tell you, that level conversion from RS232 to TTL without level inversion can be done quite simply by two diodes and a few resistors.

In fact, it doesn't matter at all, what the "standard" is. You have to find out, what your circuitry behaves like in **reality**.

For the reasons above I included the option to invert the signals on the RS232 pins in use. (**The "Invert pin" checkboxes**) The following rules apply to these checkboxes:

If a positive RS232 level translates to a positive TTL level and a negative RS232 level translates to a 0 Volts TTL level then the checkbox **should not** be checked.

If a positive RS232 level translates to a 0 Volts TTL level and a negative RS232 level translates to a 5 Volts TTL level then the checkbox **should** be checked.

Above I told you that RS232 to TTL conversion is easily done by a few passive components. Unfortunately the other way from TTL to RS232 is not as easy as that because passive components usually do not generate negative voltages (necessary for driving the RS232 input) from positive power supplies.

For that reason be prepared to find at least one active level converter like a MAX232 on your board for the MISO function (which clearly needs the "invert pin" checkbox to be checked).

To make things even more badly, a MAX232 features **two** level converters for each direction. However, in case of ISP we have **3** outputs (one too many) and **1** input. For that reason be prepared to see some pins on your board inverted (the ones that use a MAX232) and others not (the ones with "homemade" level conversion).

Now for the good news: For some designs, that I got aware of, I included "ready to go" buttons. With a single press on the bottom most of the parameters will be set correctly. These "ready to go" buttons are currently available for

- DK7JD that is the button for the AT89S8252 board which Om Burkhard Kainka, DK7JD, published in the December 2001 edition of ELEKTOR, a German electronic magazine. Note, that ELEKTOR is available under different names in other European countries. So there are chances to find this article in your own natural language. Note: This board has one Sub-D connector for serial communication and one Sub-D connector for ISP. The full functionality of my utility (i.e. the terminal function on the same port as the ISP) is given, if you install two additional wires to connect pin 2 of K1 to pin 2 of K2 and pin 3 of K1 to pin 3 of K2. Use K2 to connect the board to he PC

- DF6JB, which (no question about that) covers my own (unpublished) hardware solution.

- ATMEL, which is the button for a hardware scheme that a friend of a friend of a friend found in a ATMEL evaluation kit

- ES52, which is the scheme for the ES52-Flash Board sold by AK-MODUL-BUS GmbH, Germany (www.modul-bus.de)

- PonyProg, which is the scheme for the PonyProg programmer (thanks to Franz Dreher for supplying this)

Whatever your setup with one of the buttons may be, the setting of every single parameter may be overridden by **individually** changing it.

If you find other well established hardware schemes, do not hesitate to inform me about that. I will be happy to include additional buttons for those schemes.

ATMELISP has a built in terminal and one thing that needs proper setup when the terminal shall be used is the terminal baud rate in the lower part if the setup window.

Note: In order for the terminal to function properly, some requirements have to be fulfilled:

- The RS232 TXD pin **must not** be used for the RESET function. The terminal sends serial data using this pin and this would toggle the device between normal and reset condition if it were used for the RESET function.

- No matter what other function TXD is used for or whether it is inverted or not: You have to connect this pin **via an inverting level converter** (say: a MAX232) to the processors RXD input (which happens to be P3.0).

- You have to connect the processors TXD pin **via a inverting level converter** (say: a MAX232) to the RS232 RXD pin.

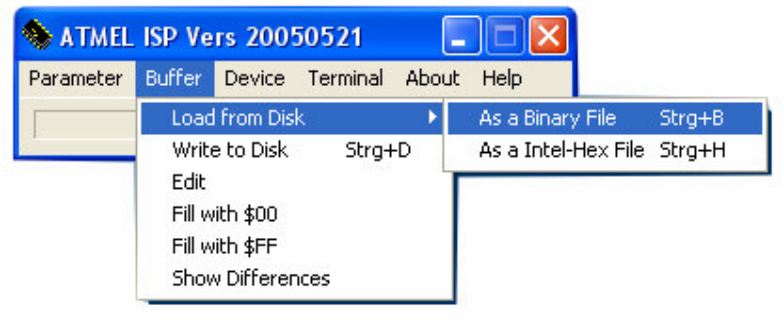
For the DK7JD board these requirements are fulfilled when the two wires mentioned above are installed. For the ES52 board from AK-MODUL-BUS these requirements are fulfilled without any modification.

As you will read later, ATMELISP features a so called REDO button to improve the typical program development design cycle. The REDO button can have one of two functions: **Complete** or **Fast** Redo. Keep in mind that you setup the Redo function in the lower left corner of the Parameter window.

ATMELISP will store all settings in an file, so that you have to perform the setup only once or when your hardware scheme changes.

Loading a file to the internal buffer

Simply choose <BUFFER> <Load from Disk> from the main menu and decide whether you need to load a Binary or an Intel-Hex-File as shown below.



A standard Windows File-Open-Dialogue will appear and will give you the possibility to open any file you want. Please note that

- For Intel-Hex files the default file mask is *.Hex but may be set to *.*

- For Binary files the default file mask is *.Bin *.Dat *.Com but may be set to *.*
- You will get a error message if you try to load a file that is bigger than the current processor's code memory (Binary) or contains memory addresses beyond the range of the current processor's code memory, like

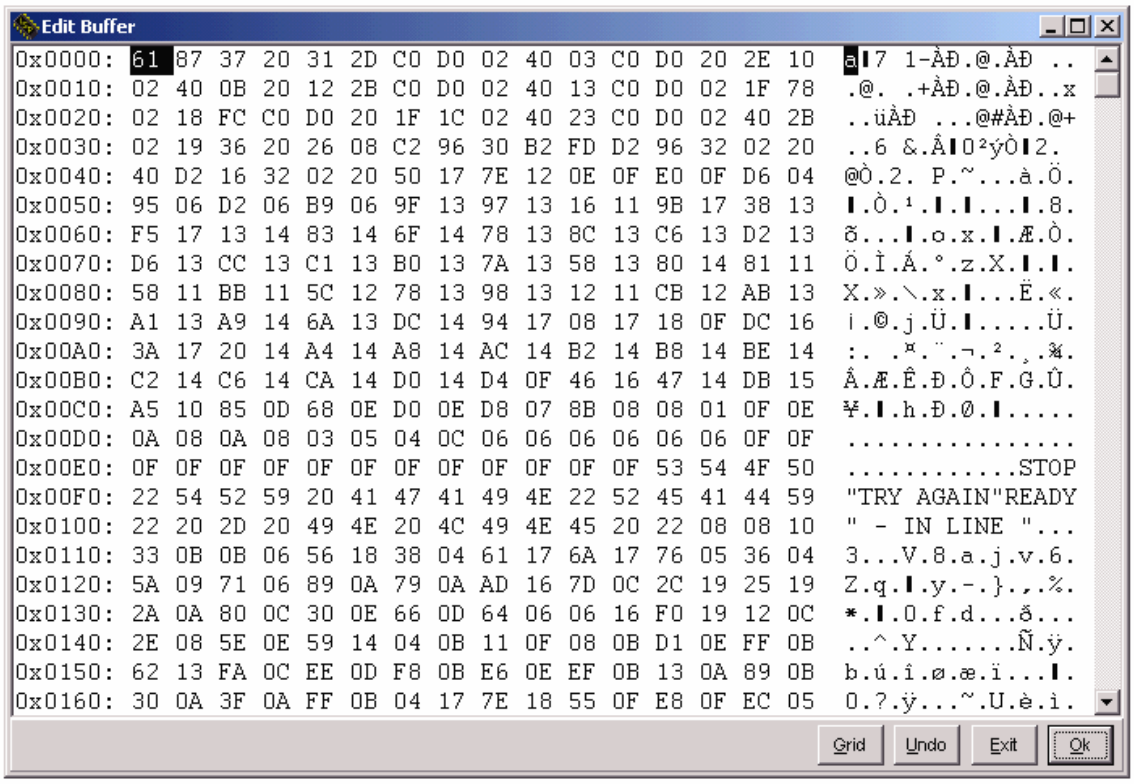
Please note, that you receive error messages of that kind **only** if you initiate loading of the file from the **Buffer** menu. If loading the file is initiated by means of a REDO action, ATMELISP assumes that it is not necessary to tell you things over and over again.

Writing the internal buffer to a file

Simply choose <BUFFER> <Write to Disk> from the main menu. A standard Windows File-Save-Dialogue will appear and give you the opportunity to save the buffer contents to any filename you wish. Files can only be stored in Binary and the default file mask is *.Bin.

Editing the internal buffer

The internal buffer may be edited by choosing <Buffer> <Edit> from the main menu. A Hex-Editor-Window will open as shown below.



Shown here are the first bytes of the INTEL MCS-51 Basic interpreter which had been loaded to the internal buffer before. By the use of the GRID button, a grid may be switched on and off. Use it or not, whatever you like best.

The cursor may be positioned with the mouse to either the hexadecimal or the ASCII part of the window. In the hexadecimal part you may overwrite any hexadecimal byte by a new 2-digit hexadecimal number. In the ASCII part you may enter a byte by just pressing its ASCII equivalent on the keyboard.

Any change that you make, may be undone by the UNDO button. UNDOS are done in the reverse direction, i.e. the last change is undone first. Change a few bytes and play around with the undo feature to get a feeling for how it works. Bytes that you changed are displayed in a different color, so it is easy to see, how the changes are undone byte by byte.

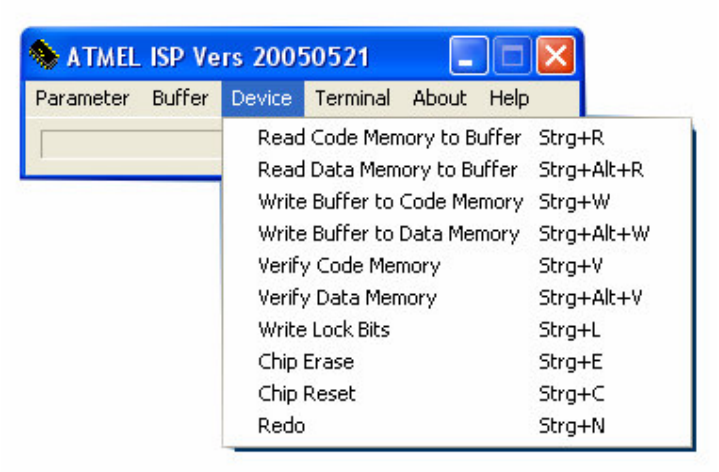
If you leave this window by EXIT all your changes will be undone. If you leave the window with OK, all your changes apply and cannot be undone if you open the window again.

Clearing the internal buffer

The internal buffer may be cleared anytime to $00 or $FF by choosing <BUFFER> <Fill with $00> or <Buffer> <Fill with $FF>

Read the device code memory into local buffer

Simply choose the first entry of the <Device> menu, as shown below



Read the device data memory into local buffer

Simply choose the second entry if the <Device> menu, as shown above. Note that this option is only available if the current device is an AT89S8252 or an AT89S8253.

Write the local buffer to device code memory

Simply choose the third entry if the <Device> menu, as shown above.

Write the local buffer to device data memory

Simply choose the fourth entry if the <Device> menu, as shown above. Note that this option is only available if the current device is a AT89S8252.

Verify Code Memory

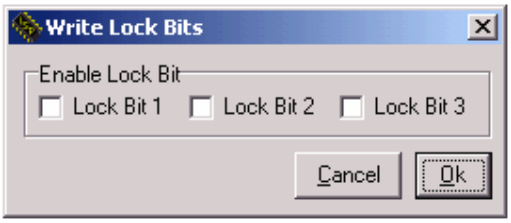
The buffer contents are compared to the processor's code memory. If any differences are found, a small window appears to tell you about. After that a window will show up all differences which verify has found between the processors code memory and the buffer. Note that you may open this window later again by means of <Buffer> <Show differences>.

Verify Data Memory

The buffer contents are compared to the processor's data memory. If any differences are found, a small window appears to tell you about. After that a window will show up all differences which verify has found between the processors code memory and the buffer. Note that you may open this window later again by means of <Buffer> <Show differences>.

Write device lock bits

Choose the fifth entry of the <Device> menu. A new window will open, giving you the opportunity to choose, which of the lock bits should be set. . Note a special feature of the AT89S52. With this device you will have to set Lock Bit 1 alone first then Lock Bit 2 alone first and then Lock Bit 3 alone in order to set Lock Bit 3.



Erase device code and data memory

Choose the sixth entry of the <Device> menu. A dialogue will appear, asking you, whether you are sure to do that. Use this option if you set any of the lock bits on a specific device. This is the only way to reset the lock bits.

Resetting the device

You may initiate a device reset any time by choosing the last entry in the <Device> menu.

Device window note

With the exception of a device reset, which involves rising and leaving a single pin only, all other actions from the device window first test, whether a device is really connected and ready for ISP. If this is not the case, you receive an error message, like

Please note, that you also receive a error message, if a device is connected to the port, but one or more of the parameters are set wrong. In other words: If anything is wrong, you will notice it with any action in the device window except reset. If everything works ok, you will see the main window's progress bar increment, indicating the progress of action.

The above mentioned test for an ISP ready device uses the following algorithm:

- Read first byte of code memory into backup location 1
- Write a $00 to first byte of code memory
- Read first byte of code memory into backup location 2
- Write a $FF to first byte of code memory
- Read first byte of code memory into backup location 3
- Write backup location 1 to first byte of code memory
- Compare backup location 2 to $00 and backup location 3 to $FF
- If both tests ok, we have a ISP ready device connected, otherwise not

Note that the above mentioned algorithm is not possible with an AT89S52, AT89S8253! With an AT89S52 or an AT89S8253 an automatic detect is not possible.

The Redo function

Whenever you load a file into the internal buffer of ATMELISP by means of the <BUFFER> dialogue, the program remembers

- The filename of that file

- Whether you loaded it Binary or Intel-Hex

By pressing the REDO button, ATMELISP again loads this file the same way you did it the last time and transfers its contents to the processor's **code memory**. Please understand that this is not a general REDO function which may repeat whatever happened before.

Instead it is intended to support the typical programmer's turnaround scheme, which consists of

- Editing the source

- Compiling and / or assembling to executable code

- Download the executable to the processor

While ATMELISP cannot help you with the first 2 issues, the third issue reduces to a single mouse click with ATMELISP. Once a file has been loaded to the internal buffer, you may use the REDO as often as necessary.

Fast Redo against Complete Redo

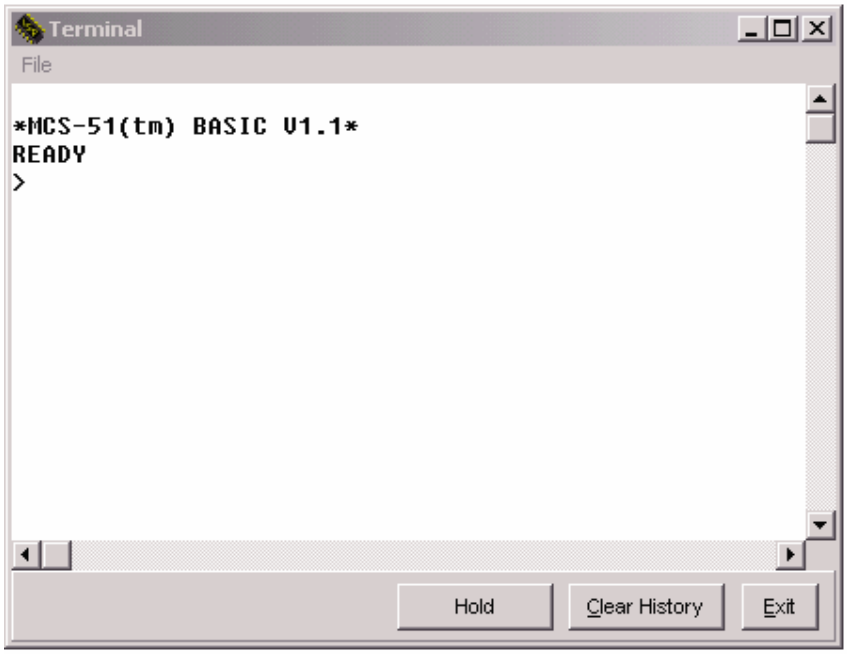
As we said in the beginning, the Redo button can act in two different ways: Fast or complete. A complete Redo will do exactly as described above: Reread the file into the buffer and then transfer **all of it** to the processor's code memory. While this is truly very safe, it is also kind of slow.

Folkert Deenik suggested a scheme in which every byte of the processor's code memory is read before it is programmed. So we would not program bytes again that do already have the correct value. Folkert states, that usually only a few bytes change their value between different program versions in the design cycle. While this is a good idea (programming flash memory needs more time than reading it) its exact measure of improvement is limited due to the fact that reading back bytes from the code memory is also a very time consuming action.

However, this made me thinking about the question: If it not possible (or too slow), to read back from the processor, why not use the information that we put into the processor the last time? In other words: Why not make a copy of the buffer contents, then load the buffer from the input file again, compare the old buffer contents to the actual one and then program **only the bytes that are different**? Exactly that happens with a fast Redo! Enjoy the new speed in the design cycle! The fast Redo option is not available with the AT89S52 and the AT89S8253!

Starting the built in terminal

Choose <Terminal> from the main menu. A terminal window opens as shown below.



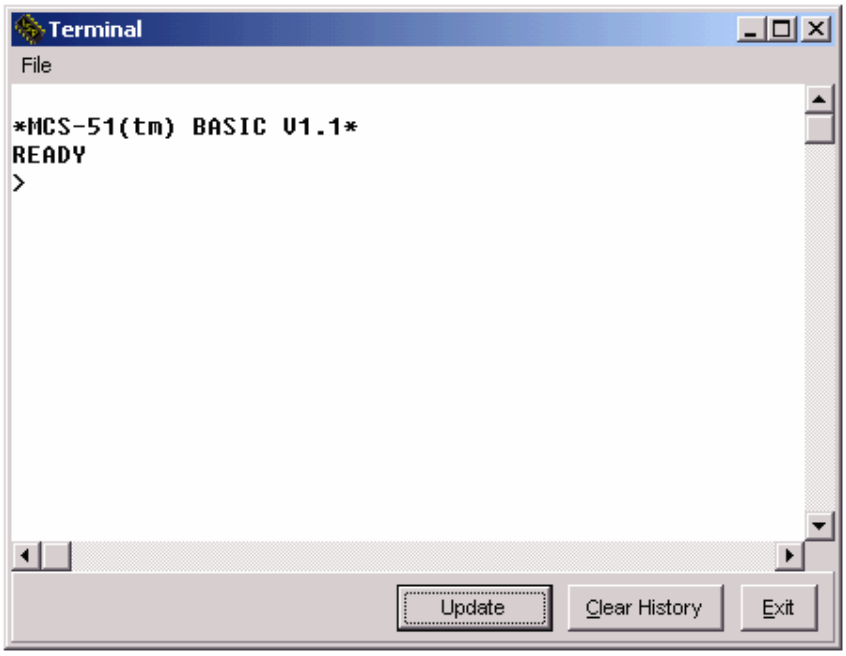
Shown above is the terminal window after the MCS-51 Basic interpreter has been loaded into the local buffer and the local buffer has been written to the device code memory. You have to press the <Space> key once because the Basic interpreter has a auto baud rate detection and needs a space character to correctly detect the baud rate.

The terminal window features a 10000 lines history buffer of 255 characters each. The history buffer may be cleared anytime by the <Clear History> button.

Whenever serial characters come in, the Terminal window appends them to the end of its buffer and adjusts the vertical scroller so that the new text is visible. This feature might be unwanted, if you

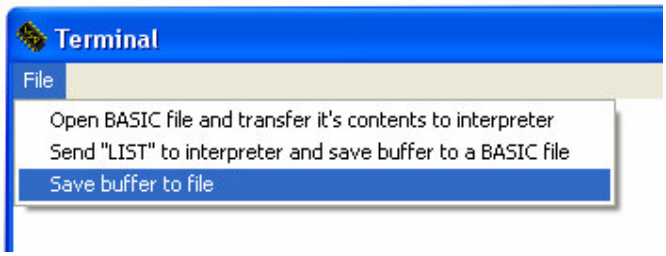
- happen to have a program that puts out something on the serial port on a regular base
- you want to scroll back to earlier outputs of that very program

Because whenever the program sends something new, you are taken away from where you just scrolled to. For that reason, a HOLD feature has been built in. Press the <Hold> button once and the window will change to



Notice that the button's caption has changed to <Update>. In this hold mode no more information will be sent to the screen, instead everything is sent to a local buffer. In this mode you may scroll up and down the window without getting disturbed by incoming characters. When you press the button again, you are in the update mode again. Notice that all characters, that came in during the hold mode, are now appended at the end of the window, nothing has been lost.

Since I thought the terminal feature would be used quite often in conjunction with the Basic interpreter, I introduced two goodies for the Basic programmer in the terminal window's file menu, as to be seen below



One is to load a Basic text file and transfer its contents line after line to the Basic interpreter. Since the Basic interpreter does part of the interpretation process when the input line is ended with the carriage return, it may need different time to do that due to the complexity of the input line. That is the reason, why a "normal" terminal program may show up problems when downloading files to the Basic interpreter. ATMELISP, in contrast, knows the properties of the interpreter pretty well and waits for the interpreter's ">" prompt before it sends a new line. That is a safe method. The second is to send a LIST command to the interpreter and write everything the interpreter lists to a user defined text file. With these two goodies ATMELISP may be called a complete Basic language development tool.

With the "Save buffer to file" you may store the contents of the terminal text buffer to a text file. Please note that due to an internal limitation a empty line cannot be stored into the file. In order to be stored, the line must contain at least one none-space character. Please note that also all space characters behind a none-space character are not stored to the file.

About

Opens a window showing one of the toughest guys in universe, me….ahem….

Help

Opens this text

Minimizing ATMELISP

If you minimize ATMELISP by means of a click on the minimize button or by means of the system menu's minimize entry, ATMELISP's icon will be displayed in the system tray and a click on this icon will open the main window again.

Terms of usage

ATMELISP is freeware. Freeware means:

- You may use the program for free, i.e. you may use the program without paying any charge or fee for it on as many computers as you want

- You may distribute the program to others for free, i.e. you may give it to other persons or organizations without paying any charge or fee for that. However, you are allowed to distribute it to others only under the term that you do not take any charge or fee for that yourself! If you distribute the executable you have to distribute the help file as well.

- You use the program *at your absolutely own risk*. While much time has been spend in making the program error free, I do not give any kind of guaranty for that. Nor do I take any responsibility for any harm that results from using the program.

Credits

Programming in the 21st century does by far not more mean: Write every line of code yourself. Instead, in a sense of "software engineering" one searches for pieces of code, which (when combined in the correct way) solve the problem. DELPHI with its *visual component library* (VCL) suits this idea very well and a lot of highly integrated components are available as freeware. Following is the list of people that supplied freeware components which I used in my utility and I would like to thank all authors for putting their work into the freeware community.

Danny Thorpe (dthorpe@subliminal.com) for the console component used as the terminal screen

Scott Pinkham (geolase@geocities.com) for the qccom32 component used for RS232 related stuff

Markus Stephany (mirbir.st@t-online.de) for the hex editor component

Edy Hinzen (EdyHinzen@aol.com) for his bigini component

I would like to thank Hartmut Päsler, DL1YDD, for our discussions about ISP in general and his suggestion of the redo function.

I would like to thank Burkhard Kainka, DK7JD, for supplying different AT89S8252 boards for test purposes.

I would like to thank Roland Jopski for supplying an AT89S8253 for test purposes

ATMELISP is presented to you by Ulrich Bangert
df6jb@ulrich-bangert.de
Ortholzer Weg 1
D-27243 Gross Ippener
Germany
Tel. +49 (0) 4224 95071

Hardware

From time to time I am asked: What kind of hardware do you prefer for ISP program development using the AT89S8252? My usual answer is: My own! The next question then is: Why that?

This question is easily to be answered: Have a look at the printed circuit boards available. Usually you are told that a specific board is a “development system” for the AT89S8252. That means: You can use it to develop the code, however it is not the system that the code is really expected to work in. This is called the “target system”. Once you think that the code is ready, you program the processor in the development system using ISP, unplug it from the development system and plug it into the target system.

According to my experience this is the point of time when the real problems are beginning to start which leads to changes in the code which in turn leads to unplug the processor from the target system……supply the rest by your own. If the target system has all the provisions made for ISP you are a lucky man but not all target systems are constructed with ISP in mind.

That made me think that I need a development system that transforms **every** target system into an ISP development system. The circuit diagram of my system is shown in ISPSOCKET.PDF and a photograph can be seen in ISPSOCKET.JPG.

As you can see in the photograph the circuit resides on a small PCB which is only marginally greater than the processor’s footprint itself. The AT89S8252 is plugged into a socket on the top of the PCB which gives a lot of room under the processor for all other parts with the exception of a jumper block and the plug for the cable connection to the pc.

While the circuit is really simple, don’t overlook that it has its tricky points. J2 is the socket on the upper side that the processor is plugged in and J1 is it’s counterpart on the lower side which is plugged into the target system. By means of J3 all ISP signals except MISO may be disconnected from the target in case that comes necessary. MISO is uncritical in that aspect because it is an **input** of the development system.

If the terminal function of ATMELISP is to be used, a jumper on J3 7-8 connects the pc’s serial output to the ATMEL’s serial input. Whether this pin is connected to the rest of the target system in that situation or not is easily changed with/without a jumper on J3 5-6.

Even with the ISP jumpers plugged, the development system puts the ISP pins into a high impedance state during normal processor operation (reset low) due to the 74HC126 line drivers. The reset circuit using U2C, R2, R3, D1 and D2 is designed in such a way, when the cable to the pc is removed the processor is in the normal (non reset) state. So you may download a program to the target system, remove the cable and then transport it elsewhere.

Beneath the AT89S8252 and the AT89S53 I like ATMEL’s AT89C1051, AT89C2051 and AT89C4051 due to their small footprint and their processing power. **These** are not ISP processors! However when I have to develop code for one of them, I use an adapter that transforms the AT89S8252’s 40 pin footprint into the AT89C1051’s 20 pin footprint.

Ok, this transformation works only for the pins that 1051 really has. And of course some care has to be taken in code development: Do not use more flash memory than the target processor provides. Do not use any of the features that the target processor does not provide. However, with these simply rules in

mind I use this adapter for ISP based code development for the small ATMEL processors. See its circuit diagram in ISPADAPTER.PDF and a photograph of it in ISPADAPTER.JPG.

Changes

2002-01-12	Added support for ES52-Flash board Secured against start of multiple instances
2002-01-12	Added HOLD function in Terminal window
2002-04-29	Added Compare function Added Hide function Added Icon for System Tray
2002-05-05	Added support for PonyProg programmer Now ATMELISP remembers the MRU path and uses that as the default in the File-Open and File-Save-Dialogs
2002-06-16	Thanks to a feedback of Jörg Hampe a bug was corrected that made it impossible to erase a chip which's lock bits had been set before. Thanks to the feedback of 14 year old Dutch Gentleman Joris, ATMELISP may now be called with a filename as a command line parameter. ATMELISP uses the file extension to decide whether it is hex or not and transfers the file contents to the chip's code memory. Note: With this usage of ATMELISP the program is started, then it downloads code to the AT89S8252 and then it **releases** the serial port and terminates itself. This scheme will only work, when the ISP hardware that you use puts the processor in a run (not reset) state when connected to a unused serial port!
2002-08-25	The terminal window has algorithm which tries to display a whole number of lines in the console no matter what window size the user sets. This works smoother now.
2003-02-08	Folkert Deenik of the Netherlands inspired me to implement the Fast Redo Function. Hope, you'll like it, Folkert!
2003-05-24	Added Menu shortcuts due to suggestion of Manfred Flume
2003-06-18	This is not a new software version. However, there are fundamental changes since from now on my own suggestion for a hardware design is included.
2004-05-01	Some minor bug fixes Uses now the more standard minimize button instead of the menu's "Hide" entry. ATMELISP is minimized to an icon in the system tray. Thanks to hardware donation from Uwe Sieg-Söder ATMELISP now supports also the AT89S52. This device, in contrast to the AT89S8252 and the AT89S53, has a nasty feature: Once a zero (low) has been programmed into a flash memory's bit position, this bit position cannot be reprogrammed with a one (high), as we know it from ancient EPROMS. Instead, the flash memory has to be erased totally before it can be programmed new. ATMELISP will do the automatic total erase before it writes the buffer to code memory, so in the most cases you will not get aware of it. However, the chip detect feature of ATMELISP and the Fast Redo option rely on the ability to modify single bytes and therefore these options are not available with the AT89S52. The same applies to the AT89S8253!
2005-05-21	Thanks to hardware donation from Roland Jopski ATMELISP now supports also the AT89S8253. Added support for the AT89LP2052 and the AT89LP4052 but not tested yet, due to lack of hardware units. I removed a lot of misspellings and wrong typing from the manual.
2005-08-27	Due to a programmer's error (shame on me…) the size of the AT89S52's program memory was set to zero as soon an someone tried to edit the buffer. Thanks to Günter Göbel for pointing at that error!
2005-10-24	Included a button for the KISS8051-Board in the parameters section Increased Terminal buffer from 5000 to 10000 lines Increased Terminal buffer line length from 128 to 255 characters Terminal buffer may be stored to a text file from now on
2005-10-31	Thanks to Richard Delmelle, ON7ZN, an error concerning reading & writing the data memory of some processors has been found and removed.
2005-11-09	AT89S52 chosen as default processor when button for KISS8051 is pressed
2006-04-18	Experimental support for AT89S2052 and AT89S4051 built in but no feedback so far