Features:

-super small – dimensions 44 x 39 x 5,5 mm
-super fast – write up to **12,5kB/s**, read up to **14,5kB/s**
-uses cheap storage medium – small MICRO SD cards
-supports **FAT16** and **FAT32** file systems
-can read, write, verify **flash** and **eeprom** memory
-can read, write, verify **fusebits** and **lockbits**
-write and read to **BIN**, **HEX**, and **TXT** files
-can set default values of fusebits, erase memories
-cheap, easy to obtain, **LPH7779** graphic display
-shows funny animations after every operation
-standard programming header – Atmel **6-PIN ISP**
-has a function of auto-update its own firmware (from SD)
-very simple to use, 4 buttons navigation
-user-setting menu
-programming speed auto-selection (up to **4MHz**)
-Operates at 3V, programs chips supplied from **3V to 5V**

Usage:

Turn on by holding the **LE**(left) button. First, the welcome screen and version info will appear – this vanishes after a while and we are ready to navigate in menu. With **UP**(up) and **DN**(down) buttons we chose a desired option, with **RI**(right) button we accept operation/enter the menu option, with **LE**(left) we cancel operation/step back in menu, which looks like this:

Flash:	**Eeprom:**	**Fusebits:**	**Lockbits:**	**Settings:**
>Write	>Write	>Write	>Write	>Auto Verify
>Read	>Read	>Read	>Read	>Auto Erase
>Verify	>Verify	>Verify	>Verify	>Show Anims
		>Default	>Chip erase	>Progr bar
				>Files type

Functions description:

Write – writes data from file to a selected memory of chip. Choose file from list. In case of fusebits and lockbits, writes settings bytes from txt file. File length is a length which is written to memory, rest of memory will not be touched.
Read – reads data from selected memory to a file. File is made automatically in proper directory. Name format is “x.yyy” where x is a file sequential number and yyy is the extension (bin or hex), i.e. “4.bin” or “22.hex”. Name is displayed after file saving, it’s good to notice it somewhere. Whole chip memory will be read, no matter how many data it contains. In case of fusebits and lockbits, settings bytes will be saved into txt file.
Verify – verifies data from selected memory with data from selected file. Result is “PASS” or “FAIL at x” where x is the address of first byte in which inconsistency occurred. In case of fusebits and lockbits, verifies settings bytes.
Default – resets all fusebytes to factory settings.
Chip erase – internal command “chip erase” erases whole memory and resets lockbits protection.

Options description:

Auto verify – allows to verify written memory (just like the "verify" option) but it is automatically performed after each one write process. Settings: ON or OFF. Concerns only flash and eeprom operations, fusebits and lockbits are read irrespectively after write.
Auto erase – automatically performs the "chip erase" command before every each write process to flash memory. Settings: ON or OFF. Concerns only flash memory, which must be erased each time before writing. Exceptions is, when we know that part of flash we want to write is already erased, and we have some data at the end of this memory (i.e. bootloader). Disabling this options will cause overwrite data without erasing, and if file is prepared properly, it will not touch the bootloader space.
Show anims – shows funny animations after each operation, depending if it was successful or not. Settings: ON or OFF. Animation can be break by pressing the **LE** button. These are funny animations, i'm sure that seasoned internet user will recognize them :) Animations are repeatedly opened BMP files from proper directory – so they can be easily replaced with our own images. File format: Bitmap (*.BMP), 84×48 pix, monochromatic (2bit). Playing sequence is not alphabetical, files are opened in that order in which they were copied to memory card.
Progr bar – progress bar for write, read, and verify operations for flash and eeprom memories. Settings: ON or OFF. Progress bar slows down programming process, see "speed tests" section.
Files type – allow to chose file type on which we will be operating. Settings: BIN or HEX. Concerns operations on flash end eeprom memories, and depending on the choice, only HEX or BIN files will be listed when opening files list. When reading, also appropriate file will be created (in case of HEX files, also CRC will be calculated).

Memory card, files:

Files must be placed in memory card in proper directories. Flash memory files in "flash" directory, eeprom in "eeprom", fusebits in "fuseb", lockbits in "lockb". "System" directory contains other files needed for programmer, including µControllers database file, settings file, BMP's images. CONTENT of "SD-FILES" directory must be copied to card.

If there is no doubts when it comes to BIN or HEX files, because these are generated by compilers, then we need to describe fusebits and lockbits file structure. These are standard txt files in which we type values in hexadecimal character:

Fusebits file should contain data in sequence for: fuse low, fuse high, fuse extended – in HEX format. In example, the content of fusebits file for Attiny45 should look like that: 62DFFF. File can be longer, you can add there any comments etc, but only first 3 bytes will be used (6 chars). If chip has only two or even one settings byte, then only two or one will be used.

Structure of lockbits file is analogous to fusebits file. The only difference is that, only one byte will be used. Please note that unused bits are always must be set to "1". In example, if we want to enable the LB0 and LB1, we need to write the "FC" value (11111100). The same rule applies to fusebits, but even if we try to enable not-existent bits (write 0), then nothing bad happens, we only get verifycation error because these unused bits will always read as "1".

When reading fusebits or lockbits, programmer will create files accordingly to described above scheme.

Selecting appropriate memory, and then write or verify function, list of available BIN/HEX or TXT files from proper directory will be displayed. Files are not sorted alphabetical, but in the order in which there were copied. User can place any amount of files in any directory, and all of them will be listed. Select a file with **UP** and **DN** buttons, choose with **RI** button, or cancel operation with **LE** button. Screen can display 6 names at once, press **DN** button when last file is selected to view next card of 6 names. Cards can be only switched forward (down), after getting to the end of list, first card will be displayed.

Device supports **MMC** and **SD** cards, tested by me: (microSD) 128MB, 512MB, 1GB, 2x2GB. No restrictions with file naming, but names will be displayed in DOS 8.3 format (8 chars + extension). Longer names will be separated with tilde char (`), and if these names will be the same within first 8 chars, files will be tagged with sequential number. Diacritic chars and other weirdos will not be displayed either, so name your files in way which you can later identify them.

In plans: SDHC support.

Supplying, voltages:

There is a place for two CR2032 battery holders on bottom site of the pcb – when using these batteries, operating voltage will be 3V, and current efficiency of these batteries will be insufficient to power up the target chip – voltage drop will kill programming process or will mess up data in memory card. Instead of soldering these holders, you can connect your own battery with higher current (i.e. from cellphone), but make sure to not exceed maximum operating voltage which is 3,6V.

Device is designed to work with battery, but there is no problem to use proper power supply.
Nominal operating voltage: 3,3V
Maximal operating voltage: 3,6V
Minimal operating voltage: 2,7V*
Power consumption at start: up to 100mA*
Power consumption while programming: up to 10mA*
Power consumption in menu: 5mA
Power consumption when off (power-down): 0.10µA

*depends on SD card

Programmer works with exactly that voltage which it gets from battery, because it does not have any voltage regulator on board. The whole programming process is also at this voltage (high state). Target chip can be programmed from 3V to 5V when programmer works with 3V. From one side, MI-SO input line is connected trough resistor and 3V3 zener diode to protect input line from voltages higher than VCC. From other side, target chip powered from 5V will recognize 3V at SCK and MO-SI lines as high state (see datashhet: electrical characteristics > dc characteristics > **V_{IH} = 0.6 VCC**, this is 0.6*5V=3V – guaranteed value).

There is no mechanical switch, turning on and off is made by holding the **LE** button. After switching off, programmer will cut off power from display and memory card, and goes into deep sleep mode, taking only (typically) 0,1µA from battery.

ISP – Connectors:

Programming header – 6pin male connector in **ISP ATMEL** standard. Pin allingment is shown on drawing – plug indentation on the bottom side of pcb. I'll suggest to make yourself an universal programming cable with combination of 6pin atmel and 10pin kanda headers. After programming ends, **SCK**, **MOSI**, **MISO** and **RESET** lines becomes transparent to the target device.

ISP – programming speed auto-selection:

Programmer automatically selects the best programming speed (SCK line speed) before every programming operation. There is 7 regulation steps: 4MHz, 2MHz, 1MHz, 500KHz, 250KHz, 125KHz, 62,500KHz. Theoretically, 1/4 of target operating frequency will be chosen, but no always. Every time programming mode in target is initialized, programmer starts from highest frequency and performs speed test by reading signature bytes 10 times – if error occurs, speed is decreased and initialization with test are performed from the beginning. If the lowest speed fails, programmer returns "no answer" error. In case of very long programming cable it is not possible to detect speed errors in this short test. I suggest to use not longer cable than 20cm. Given SCK values apply when programmer chip runs at 8MHz.

ISP – assistant clock signal

Because programmer is battery-powered, i resigned from using the standard feature of VCC pin (pin 5 in 6 ISP header) and there is no possibility to give supply for target chip trough this line. Instead of, assistant 8MHz clock signal is present on this line, which can be used for targets with some weird/external clock enabled. This signal is connected trough 10K resistor, so you don't have to mod your current programming cable, no matter if you got 5V from target on this line.

Firmware update:

If newer version of firmware will show up, it can be updated in two ways:
ISP: Connect your ISP programmer into ISP-6 header and close "SLF-PRG" marked pins – programming header will become update header (just like in the USBasp programmer).
BootLoader: Device has a build-in bootloader which can update the firmware in very simple way! Get the update BIN file, rename it to "000.bin", and place it into main (root) directory of your memory card. If after powering-up display is blank, thats good! Firmware file has been successfully recognized and device awaits for your confirmation. Hit the **RI** button to confirm. Flashing takes few seconds, and after success device will power-up and you should see new version on welcome

screen. Now, delete “000.bin” from sd card to prevent enabling actualization mode. If you put your file according to these instructions and actualization mode will not start – you got an SD card which is not supported in actualization mode, but can be supported in programmer mode. Just try another card.

Remarks:

Fusebits: **Ext:07, high:D0, low:A2**
(internal 8MHz oscillator without /8 division, enabled CLKO output, brown-out detection 2,7V, reset vector at bootloader, bootloader size 4kB, EESAVE enabled).

Internal Atmega328P oscillator can be calibrated in very wide range. Fabric calibrated to 8MHz can be “calibrated” by user from 50% even to 200%, i.e. from 4MHz even to 16MHz. Change bit4 to “1” in “config.ini” file will chose the maximum available speed – this will significantly speed up programming process, but manufacturer don’t guarantee correct work in such conditions (supply voltage vs frequency). By default, this setting is disabled.

Display which i used is original (taken from 3310 cellphone) – it has a PCD8544 driver, 84×48 organization, standard set of instructions, and it can work with 4MHz signal – and it works good at 4MHz. If your display shows only trash or it don’t show anything at all, there can be two reasons for that:
a) it is a Chinese replacement – it has different driver (not PCD8544) – in “config.ini” file, bit6 switch to 1 – display will be threated in different way, but this do not guarantee its correct work.
b) it can’t work at 4MHz – in “config.ini” file, bit5 switch to 1 – display will work with 2MHz, that speed should be good.

Display needs a proper capacitor for build-in voltage converter (C2, C3), nominal value is 1µF. Mine works very well with 200nF capacitance. If you experience problems with contrast, increase this capacitance – more capacitors can be soldered in place of C2 and C3 – just solder them vertically. If your LCD have a copper solder pads, you can easily solder him to PCB with wires. If it have a glass contact pads, use original rubber contact strip (zebra strip) which was in cellphone, lcd in that case must be well mounted. In my case, i used parts of “E+J connector” case. You can use just anything which internal dimension is 5,2mm.

Memory card, depending on what type we get, can take up to 100mA while init – this is a huge current for small battery. C4 capacitor helps to keep this voltage for initialization process, and will be good if this cap has high capacity, but not too high because it is charged from three µC pins, which we can’t damage. For me, 10µF capacitor is sufficient to start every SD card that i tested.

Programmer works only in its own directories and it wont touch rest of files/dirs located on card. **ATTENTION** – remember that file structure can be damaged if some error occurs, and other data contained on card may be permanently lost or will be hard to recover. This can happen when you insert/remove card or cut-off power while reading or writing operation is pending. May also happen (very rarely) where programmer will freeze when trying to open file list in some directory. Solution – delete last saved by programmer file in that dir.

39mm x 44mm x 5,5mm size PCB, doublesided. Vias pads are placed in easy accessible places (not under ic packages etc), so its easy to make it at home. Used micro SD card slot is the (i think popular) MSDE208, footprint from **here** (from 500873-0806 socket) – and as seen, its not that hard to find a few of compatible sockets, just search for socket which has pads “under itself” and compare with datasheet – it should fit.

Speed tests:

Speed test made with Atmega644A as target running at 25MHz, SCK speed 4MHz, flash memory, 64kB file size, programmer speed 16MHz:

BIN file, disabled progress bar:
Write from file: 5,1s.
Read to file: 4,4s.

BIN file, enabled progress bar:
Write from file: 5,1s.
Read to file: 5,0s.

HEX file, disabled progress bar:
Write from file: 12,5s.
Read to file: 11,6s.

HEX file, enabled progress bar:
Write from file: 12,5s.
Read to file: 12,2s.

As seen, the higher speeds are achievable when working with BIN files, because file length is the data length and there is no data formatting etc. Read from HEX files first require to read whole file and calculate the length of data, and when write to HEX we need to form data and calculate CRC of each row – so its take more time.
Progress bar does not slow down the programming process if target chip is programmed in page mode (like in above test). When programming in byte mode or reading (which is always performed in byte mode), enabled progress bar can significantly slow down the process.

Supported chips list:

Green – tested chips.
Note that yet not all chips are fully supported – see “chip.db” file. Incomplete entries means that only fusebits and lockbits operations are working.
Please report chips that are working good, i will add them to the list!
1kB:
AT90s1200, Attiny12, Attiny13/A, Attiny15
2kB:
Attiny2313/A, Attiny24/A, Attiny26, Attiny261/A, Attiny28, AT90s2333, Attiny22, Attiny25, AT90s2313, AT90s2323, AT90s2343
4kB:
Atmega48/A, Atmega48P/PA, Attiny461/A, Attiny43U, Attiny4313, Attiny44/A, Attiny48, AT90s4433, AT90s4414, AT90s4434, Attiny45
8kB:

Atmega8515, Atmega8535, Atmega8/A, Atmega88/A, Atmega88P/PA, AT90pwm1, AT90pwm2, AT90pwm2B, AT90pwm3, AT90pwm3B, AT90pwm81, AT90usb82, Attiny84, Attiny85, Attiny861/A, Attiny87, Attiny88, AT90s8515, AT90s8535

16kB:

Atmega16/A, Atmega16U2, Atmega16U4, Atmega16M1, Atmega162, Atmega163, Atmega164A, Atmega164P/PA, Atmega165A/P/PA, Atmega168/A, Atmega168P/PA, Atmega169A/PA, Attiny167, AT90pwm216, AT90pwm316, AT90usb162

32kB:

Atmega32/A, Atmega32C1, Atmega323/A, Atmega32U2, Atmega32U4, Atmega32U6, Atmega32M1, Atmega324A, Atmega324P, Atmega324PA, Atmega325, Atmega3250, Atmega325A/PA, Atmega3250A/PA, Atmega328, Atmega328P, Atmega329, Atmega3290, Atmega329A/PA, Atmega3290A/PA, AT90can32

64kB:

Atmega64/A, Atmega64C1, Atmega64M1, Atmega649, Atmega6490, Atmega649A/P, Atmega6490A/P, Atmega640, Atmega644/A, Atmega644P/PA, Atmega645, Atmega645A/P, Atmega6450, Atmega6450A/P, AT90usb646, AT90usb647, AT90can64

128kB:

Atmega103, Atmega128/A, Atmega1280, Atmega1281, Atmega1284, Atmega1284P, AT90usb1286, AT90usb1287, AT90can128

256kB:

Atmega2560, Atmega2561

Files:

Before you burn flash! If you don't have a programmer with signal voltage adjustment, please read this. The maximum input voltage that you can attach to the pin port is **VCC+0.5V**. So, if you supply this device from 3,3V and want to program it with standard 5V programmer – you will need to protect its ISP lines from damage.

Do it in this way – this is simple solution but it will protect your chip.

DO NOT do it in this way – if you connect it just like this, you can damage your chip!