A8Eprog
(An 8-bit MCU Electronic Reader/Programmer)

Hardware and Software Information

Preface

This document describes the hardware, firmware and software specifications of the A8Eprog project. Effort was made for this document to contain all the necessary information without errors in a form that can help the novice user as well as the experienced one. The language used is as simple as possible in order for everyone to understand the functions of the project. All information presented here provided "as is". No responsibility/liability is held true if any error occur by the usage of the data presented in this project. You may use all info for personal use only – as always at your own risk!

Copyright notice

All information presented here might be trademarks or registered trademarks of their respective holders. For more information about components / info mentioned in this document, please refer to the manufacturers. Whenever is possible all trademarks are referred to the footer of each page. All trademarks appear in this document are refereed to as 'registered' trademarks (others are, others might not).

A8Eprog Copyright information

A8Eprog project and all of its accompanied material / documents are copyrighted by
Nikos Bovos © June 2016 except where mentioned copyrights of others.
You might see some other dates on boards and windows, this project started on
March 2016, but because of lack of free time, the first version finished in June 2016
You may not dis-assemble or reverse engineer the provided with this project files
You may use all info for personal non-profit use, for professional use please contact me.
Please always give credit where deserved, I spend many-many hours designing and building the hardware, firmware and software and implementing all these features, when distributing retain the original format, information and files.
The project is designed, tested and work with the accompanied software.
No responsibility/liability by any means is held if you decide to use other kind of software
to command the A8Eprog device.

Contact info, bugs report and other info

If any bugs or errors found, please report them to tano@hol.gr (alternative email is nbovos@gmail.com) in order to be corrected as soon as possible.
For any questions feel free to send me an email, I will reply as soon as possible.
Please note, the construction and usage of data of this project is at your own risk.
Extra attention must be taken, especially when setting the programming voltages
of the various chips (+25V on any low voltage chip will sent it to trash can immediately...)
Please note also that any future updates will be done in software and firmware only,
no circuit modifications will be needed (at least that's my original intention)
Donations via paypal to keep coding to tano@hol.gr would be very much appreciated :-)

This document was written using Apache Open Office™ 4.1.1 on my
Pentium 4 @ 3,2GHz P.C, running Microsoft Windows™ XP Professional©
and exported to PDF via the PDF export function of Open Office.

Usually i'm told that i over-analyze things, I'm afraid I might do it also here, so I apologize in advance for that...

Open Office first build by Sun microsystems & the community (now acquired by Apache Software Foundation)
Apache Open Office is a trademark of Apache Software Foundation
Microsoft, Windows, Windows XP Professional are registered trademarks of Microsoft Corporation

What is A8Eprog ?

A8Eprog is an 8-bit MCU based Electronic Programmer and reader. Starting as CMOS/NMOS EPROM Programmer and reader, but in the future will cover more devices on programming/reading (at least as many as I can add – or use !). As for the starting version, it supports EPROMs with programming Voltage Vpp = 12,75V, EPROMs with Vpp = 21V and Vpp = 25V. Currently A8Eprog has been tested to read and program the following EPROMs from the stated manufacturers (more of them will be added in the future). I assume it can also read/program all others not mentioned here. The voltages shown as 'Vpp' are the programming voltages:

Manufacturer	EPROM Model	Total Size (kb/Mb)	Arrangement (words x bits)	Case	Vpp (V)
ST	M2764AF1	64 kbit	8k x 8bits	DIP-28W	12,75 V
Fairchild	NM27C64Q	64 kbit	8k x 8bits	DIP-28W	12,75 V
AMD	Am27C128	128 kbit	16k x 8bits	DIP-28W	12,75 V
Mitsubishi	M5L27128K	128 kbit	16k x 8bits	DIP-28W	21,00 V
Atmel	AT27C256R	256 kbit	32k x 8bits	DIP-28W	12,75 V
NSC	NM27C256	256 kbit	32k x 8bits	DIP-28W	12,75 V
Intel	D27C256-1	256 kbit	32k x 8bits	DIP-28W	12,75 V
ST	M27C512	512 kbit	64k x 8bits	DIP-28W	12,75 V
Atmel	AT27C512R	512 kbit	64k x 8bits	DIP-28W	12,75 V
TI	27C512-12	512 kbit	64k x 8bits	DIP-28W	12,75 V

Table 1
Tested EPROM chips

Some of the above EPROMs support identification / manufacturer code to be retrieved by the programmer, to set the programming algorithm accordingly. The PC software that accompanies A8Eprog includes an option to instruct the device to identify the EPROM in socket and report the identification bytes. For the tested Eproms shown above (and some others) the manufacturer codes has been already added to the software. Others may be added by the user as well.

Please note that A8EProg currently supports EPROMs that have 8-bit data-bus and internal array arrangement (meaning words of 8 bits, not 1 bit, not 16 bits etc). The current version supports the 27xxx family of EPROMs – In future versions, others will be added (E-EPROMS of 28xx family). Some plans for update to support serial Eproms as well (SPI* and I^{2}C*) - but this will be in a future update and it's highly dependable on my free time...

Please note also that the maximum supported parallel 27xxx family EPROM in this project is 2MBit (256kwords x 8bits - the 27x020 EPROMs).

As for firmware 1.00, the project supports only 2 bytes for chip identification. The first byte is the manufacturer byte and the second the device type byte. Maybe in the future will include all possible implementations (I think they are about 6 bytes now...and counting...)

SPI : Serial Peripheral Interface invented/developed by Motorola
I^{2}C : Inter-Integrated Circuit invented/developed by Philips Semiconductor (now known as NXP semiconductors)

The heart of A8Eprog – the AVR ATmega8515 microcontroller (MCU)

A8Eprog programmer is buit with Atmel's **ATmega8515*** MCU. The micro-controller used is the one in PDIP-40 package presented below (pic from Atmel's datasheet):
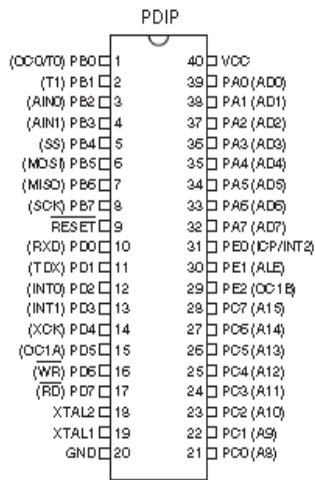
This MCU has the AVR 8-bit RISC architecture*
It has 130 instructions, most executed in one clock cycle
32 general purpose registers x 8 bit each
On-chip 2-cycle multiplier
8 kBytes of In-System programmable flash memory
512 bytes EPROM memory
512 bytes internal SRAM memory
Capability to address up to 64kBytes external SRAM memory
One 8-bit Timer/Counter
One 16-bit Timer/Counter
Programmable serial USART
Up to 16MHz clock
and many other features, check on Atmel's web site this microcontroller http://www.atmel.com/devices/ATMEGA8515.aspx

The selected MCU has 35 Input/Output programmable pins and it was selected due to the fact that it is a micro controller that have used many times in the past and know it fairly well. Also the DIP package is more easy for me to manage than the PLCC or TQFP package that this MCU is available.

Due to the selection of the MCU and because 35 I/O pins are available, there is a maximum limit of the external parallel EPROM that can be addressed (directly without multiplexing). The maximum EPROM chips I currently have available are 512 kBit, so I decided to set the maximum programming capability to 2MBit EPROMs (which are 256kWords x 8 bits). To address a 2MBit Eprom you need 18 bits A0 to A17 (because 2^{18} = 262144 addresses x 8 bit each = 2097152 bit or 2MBit). The data-bus as mentioned earlier is 8-bit wide. Because the USART of the MCU will also be used to connect to PC and exchange data with the PC software, 2 more I/O pins needed (PD0, PD1) for a total of 28 I/O pins.

Most Eproms have 3 control signals, others have 2. These are:

a) Chip Enable ($\overline{CE}$) - JEDEC Pin notation $\overline{E}$
b) Output Enable ($\overline{OE}$) – JEDEC Pin notation $\overline{G}$
c) Program Enable ($\overline{PGM}$) – JEDEC Pin notation $\overline{P}$

So the total pins needed are 31, which leaves 4 I/O pins to distinguish all signals for the various operations and chip-identification for this project, which are not enough if we like to give also some sort of output to the user. To overcome the luck of available pins, one could choose a bigger MCU or multiplex some signals. I didn't want to do it, so I had to insert a second MCU to the project (which is easier than constructing mux/demux with digital circuits and also I have some available). More about this in the following pages...

The port map of ATmega8515 for A8Eprog
The following table displays the port-map of the microcontroller used for the project. I always construct a table like this to have a quick look at the project's data.

PORT A	Function	Direction	PORT B	Function	Direction
PA0	Address bit A8	OUT	PB0	Data bit D0	IN / OUT
PA1	Address bit A9	OUT	PB1	Data bit D1	IN / OUT
PA2	Address bit A10	OUT	PB2	Data bit D2	IN / OUT
PA3	Address bit A11	OUT	PB3	Data bit D3	IN / OUT
PA4	Address bit A12	OUT	PB4	Data bit D4	IN / OUT
PA5	Address bit A13	OUT	PB5	Data bit D5	IN / OUT
PA6	Address bit A14	OUT	PB6	Data bit D6	IN / OUT
PA7	Address bit A15	OUT	PB7	Data bit D7	IN / OUT

PORT C	Function	Direction	PORT D	Function	Direction
PC0	Address bit A0	OUT	PD0	RxD from PC	IN
PC1	Address bit A1	OUT	PD1	TxD to PC	OUT
PC2	Address bit A2	OUT	PD2	Bit0 for State MCU	OUT
PC3	Address bit A3	OUT	PD3	Bit1 for State MCU	OUT
PC4	Address bit A4	OUT	PD4	Bit2 for State MCU	OUT
PC5	Address bit A5	OUT	PD5	Bit3 for State MCU	OUT
PC6	Address bit A6	OUT	PD6	Address bit A16	OUT
PC7	Address bit A7	OUT	PD7	Address bit A17	OUT

PORT E	Function	Direction	Notes		
PE0	Program Enable $\overline{\text{PGM}}$	OUT	The maximum EPROM address is 18 bits (A0-A17), so 2^18 addresses or 256 kwords x8 bits can be addressed. This translates to 2 MBit maximum EPROM program/read capability.		
PE1	Output Enable $\overline{\text{OE}}$	OUT			
PE2	Chip Enable $\overline{\text{CE}}$	OUT			

Table 2 - Microcontroller Port-Map

- One could argue "Why on earth ATmega8515?". First because I have some left available from other projects and second I didn't want a larger mcu for something I don't have (> 2MBits Eproms). If the need arises,something will come up.
- "EPROMs ?" - Seriously man? In 2016 ? - Yes, why not? As this project will be expanded to the electrically erasable (eeproms) also, the same board will be used for the other chips as well. I wanted to cope with the most difficult tasks first, also maybe – maybe - maybe someone still has OTP/UV Eproms (I do have about 20), so if I can help someone else apart from myself, it's fine. I decided to build this project, learn and share the info with all of you. Keep it or leave it !
- The Eprom's control signals will be delivered by PORT E of mcu. Note that not not all signals are available to all EPROMs.

Before analyzing some of the internals, a few notes about the rest of the modules comprising this project...

Multi-Voltage Power Supply Circuits

Before referring to the firmware and software, a few information about the various power supplies. As already seen, multiple voltages are required to be produced, in order to read, program, verify and identify these ICs (and any other in the future). As this project follows my 2 rules : "keep it simple and if it works, leave it simple" and "If it works that way, don't change it" and of course because power consumption is not an issue here (there is no need to use any LDO regulator or other low power components – even low power MCU). The well known, multi-used and flexible LM317* along with some helping circuits used. The following schematic (and all others) designed using the software BSCh3V (c)1997-2014 H.Okada (Suigyodo) and can be found on http://www.suigyodo.com/online/e for the English version. The power supplies used in this project are mainly supplies I have used in the past (and still working), so many of the PCBs were already available (or refurbished !).

The necessary voltages to be produced for all circuits
- 25V and 21V for Vpp (mostly NMOS Eproms / Older CMOS)
- 12,75 V for Vpp (most CMOS Eproms)
- 12 V for Eprom identification (where supported)
- 6,25V for Vcc in programming mode & verification mode
- 5 V for read mode on Vpp pin
- 5V on Vcc pin (in future this will power 5V serial eeproms too.)
- 5V for the MCU (or other logic circuit demands this voltage)
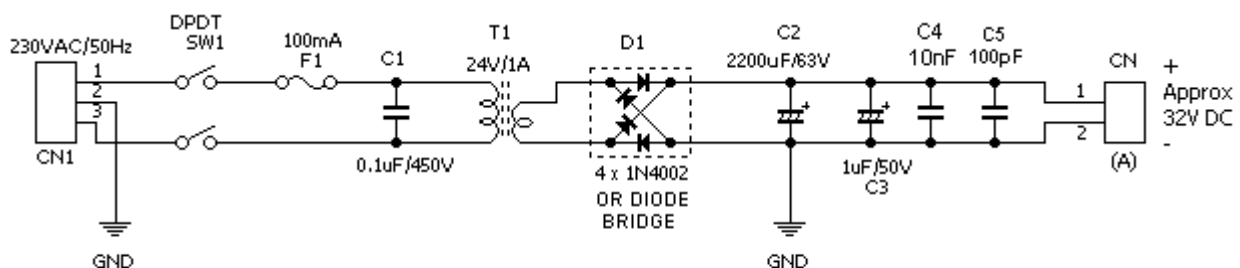
**Main Power Supply and filters circuit
Designed by Nikos Bovos**

Figure 1

Not much to analyze about the above circuit... The mains voltage is stepped down by the transformer to 24 VAC, rectified by the diodes and filtered by the capacitors. At the Output (A) point we have approximately 32Volt DC voltage (rectified 24V AC minus the voltage drop on the diodes), which will be used to feed all the other power supply circuits. Please note you must earth ground the whole device. This is essentially because of the connection and communication with the PC's Serial Port / USB Port which is earth grounded too. The whole project is going to be placed in a metal case (for better EMC behavior). The transformer shown above is for 1A maximum current, but also a 200mA can be used (I had one available and didn't want to buy a new one for the project). The fuse in the mains supply can be a lower value as well to match the maximum current consumption.

LM317 is an adjustable linear voltage regulator first created by National Semiconductor (now acquired by Texas Instruments)

Starting with the 32V DC available after the main power supply, the first (and closest) voltage to produce is the 25 Volt for the programming (Vpp). The following circuit accomplish that:
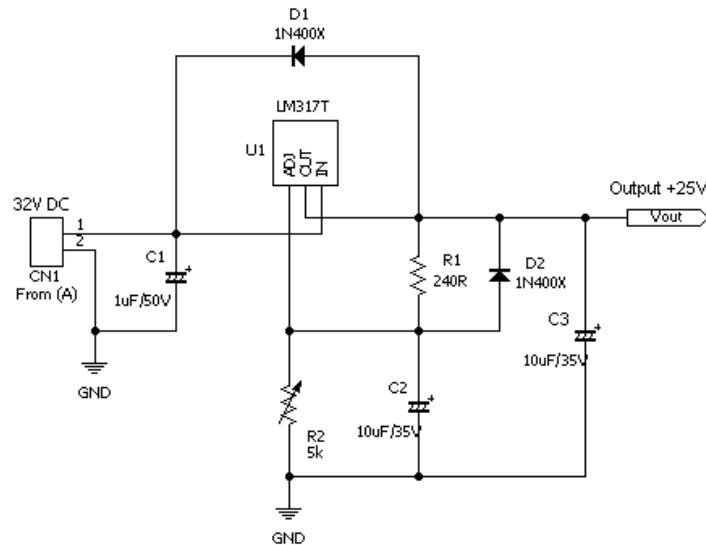
Figure 2
LM317 based power supply to provide + 25V DC output
Designed by Nikos Bovos

A typical LM317 application circuit with added diodes for protection.

As known by the 317 manufacturer, the output voltage is given $\mathbf{V}_{OUT} = \mathbf{V}_{REF} * (1 + \frac{R_2}{R_1})$

(*Because the output voltage is independent of the input voltage, you can (and you should) adjust/tune the power supply alone, without any other boards connected)*. Ignoring the current flow through the adjust pin (I_{ADJ}), which is held true given the fact that at a maximum this current equals 100uA and so the largest voltage error will be negligible. Resistor R_1 is usually 240 Ohm and R_2 can be in a value that the IC will produce the desired output voltage. V_{REF} is typical 1,25 Volt (max. 1,30V) and it is the internal voltage reference of the IC. Solving the above equation for **R_2 = 4560 Ω** the necessary value. For better accuracy the voltage on the output was measured with a digital voltmeter while fine tuning R_2 (which is multi-turn trimmer resistor, specifically a **Bourns 3296 5kΩ Trimming potentiometer**). The current flow through the adjust pin of the 317 is max at 100uA, so for the worst case scenario and the resistor value, the voltage difference is very small (about 0,38 Volt which does not affect any of project's parameters). But because the output was measured while trimming the potentiometer, a stable +25 V appears on the output.

The voltage difference between input and output is approximately 7 Volt DC and given the fact that the current consumption of the Eproms is at maximum 100mA (for old NMOS Eproms, CMOS have much lower), a small heat sink is required for this 317 (you can place it on a larger one if you like) because this circuit will feed all other power supplies, so this 317 will provide the total current of the whole project (which is not very large, approximately 250mA in the very worst case which is low for this regulator). D_2 and D_1 diodes protect the 317 IC by output shorts and discharging of capacitive loads connected. The 317 preferred type is in **TO-220** case (maximum current capability in excess of 1,5A)

After the 25 V output, a similar circuit to produce 21 V output:
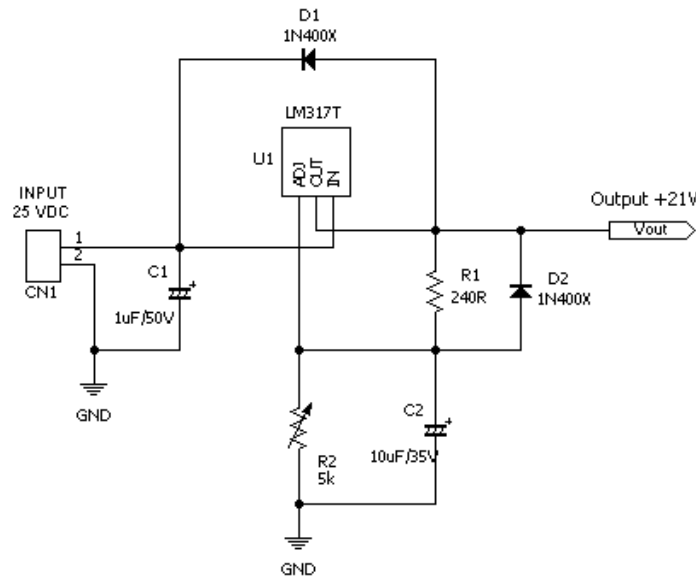
Figure 3
LM317 based power supply to provide +21V DC output
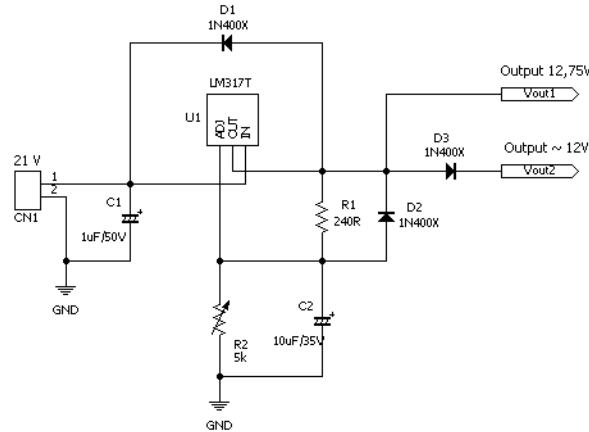**Designed by Nikos Bovos**

The voltage difference between input and output is approximately 4 Volts DC and given the same facts as before, a small heat sink may required for this 317 also. The calculated value for the **R_2 = 3800 Ω** and again the output was measured while trimming the R_2

To produce the 'standard' voltages of +5 Volt and +12 Volt, a similar circuit was built (please note that we will use three power lines of 5Volt. One is to be always available and will power the microcontroller circuits, one will be selectable for Vcc and one for Vpp of the Eprom ICs). Usually in these situations, the LM7812 was my main choice, but I have noticed a tendency to drop the output voltage (a little) with some not so heavy loading, so I decided to use also a '317' to produce the +12 Volt for the Eprom identification and any other use in future versions.

One could use the proposed applicatrion circuit (in 317 datasheet) for electronic selection of the output voltage and use one '317' circuit, I didn't do it and built the circuits presented (some were available also from older projects)

The following circuit is the same as the one in figure 3 above, the only difference is the addition of diode D3 and the value of the resistor R2 (calculated approximately 2200 Ω). The output voltage was set on +12,75 Volt approx. and a diode connected , to introduce voltage drop of about 0,7V, so about +12 V appear on the other output. The input voltage of the circuit is the output of the previous one, so the 317 will have a voltage difference of about 21 – 12.75 = 8,25V. Again a small heatsink will make sure that the circuit will operate for a very long time without any problems.

Figure 3
317 based power supply for 12,75V and 12 Volt
Designed by Nikos Bovos

From the output of the above circuit (either +12,75 or from the +12V line), the well-known 7805 was used to produce the 5Volts for the microcontroller and the Vcc and Vpp lines. The typical 7805 circuit constructed with added protection...A small heat-sink will make your power supplies work for ever without problems...
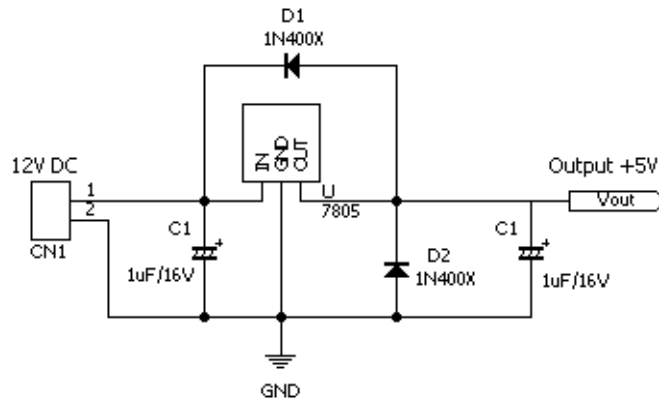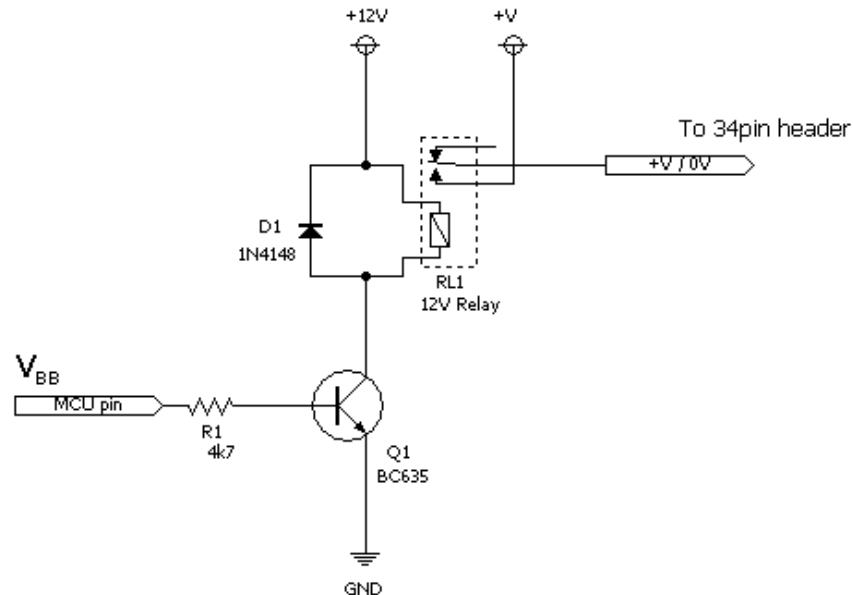
Figure 4
7805 based 5Volt Power Supply
Designed by Nikos Bovos

Three power lines were drawn from output of +5V supply as mentioned earlier. 7805 can provide more than 1A output current which is more than enough for all usage.

Multi-Voltage Selectors

This is a critical task because a mistake will send the entire EPROM (and maybe the MCU also) in tears and later in the trash can !!! The separation was made using the good-old friend – the relay. Because timing in voltages is not critical (but the correct sequence is) the following circuit was used, to ensure 100% isolation of the output (an opto-isolator could also be used rated for the necessary voltages to switch)

Figure 5
Voltage Seperator/Selector Circuit

A few words for the theory and operation of the above circuit... Every relay is rated for voltage and current and some times the relay resistor is given – if not you can measure it. My relay has a (measured) resistance of **R_{RL} = 265 Ω** approximately and it's rated for **V=12V DC**. So the relay current will be **I_{RELAY} = 12V/265Ω = 45mA**. The manufacturer of the relay gives 44mA current rating, which is about the same as in calculations. To be sure that the transistor will be able to drive the relay (guaranteed saturation), read from the BJT datasheet the worst case DC beta gain (h_{FE}) for the current you will need. For the **BC635** Bipolar Junction Transistor i'll use the worst case h_{FE} which is about 50. The necessary base current is calculated by the fact that I_C = I_{RELAY} and

I_B = I_C / h_{FE}, so I_B = 45mA / 50 = 90µA for the base current and so $$R_1 = \frac{V_{BB} - V_{BE}}{I_B}$$

whereas V_{BB} = 5 Volt from "state MCU" pin, V_{BE} = 0,65V approx for the BJT B-E junction. (the BC635 has a max.V_{BE} = 1V for I_C = 500mA) and so the required resistor R_1 = 4833 Ω. We will choose a smaller value I.e **R_1 = 4700 Ω** to ensure guaranteed saturation of the BJT. The circuit of figure 5 was used for all the required voltages, <u>except</u> the selector circuit for the automatic identification of EPROM, which needs +12V on **A9** pin of EPROM. As seen above the output voltage is either +V or floating. For the automatic identification 12V DC on A9 line is needed and in normal operation the A9 line can either be +5V or +0V (logic one or logic zero). The above circuit changed on the output side only, having A9 line from MCU (on N.C contact) and +12V for the automatic identification (on N.O contact), instead of +V and floating as shown above. The diode is included to protect the transistor from the realy's coil kickback current when releasing. Smaller transistors than BC635 can be used also, I had plenty of them and used them.

The State MCU (ATtiny2313*)

The inclusion of a second microcontroller as mentioned earlier was a necessity because the available I/O pins of ATmega8515 were not enough for all functions I wanted to implement.

As shown on table 2, there are 4 I/O pins dedicated to the "State MCU". These four bits can be translated to 16 possible combinations. The following operations were implemented to the ATtiny2313:

- "Safe to remove" Led when EPROM socket has no voltage
- "Don't touch it !" Led when performing some action (read/write)
- "Reading" Led when reading is performed
- "Programming" Led when in programming phase.
- etc etc etc any other desired state (i.e failed chip, verifying etc)

For the implementation of the "state MCU" the **Atmel ATtiny2313** was selected (in PDIP-20 package) because it was used in many of my past projects and never failed me (and is still working in active projects without problems) and I have many more of that MCU currently available – more than any other MCU I use. The main difference from it's predecessor AT90S2313 (which is still running in some projects) is the internal clock capability, the alternative use of $\overline{\text{RESET}}$ pin and as always with newer devices, the "updated" construction and implementation of new features and correction of bugs found on previous implementations. The pinout of the mcu is presented in the following picture, as provided by Atmel's datasheet:
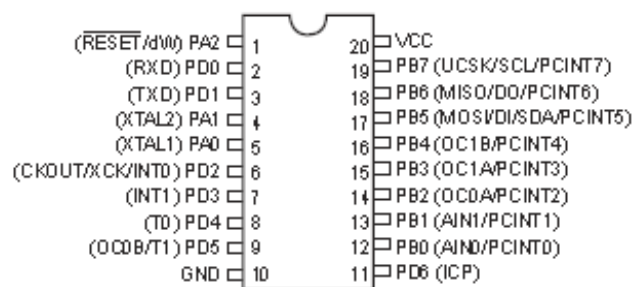
**ATtiny2313 microcontroller
(PDIP/SOIC pinout)**

You can check the mcu at http://www.atmel.com/devices/ATTINY2313.aspx

In the following table, the port-mapping for this MCU is presented along with a few notes about it's operation. All pins configured as inputs on ATtiny2313, have the internal pull-up resistors enabled (by setting DDRD = 0 and PORTDx = 1) which is not a critical thing, it can work without them also.

ATtiny2313 is an 8bit AVR micro-controller by Atmel Corporation

Port-map of ATtiny2313

PORT A	Function	Direction	PORT B	Function	Direction
PA0	"Reading" LED	OUT	PB0	Select 6V25 for Vcc	OUT
PA1	"Programming" LED	OUT	PB1	Select 5V for Vcc	OUT
PA2	Used as $\overline{\text{RESET}}$	IN	PB2	Select 5V for Vpp	OUT
			PB3	Select A9 / 12V line	OUT
			PB4	Select 12V75 for Vpp	OUT
			PB5	Select 21V for Vpp	OUT
			PB6	Select 25V for Vpp	OUT
			PB7	*Not used*	OUT

PORT D	Function	Direction		
PD0	"Safe" LED	OUT		
PD1	"Don't touch" LED	OUT		PORT A is 3-bits port
PD2	Bit0 from ATmega8515	IN		
PD3	Bit1 from ATmega8515	IN		PORT D is 7-bits port
PD4	Bit2 from ATmega8515	IN		
PD5	Bit3 from ATmega8515	IN		PORT B is 8-bits port
PD6	*Not used*	OUT		PA2/$\overline{\text{RESET}}$ is configured for Reset operation

The internal 1MHz clock of ATtiny2313 was selected as the clock source (at 25oC) as timing is not critical for this MCU. This ATtiny2313 reads the input voltages of PD2 – PD5 pins and according to the table that will be presented below, it will select the appropriate signals. Because 4 bits are used, there are 16 possible combinations. In the current implementation, not all of these combinations have been used, some of them refer to future expansion of the project or will not be implemented at all.

The default state for this microcontroller is to have all available voltages deselected and so no voltage appear in the chip socket (this is referred as "Safe to remove" state, in which you can insert or extract the inserted EPROM), remove the board etc. The selection circuit was presented in **figure 5** in the previous pages. A logic high (+5V) will drive the base of the BC635 transistor through the 4k7 resistor and this will create the collector current which will "latch" the selected relay and so the Common Pin will connect to "Normally Open" pin of the relay. A logic low (0V) will remove the base current, thus the collector current and have the relay unlatched and so anything in the "Normally Close" contact of the relay will appear in the output. All of the selection circuits except A9/12V have the N.C contact floating and so no voltage or other signal appear on the EPROM socket when all outputs are low.. This is done because all relay outputs are wired together on the power pins, so it wasn't possible to have any of the Normally Open contacts to ground (because that would cause a short circuit when another relay was latched). This has the effect that a small "floating" signal might appear on '0' state, but so far there were no problems on testing. All led diodes were connected through a **220 Ω** resistor in series for current approximately 13mA (can be omitted, the MCU port can drive leds directly, but a good practice is to use them), you can change this value to a smaller one for more current (calculate for the led diodes you have). Green Led diodes where used (approx 1,5 – 2 V dropout voltage) so (5V – Dropout/(Led current) equals the resistance.

Table of operations & firmware settings of state MCU

The following table shows the different states that implemented. One (1) mean +5V and zero (0) means 0 Volt input on any of the PD5 – PD2 pins of ATtiny2313 MCU.

| PD5	PD4	PD3	PD2	Implemented function / Operation to be performed
0	0	0	0	Vpp = 0V , Vcc = 0V , A9 line = A9 (Safe mode)
0	0	0	1	Vpp = 5V , Vcc = 5V , A9 line = A9 (Read)
0	0	1	0	Vpp = 5V , Vcc = 5V , A9 line = +12 V (Auto-ID)
0	0	1	1	Vpp = 12,75V , Vcc = 5V , A9 line = A9 (Programming)
0	1	0	0	Vpp = 12,75V , Vcc = 6,25V , A9 line = A9 (Programming)
0	1	0	1	Vpp = 21V , Vcc = 6,25V , A9 line = A9 (Programming)
0	1	1	0	Vpp = 21V , Vcc = 5V , A9 line = A9 (Programming)
0	1	1	1	Vpp = 25V , Vcc = 6,25V , A9 line = A9 (Programming)
1	0	0	0	Vpp = 25V , Vcc = 5V , A9 line = A9 (Programming)
1	0	0	1	$\overline{OE}$/Vpp = 0V , Vcc = 5V , A9 line = A9 (Read for 27X32 & 27X512)
1	0	1	0	$\overline{OE}$/Vpp = 0V , Vcc = 5V , A9 line = +12 V for Auto-ID (for 27X32 and 27X512)
1	0	1	1	*Reserved for future use*
1	1	0	0	*Reserved for future use*
1	1	0	1	**Leds only/no voltage changes:** It's safe to touch and remove the chip
1	1	1	0	**Leds only/no voltage changes:** Don't touch it, i'm reading
1	1	1	1	**Leds only/no voltage changes:** Don't touch it, i'm programming

The reserved combinations are currently ignored by the state MCU and probably will be used (or not) in future expansions of this project. The near future plans are for 5V Serial I^2C EEproms reading and programming and 5V Serial SPI EEproms (State = 0001 in the above table) it will highly depend on the (not much) free time... Some thoughts about a high-voltage parallel programmer for AVR exist but it's really in preliminary state... Also 3,3V devices are examined but still is too early to predict the future...

The firmware of ATtiny2313 was written assembly. The main routine scans the port pins PD2 through PD5 of PORT D and according to the previous table will signal the voltage selectors and the A9/12V selection circuit and the appropriate leds for visual information to user. Upon reset the default state is all PORT B bits to zero (PB7 is not used in this project but for compatibility with future versions it is written to zero when using all port bits of PORT B and not used in individual bit setting).

The clock source as mentioned is the Internal 1MHz clock@25oC. This is selected by programming the CKSEL fuses values 0100 (thus selecting 8MHz internal clock) and then program the CKDIV8 fuse for divide by 8, resulting in 1MHz clock. The programming of the fuses is performed by the programmer as with any other AVR (it is the default setting for all new ATtiny2313 produced, so you don't have to program any fuse, except if fuses had been programmed for other usage before).

State MCU circuit schematic

The following schematic shows the state MCU along with the signals produced.
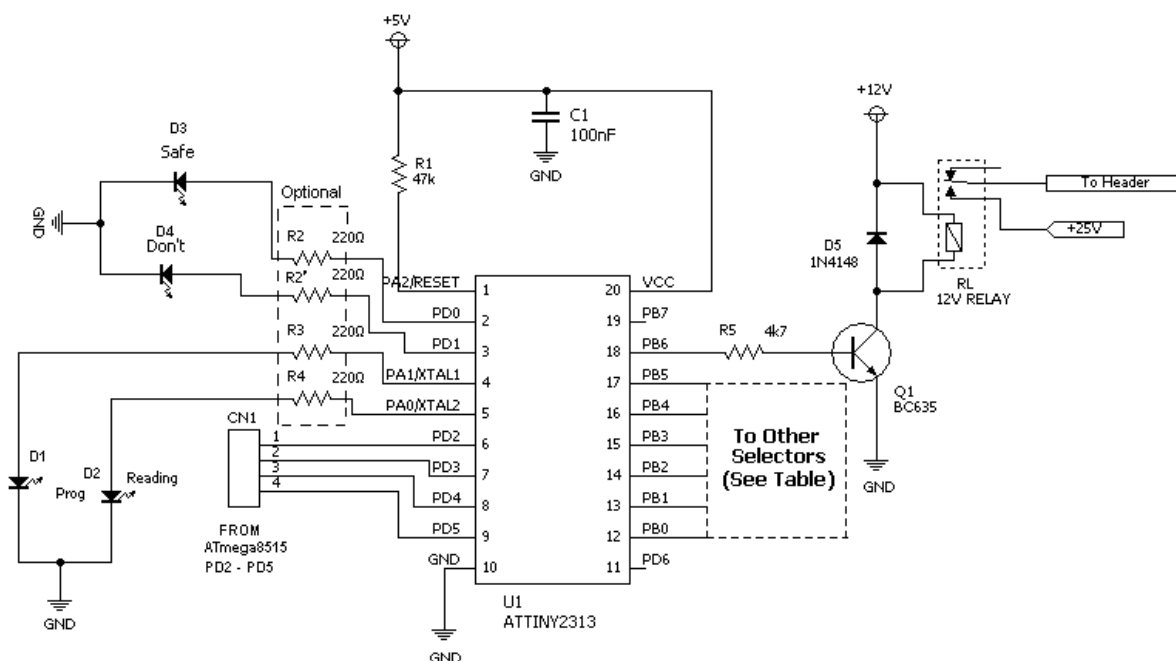
Figure 6
State MCU schematic
Designed by Nikos Bovos

The "Other Selectors" Block consist of identical R, BJT, DIODE, RELAY components connected to the appropriate PORTB pins (see the port-map of state MCU for details) and having the voltages referred to Normally Open pins of each relay. Don't ask for full schematic, I didn't made one !

PD2...PD5 of ATtiny2313 were connected to PD2...PD5 of ATmega8515 (It's a coincidence that the same port pins used on both MCUs, any other pins could function the same) and receive the desired "state" from the main mcu.

Maybe the relay circuit is not the most efficient, it was a quick choice, maybe in the future will be updated with something more flexible (or not, if no problems arise. If it is working good, leave it that way...). Also as mentioned earlier, by using the relays the Vpp pin must be left floating when 0V is desired (or I had to add one more of these blocks for PB7). Until now works fine without it.

State MCU main routine

The main routine scans the input pins and according to the values, perform the desired operation. The scanning of the pins is performed every 10ms. The delay routine to accomplish the 10ms delay is presented below. For better view a fixed length font is used:

```
;****************************************************************
;* Delay Routine By TDC_AVR 0.02 By Nikos Bovos
;* XTAL/Res Frequency : 1 MHz
;* AVR Clock Cycle    : 1 usec
;* Delay Produced     : 10000 Clock Cycles
;* Equivalent Time    : 10 msec
;****************************************************************
delay10ms:
    ldi     medium,17       ; Load Medium Counter ($11)
lf10ms:
    ldi     fine,195        ; Load Fine Counter ($C3)
std10ms:
    dec     fine            ; Decrease Fine Counter
    brne    std10ms         ; If fine Not Zero repeat
    dec     medium          ; fine = 0, Decrease Medium
    brne    lf10ms          ; If Medium >0, reload fine
    ret                     ; Return From Subroutine
```

A standard delay routine using registers. The software to produce the above listing is an old program of mine (tdc_avr) built way back in 2002. I'm sure that many similar software from others exist also...If you count the clock cycles for every instruction you can calculate for any delay you want. In the above routine, **medium** and **fine** are high registers (r16 - r31) declared with .def directive in the assembly listing (high registers only support the **ldi** instruction)

After calling the above delay routine, the mcu gets the input value of pins of PORTD and branching to the appropriate routine to perform the needed action. Because as you already know assembly listings consumes many lines for each operation, only a portion of the listing will be shown here, along with some defined macros to perform the basic operations. **genio** register has been defined as **r16** (with .def genio = r16 in the .asm file).

```
    . . .      (previous commands)
    . . .      (previous commands)
Mainscan:
    rcall   delay10ms       ;Small delay before check
    in      genio, PIND     ;Read value of pins in port D
    andi    genio, $3C      ;Isolate PD5-PD2 bits on genio
    cpi     genio, 0        ;Is=0? (genio=00XXXX00 after andi)
    breq    Sub_Zero        ;Branch to Sub_Zero if genio=0
    cpi     genio, 4        ;Is=4 ? (00000100)
    breq    Sub_V5V5        ;If=4 branch to Sub_V5V5
    cpi     genio, 8        ;Compare to 8 (00001000)
    breq    Sub_AutoID      ;If genio=8 branch to Sub_AutoID

    . . .   . . .               (next compares and branches)

    rjmp    Mainscan        ;Jump back and repeat
```

Some macros were used also. In AVR assembly are declared with **.MACRO** and **.ENDMACRO** directives. For more information refer to Atmel's help.

```
;***** Macro for Zero State – all outputs low immediately
.MACRO ZeroState
    clr     genio               ;Clear genio
    out     PORTB, genio        ;All bits of PORTB = 0
.ENDMACRO


;***** Remove voltages after completing operation – Vcc last
.MACRO RemoveAll
    cbi     PORTB, Vpp25        ;Vpp25(PB6) low (remove Vpp if any)
    cbi     PORTB, Vpp21        ;Vpp21 low
    cbi     PORTB, Vpp12        ;Vpp12 low
    cbi     PORTB, Vpp5         ;Vpp5  low
    rcall   delay1ms            ;Delay 1ms to remove Vcc
    cbi     PORTB, Vcc6         ;Vcc6 low (remove Vcc if any)
    cbi     PORTB, Vcc5         ;Vcc5 low (remove Vcc if any)
.ENDMACRO
```

and many others that is out of purpose to be shown here. The assembled program is about 295 words (approximately 14% of total program space) and so has enough program space for any future expansion.

The Vpp25 and others that appear in the listing were defined to the mcu port pins with the .equ directive (.equ Vpp25 = PORTB6 , .equ Vpp21 = PORTB5 etc).

The main MCU - ATmega8515

ATmega8515 was programmed in Assembly also. The assembly chosen because it's been a while since my last assembly program and essentially this project was also an assembly-programming refreshing for me because I was (I am) a little rusty in assembly. Unfortunately all information available on the internet about AVR assembly is mostly for simple tasks, so I had to 're-invent the wheel' by trying and implementing all these functions by myself in assembly (I had done this in the past for 1-Wire communication and I²C communication, so i'm used to struggle with myself...). I must however give credit to Atmel for their easy-to-read documentation, AVRFreaks.net for the excellent site and information and avrbeginners.net as these three were my destinations when I started exploring AVR mcus a long time ago...

The microcontroller has to perform several tasks. Receive commands, apply the correct algorithm while retrieving data from PC in the USART, monitoring address,etc, etc. Some operations are performed using interrupts, while others (the sequence of state and mode selection) are not. Because MCU have very small internal memory in comparison to the memory it has to program, the programming is accomplished in "packets".

The serial communication between the MCU and the PC is done at **38400bps** with **8 data bits**, **1 stop bit** and **no parity.** The MCU is connected to a **D9-USB-D5-F module*** You can omit that module and construct the "classic" MAX232* circuit and the RS-232 female connector. My PC has a serial port, but I wanted to test this module also that I purchased some time ago...The drivers of the D9-USB-D5-F module is available by the manufacturer of that module "Future Technology Devices International Ltd" on the web address http://www.ftdichip.com/FTDrivers.htm Of course a bigger communication speed could be selected (by choosing different crystal and calculating the necessary values for UBRR register), didn't bother to do so.... Data are send and received to the computer so it's obvious that the total time will be larger compared to when the MCU is performing some action without communication, i.e a blank check is much faster because the MCU is checking every EPROM address and only reports to the computer if a non-blank address found (or if the check is finished and the EPROM found blank). On the other hand a whole chip read will read and report the bytes, so consuming more time. The following table shows some of the times for various chips.

Chip in target board	Action performed	Time	Notes on action performed
ST M2764AFI	Whole chip read	~ 4 sec	Whole-chip read (after set-up)
Am27C128-155DC	Whole chip read	~ 11 sec	Whole-chip read (after set-up)
ST M27C512-20XFI	Whole chip read	~ 50 sec	Whole-chip read (after set-up)
ST M27C512-20XFI	Chip Identification	~ 2 sec	With the "Technical delay"

As I have already state and you might already have noticed, this project is <u>not a gang programmer</u> for mass production, but a home-project for fun and knowledge and of course for use! If you want something faster/more flexible (and with more chips support) there are definitely many-many very good commercial chip programmers to look for...

D9-USB-D5-F module is a UART converter designed by Future Technology Devices International Ltd
MAX232 is a TIA-232 to TTL converter IC first created by Maxim Integrated Products

The schematic of the main MCU Board

The following schematic design shows the ATmega8515 microcontroller along with the signals get/produced.
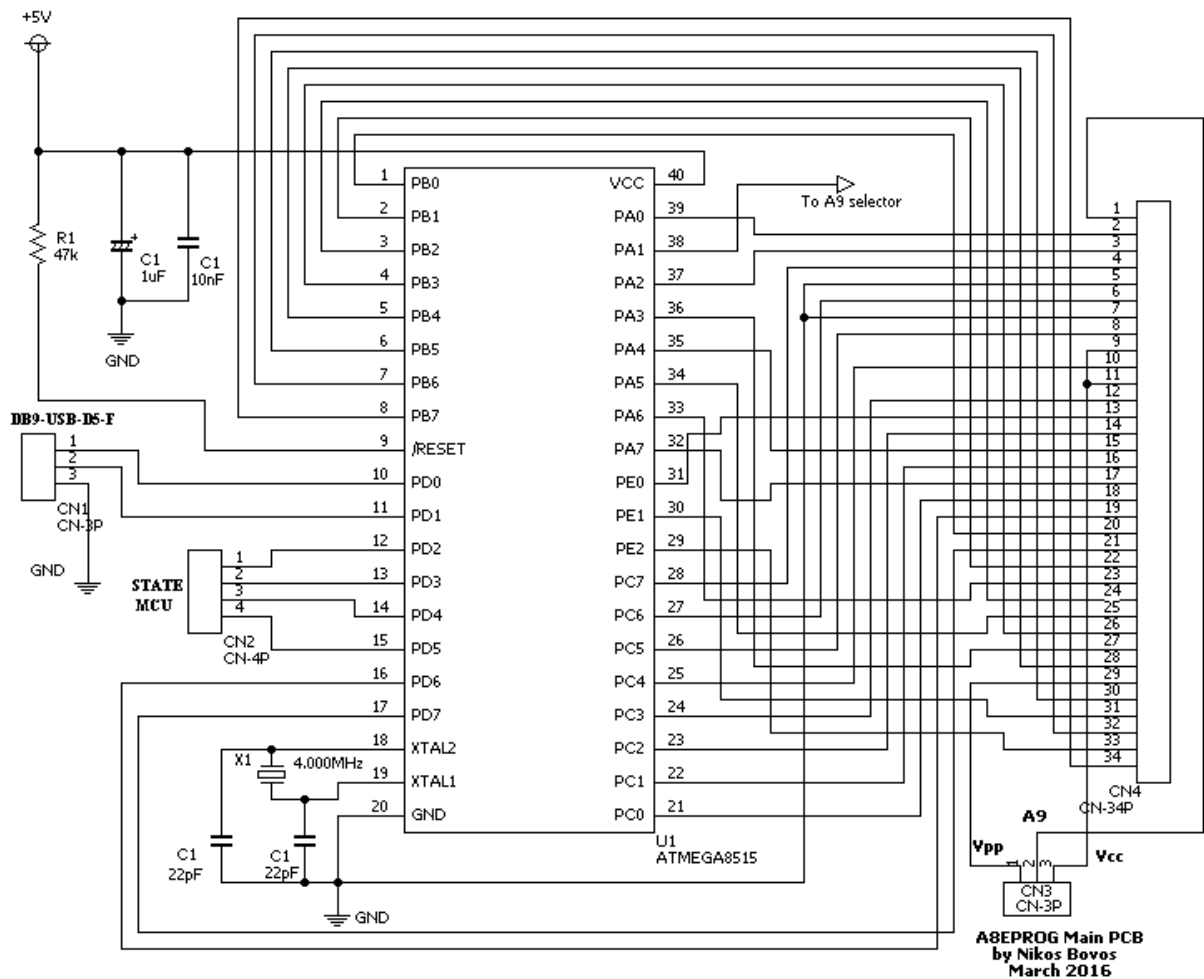
Not much to analyze, only that the three critical signals (Vpp, A9 signal line and Vcc) arrive at connector CN3 from the ATtiny2313 PCB. The crystal used is 4.000MHz. The RESET line is connected through a 47kΩ pull-up resistor as suggested by Atmel and there is no ISP header in the main PCB to program the MCU in-circuit.

The port pins PD0 and PD1 (the USART) along with GND are connected through a 3-pin connector to the DB9-USB-D5-F module for communication with the PC (mini USB).

The port pins PD2,3,4,5 are sent to ATtiny2313 MCU according to the states that have been presented in the table of operations of the state MCU earlier.

The 34-pin header is guided to the case, where any "target" PCB can be connected. In future versions a smaller-pin header will be connected to this as well, for the serial eeproms (already tested and worked, but this will come in future updates, one operation at a time is better than all together...There is also the software that needs the necessary update to support the newer functions and didn't have the time to build it so far).

Pinout of the 34pin connector & signals of supported EPROM ICs

Pin #	MB Signal	27C16	27C32	27C64	27C128	27C256	27C512	27C010	27C020
1	A9	A9	A9	A9	A9	A9	A9	A9	A9
2	A8	A8	A8	A8	A8	A8	A8	A8	A8
3	A10	A10	A10	A10	A10	A10	A10	A10	A10
4	A7	A7	A7	A7	A7	A7	A7	A7	A7
5	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd
6	A6	A6	A6	A6	A6	A6	A6	A6	A6
7	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd
8	A5	A5	A5	A5	A5	A5	A5	A5	A5
9	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc
10	A4	A4	A4	A4	A4	A4	A4	A4	A4
11	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc	Vcc
12	A3	A3	A3	A3	A3	A3	A3	A3	A3
13	$\overline{\text{PGM}}$	- - -	- - -	$\overline{\text{PGM}}$	$\overline{\text{PGM}}$	- - -	- - -	$\overline{\text{PGM}}$	$\overline{\text{PGM}}$
14	A2	A2	A2	A2	A2	A2	A2	A2	A2
15	A12	- - -	- - -	A12	A12	A12	A12	A12	A12
16	A1	A1	A1	A1	A1	A1	A1	A1	A1
17	A15	- - -	- - -	- - -	- - -	- - -	A15	A15	A15
18	A0	A0	A0	A0	A0	A0	A0	A0	A0
19	A16	- - -	- - -	- - -	- - -	- - -	- - -	A16	A16
20	D0	D0	D0	D0	D0	D0	D0	D0	D0
21	A17	- - -	- - -	- - -	- - -	- - -	- - -	- - -	A17
22	D1	D1	D1	D1	D1	D1	D1	D1	D1
23	A14	- - -	- - -	- - -	- - -	A14	A14	A14	A14
24	D2	D2	D2	D2	D2	D2	D2	D2	D2
25	A13	- - -	- - -	- - -	A13	A13	A13	A13	A13
26	D3	D3	D3	D3	D3	D3	D3	D3	D3
27	A11	- - -	A11	A11	A11	A11	A11	A11	A11
28	D4	D4	D4	D4	D4	D4	D4	D4	D4
29	Vpp	Vpp	Vpp	Vpp	Vpp	Vpp	Vpp	Vpp	Vpp
30	D5	D5	D5	D5	D5	D5	D5	D5	D5
31	$\overline{\text{OE}}$	$\overline{\text{OE}}$	- - -	$\overline{\text{OE}}$	$\overline{\text{OE}}$	$\overline{\text{OE}}$	- - -	$\overline{\text{OE}}$	$\overline{\text{OE}}$
32	D6	D6	D6	D6	D6	D6	D6	D6	D6
33	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$	$\overline{\text{CE}}$
34	D7	D7	D7	D7	D7	D7	D7	D7	D7

- The main board produces all signals and connect them to 34-pin connector.
- Every target board must comply with the above pin-out.
- Not all signals are available in every EPROM (obviously)
- Pins marked as - - - are No Connect (N.C) on target boards
- 2732 and 27512 have $\overline{\text{OE}}$ / Vpp on the same pin. Pin 31 is No Connect.
- $\overline{\text{PGM}}$ pin is not available in all EPROMs
- 27020 uses all 34 pins (is the maximum EPROM supported for firmware 1.00)

The 34-pin connector pinout (re-arranged view)

Pin #	Signal	Pin #	Signal
1	**A9**	2	**A8**
3	**A10**	4	**A7**
5	**Gnd**	6	**A6**
7	**Gnd**	8	**A5**
9	**Vcc**	10	**A4**
11	**Vcc**	12	**A3**
13	**$\overline{PGM}$**	14	**A2**
15	**A12**	16	**A1**
17	**A15**	18	**A0**
19	**A16**	20	**D0**
21	**A17**	22	**D1**
23	**A14**	24	**D2**
25	**A13**	26	**D3**
27	**A11**	28	**D4**
29	**Vpp**	30	**D5**
31	**$\overline{OE}$**	32	**D6**
33	**$\overline{CE}$**	34	**D7**

Actual connector used on PCB

- It is the same pinout as in previous table, just the pins are re-arranged as exist in the connector (17 pins X 2 rows) for better view.
- For the pairing of pins between MCU-connector, refer to ATmega8515 port map
- Vpp, Vcc, A9 values selected on the ATtiny2313 MCU board.

The supported commands & flow of MCU operations

After the initialization process ($\overline{\text{RESET}}$ interrupt), the ATmega8515 enters the "idle" state, waiting for the commands from the PC software for the task to perform (and wastes its time waiting...) The state of operations is presented below (at least most of them). The MCU receives the various commands from the PC-software using the USART interrupt and checks the received commands against the table of supported commands. When a valid command is received, it performs the actions described in the table below and/or acknowledges the command received or processed.

All commands begin with the **!** character (ASCII 33) and terminated with the **CR** character (ASCII 13). The MCU acknowledges most commands by **!OK** followed by **CR** character, or by the **!NO** or some others with **!ERR** response (i.e on blank check if chip is not blank). Other responds noted in the table below can be sent by the MCU as well.

- a) "Idle mode", wait for command (quit or chip selection command)
    1) If chip selection command received, set pointers, send !OK and goto (b)
    2) If quit command received (software closes) reset pointers, stay "Idle"

- b) "Waiting mode", Wait command from PC
    1) If quit command received reset settings and return to (a)
    2) If system command received execute it & return for next command.

- c) "Action mode", covers the execution of the various commands.
    1) As with all modes, quit command reset settings and return to (a)
    2) Depending on command received, the MCU executes and return to (b)/(a)

Programming mode is entered when the command **!PRO** is received. This command is sent after the **!SMC** command, in order for the MCU to select the appropriate voltages and apply them to the selected EPROM. Obviously all addresses in buffer that contain **FF** value are not programmed to the chip (FF is the default value after a chip-erase), but send by the computer in the whole-chip programming, so for the MCU to increase the address counters accordingly.

After entering "programming mode", the software sends the data bytes to the MCU. For better quality and stability, the MCU responds with the response **!OK** if it receives the data bytes correct, or send **!NO** if something went wrong (parity error, frame error or data overrun). This have the effect that the total programming time is lengthened, but confirms that the correct data bytes are received and programmed to the chip.

After successful programming, a verification is performed and the MCU returns to "Safe to Remove" state so you can insert another chip and remove the current one, or change cables, target PCBs etc.

Table of commands supported on firmware 1.00
More will be added as the project evolves.

Command	Description – Action to perform – Useful info	MCU Respond
!A8E	A8Eprog Master check (if MCU is 'listening' to the software)	!OK
!AIB	Send by MCU with the 2 ID bytes of EPROM identification	*Send by MCU*
!AID	Instruct the MCU to perform Auto-ID of EPROM in socket	Bytes / !ERR
!BLK	Instruct the MCU to perform a Blank-Check of chip in socket	!OK / !ERR
!DAT	Data bytes read from EPROM	*Send by MCU*
!END	Send by MCU when End of chip reached on whole-chip read	*Send by MCU*
!EPx	x=1 to 8 (8 possible parallel Eprom chip selection)	!OK
!ERR	Send by the MCU if 'error' occurred in any operation	*Send by MCU*
!PAx	Select Programming Algorithm x (see table below)	!OK
!PRO	Program address with supplied data byte	!OK / !ERR
!QUI	Software quits (semi-implemented into MCU)	*No response*
!RDC	Read Whole Chip in socket	*Read chip*
!RDO	Read one specific address and report the byte to PC	!DATxx
!SMC	Inform State MCU for Vpp, Vcc, A9 settings	!OK

The MCU checks only for the above commands. Other commands are ignored instead of negative acknowledged (this mean that no !NO or !ERR is sent by MCU if any command is not recognized). Generally, commands not implemented are ignored. In the assembly file of ATmega8515 (and so the .hex file) there are more commands declared and defined, but not yet implemented (these were put there during the testing period for debugging purposes and are disabled). The total size of the assembled code is approximately 3330 words (about 40% of mcu capacity), very good for what is already inside the ATmega8515 (which has 8kbytes – or 8192 words of program memory in a 4k x 16bit arrangement).

Because many different chips exist from various manufacturers and every one has it's own specifications for programming/verifying etc, to be more sure that you can program any supported chip, the following programming algorithms implemented. Others not mentioned, can be implemented in future versions if you let me know about them !

Command	Programming algorithms and details
!PA1	Standard 50ms programming pulse (NMOS EPROMS - mainly 2716 and 2732)
!PA2	1ms programming pulse (NMOS compatible to Int$_e$ligent Algorithm by Intel)
!PA3	100us programming pulse (mostly used) - for CMOS EPROMs
!PA4	500us programming pulse (found on some NSC's CMOS Eproms)

Regarding the programming algorithms for EPROM ICs, I didn't find information about first implementations of each one, I assume each manufacturer followed a standard while maintained compatibility with others with small variations, so to be fare with all manufacturers, I will mention all the programming algorithms that searched, read and implemented (alphabetically by manufacturer), most of them are mostly the same with few differences (for different manufacturer, mainly in the verification stage and the number of retries and pulse width).

Manufacturer	Programming algorithms (™ + © of manufacturers)
Advanced Micro Devices	Flashrite Algorithm
Atmel	Rapid Programming Algorithm
Intel	Standard, Int$_e$ligent and Quick Programming Algorithms
National Semiconductor	Fast and Interactive Programming Algorithm
SGS Thomson/ST micro	PRESTO IIB and Fast Programming Algorithm
Texas Instruments	SNAP! Pulse Programming Algorithm

In the next pages i gathered the Bill Of Materials (at least of components that I could find a recent price, because most of the components were purchased a long time ago for other projects. Some photos of the prototype construction and some operation pictures will be shown as well. All PCBs are single sided and created with the toner transfer method (and some older ones by hand drawing with permanent ink marker). The dimensions shown are of the actual boards - as shown in the pictures (width X height in centimeters – if you work in inches, 1 inch = 2,54cm). The software that comes in the same package as this document is presented in a few pages as well.

I think, as I wrote in the beginning of this document, that i over-present the project and the various stages, but that's my way...

Two Intel-Hex files are provided, the **a8eprog.hex** is to program the ATmega8515 mcu and the file **state.hex** the ATtiny2313 mcu. The ATmega8515 must be configured for external crystal clock (or you can use it's internal clock and don't place the crystal/capacitors network, I haven't test it) and the ATtiny2313 for 1MHz internal clock (the default of new devices). As already stated, the presented material is intended for personal non-profit use only. For professional use you must contact me (because I have a family to support !!!).

I would like to thank you for your patience, baring and reading up to here... If by reading these pages any idea triggered your brain about your next project, I would be very glad to know and help you - if I can of course !

A8Eprog 1.00 Bill Of Materials

Component / Description	Items	Unit price(€)	Total (€)
PCB Terminal Block (2 pin)	16	0,700	11,20
PCB Terminal Block (3 pin)	2	0,935	1,870
MCU Atmel ATmega8515L-8PU	1	5,700	5,700
MCU Atmel ATtiny2313-20PU	1	2,650	2,650
BJT Transistor BC635	7	0,100	0,700
Panasonic SPST 12V Gen.purpose Relay (N.Open)	7	1,500	10,50
SPST Relay socket (base) for PCB	7	1,000	7,000
Resistors ¼ Watt various manufacturers	15	0,010	0,150
Rectifier diode 1N400x	12	0,140	1,680
Low signal diode 1N4148	7	0,090	0,630
TI/ST/ON Voltage regulator LM317	4	0,290	1,160
Bourns Multi-turn potentiometer 5kΩ	4	1,600	6,400
Crystal 4.000MHz	1	0,250	0,250
Socket IC Through-hole 40pin for ATmega8515	1	0,250	0,250
Socket IC Through-hole 20pin for ATtiny2313	1	0,250	0,250
Socket IC Through-hole 28pins (27C64/27C128)	1	0,250	0,250
Socket IC Through-hole 32pins (27C256 and above)	1	0,250	0,250
Single Side Copper FR-4 board 10x15cm	3	2,600	7,800
34pin female header for flat cable (FDC like)	2	0,800	1,600
34pin male header (pins) & plastic housings with edge	2	0,800	1,600
Capacitors various manufacturers/various values	30	0,050	1,500
Diode bridge KBPC808 (I know, too much but hadn't smaller)	1	2,700	2,700
Transformer 230V/24V/1A (too much, but was available)	1	4,000	4,000
Flat cable 34pin (0,5m or less used – 1m purchased)	1	0,800	0,800
Plastic distance sleeves (spacers) for PCB corners	30	0,040	1,200
Nuts & bolts for fixing PCBs in case	30	0,002	0,060
DB9-USB-D5-F Interface Module (mini USB/DB9)	1	14,00	14,00
Peoject Metal Box (dimensions approx 32x25x8,5cm)	1	20,00	20,00
Totals	190		**106,15**

The above are prices found here in Hellas (a little spicy ones!). Surely you can find most components cheaper, lower total cost by don't using sockets for ICs, relays, use smaller transformer don't use USB to serial module, use pin headers instead of terminal blocks, other box or no box etc etc etc. Lowering of the total cost can be done, it wasn't my intention, as many of the components already existed (I don't know if you can find any BC635 transistors now, they are probably obsolete – use any other i.e BC337, BD series etc) so it was a good project to cleanup some older components.

Main power supply & filter

'317'-based power supply for +25V DC
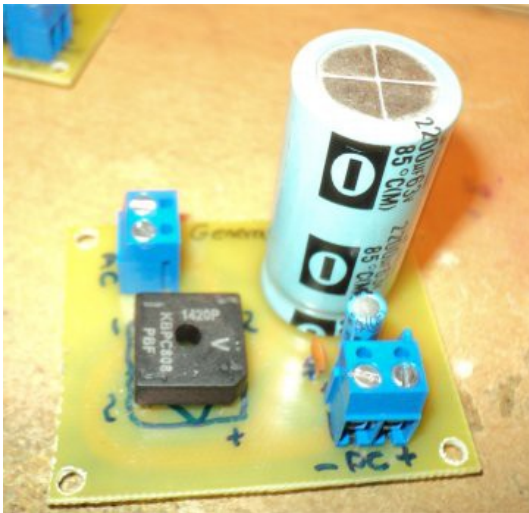
(board size 6cm X 4,5cm)

(board size 5cm X 4cm)

Multi-output 317-based power supplies +21V, +12.75V, +12V DC (Left)
Multi-output 317 & 7805 5V and 6,25V power supply Vcc / MCU (Right)
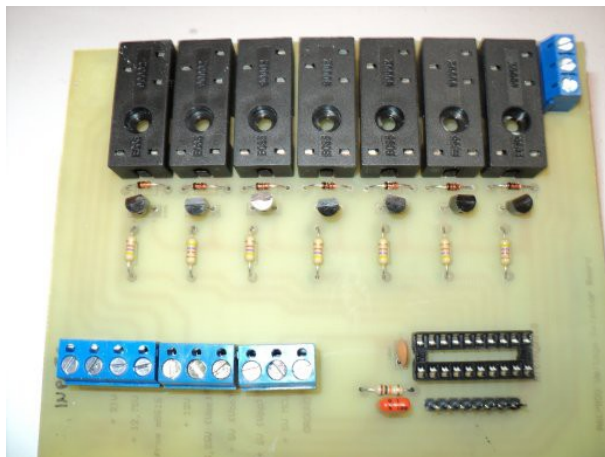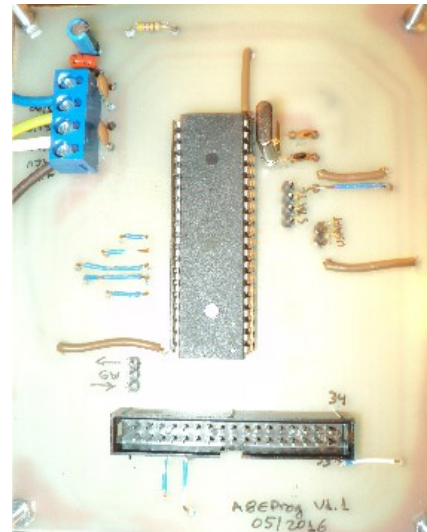
(board size 6cm X 7,5cm)
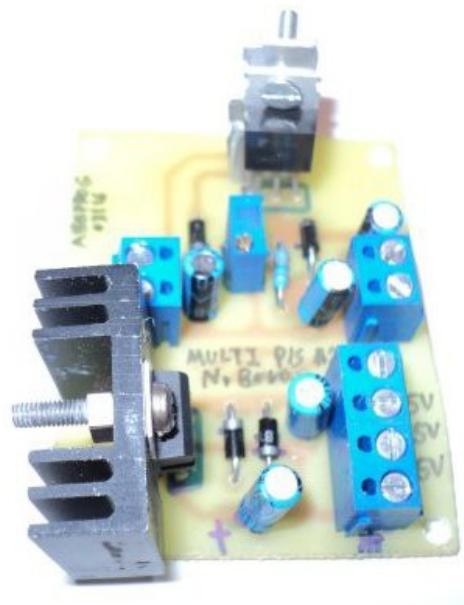
(board size 5,5cm X 6,5cm)

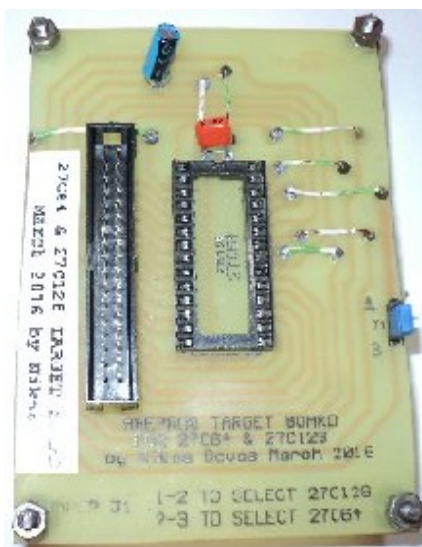**State MCU board & Voltage selectors**   **Main MCU board (ATmega8515)**

(board size 12,5cm X 10cm)   (board size 10cm X 12,5cm)

Some target boards that created (for 2764 – 27128 – 27256 - 27512)

board size 7cm X 10cm   board size 8,5cm X 10cm   board size 7,5cm X 10cm
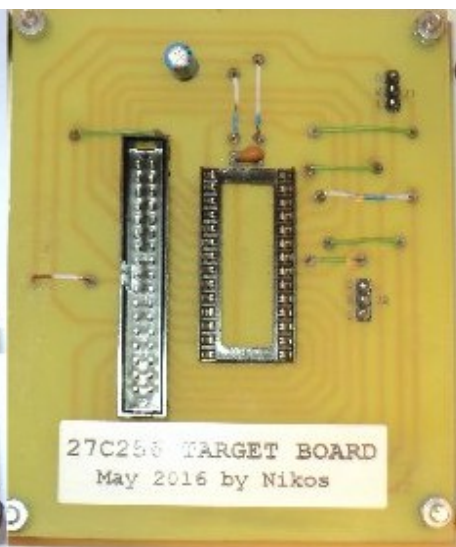
Inside view (circuits and inter-connections)
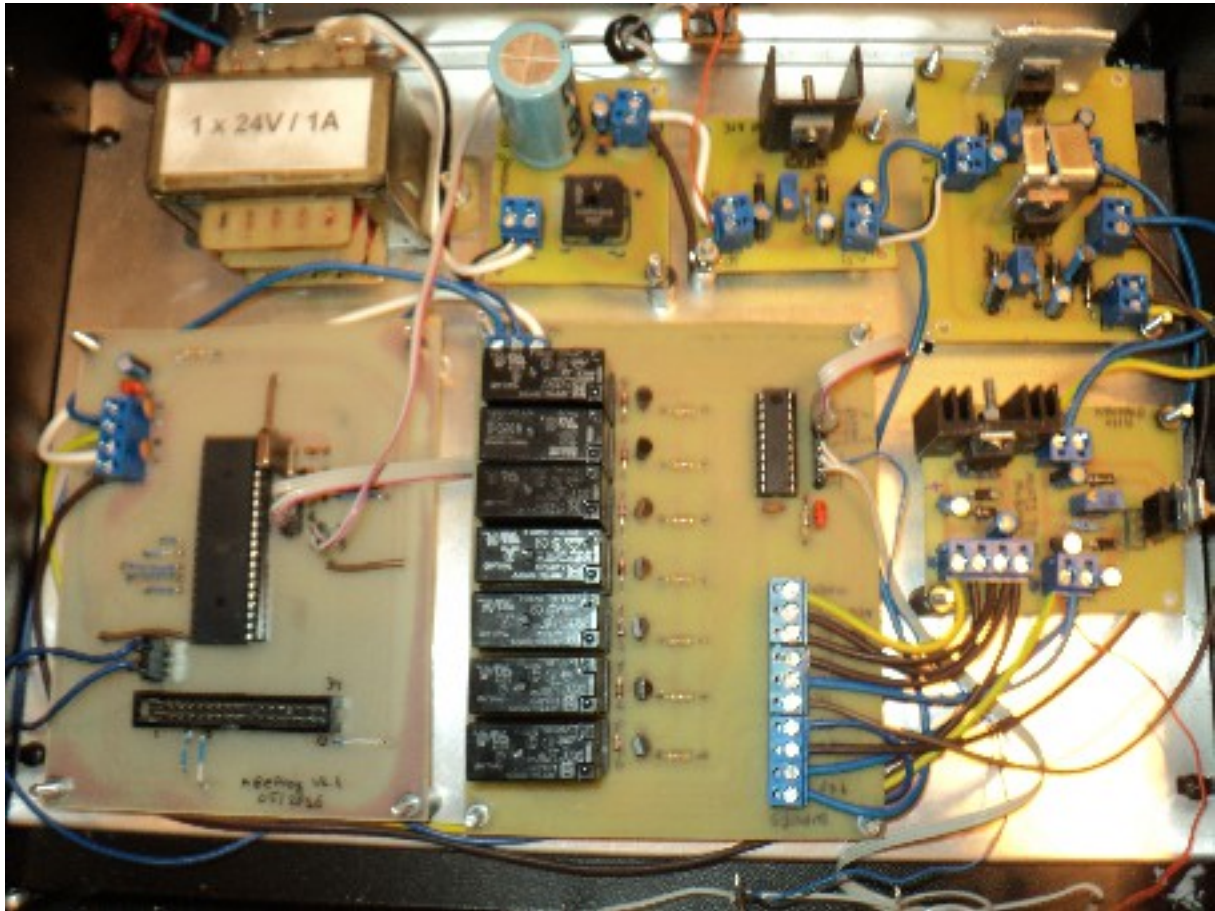
(Not my best wiring ever, but descend enough for the construction and number of PCBs)

The above picture shows the boards on a "plateau" inside the metal box (an aluminum plate that is placed and screwed inside the metal box). The placement is as shown (back side is on top and front side is on bottom of the picture). The case is grounded (in the left/bottom screw/bolt of the second pcb from top-left). Also the left screw/bolt of transformer is the grounding point of the mains supply (not shown in this picture, will be shown in the next pictures). The transformer as already stated is too big for the project, a 200mA could do the job as well, I just had it and used it. All PCBs were placed using spacers and screws/bolts to the plateau (to prevent any PCB to contact the aluminum base, which will lead to a short circuit – boom - boom !!!).

If you want to operate 24/7 then you might consider adding a small fan to the box, to deliver the heat out of it. In my tests all ICs of the power supplies where running relatively warm - considering the high temperatures in Greece this period (June) of about 35° C in free air and a fan wasn't necessary (but of course I didn't use it 24/7, a few hours per day during the tests and reading / programming). If you want 100% cool supplies you might use a fan and larger heatsinks than mine, but with those heatsinks on photos the maximum temperature recorded was approx. 55°C (about 20 degrees above room temperature)

(rear view of the box)

The D9-USB-D5-F module of Future Technology Devices International Ltd is shown along with the power plug, switch and fuse holder. The labels/stickers are "Nikos made" because I create them easier than printing on metal ! The 100mA fuse could be less if you like (it was the smallest fuse I had), for the 230V power supply, 100mA will occur on 23 Watt of power consumption, which is rare to happen in normal circuit operation as power consumption is much smaller, but if you place a larger capacitor in the main power supply, you probably have to maintain fuse's value, because the start-up current of the main power supply might be larger. For 110V/120V AC countries change it accordingly.
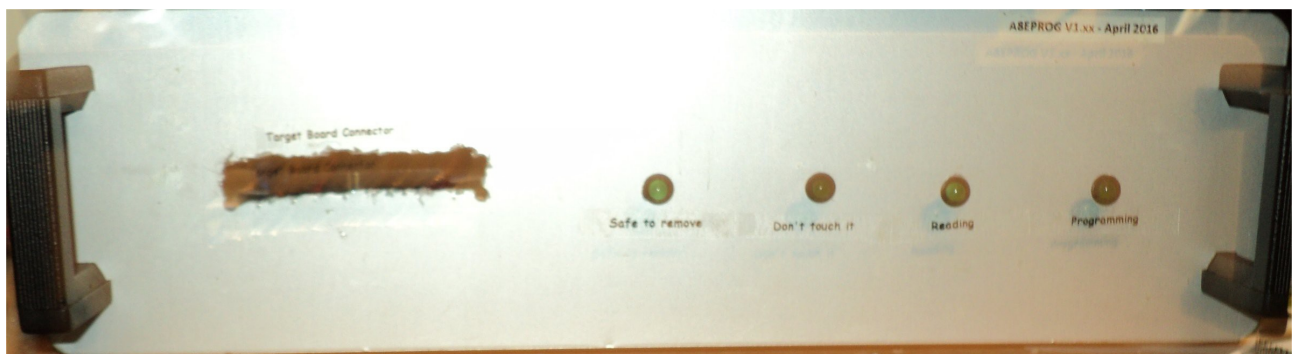
(front side of the box – the hole in the left is for the output connector)

I was searching for a connector to add there in the whole (the dimesnions are for a male connector similar to that shown in the picture below – with 34pins), but finally I settled to just connect the ribbon cable to the main board and cover the rest of the hole...

I'm more than happy with that as connectors are "reversed" so I had to make another PCB which I didn't want to do (a male connector have the pins mirrored compared with the female connector that the ribbon cable has).
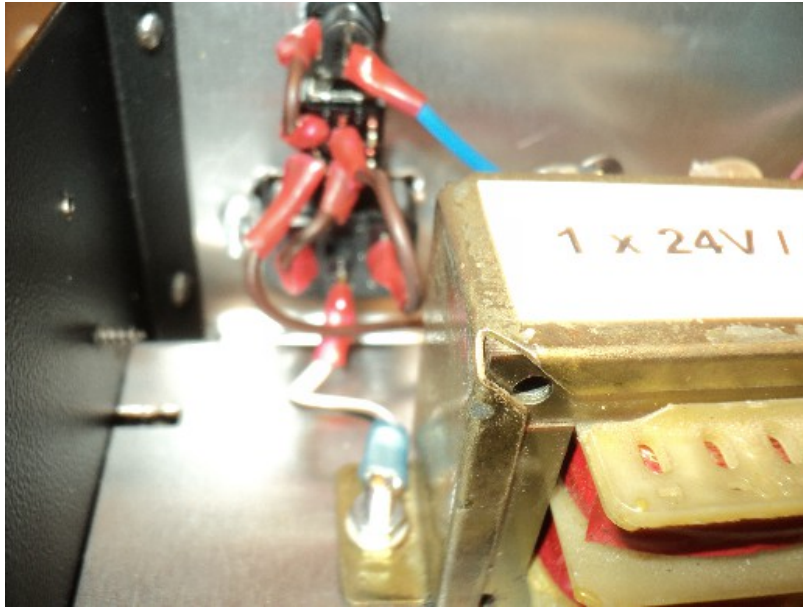
(Inside view of the power plug, switch, fuse holder, mains grounding and the transformer)

Safety first! All live contacts were covered with thermal-shrinking material, the contacts on the transformer (primary and secondary) with silicon, so to decrease to the minimum the possibility of touching any live parts. A dual pole double transfer switch (DPDT) used for the ON/OFF function.
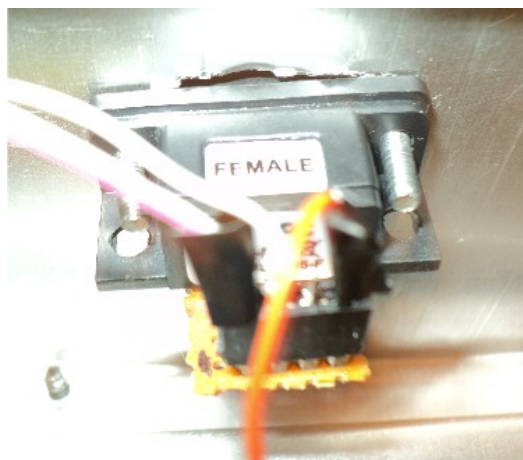
(closer view of D9-USB-D5-F rear - inside on box)

A small PCB was made from general purpose PCB to attach the module. Three wires required for the communication (Rx, Tx and Ground). The module was screwed to the back of the box, soldered to the general purpose PCB and a header is used. The 2 wires (on the left) go to ATmega8515 board, whereas the other one is the signal ground.

The A8Eprog software

The software that accompanies the device, is the A8Eprog PC Software. It is a Microsoft Windows™ compatible application and it is used to control the A8Eprog board.

The package contains the setup of the software also for machines not having the Visual Basic* libraries installed. For 64-bit operating systems you must replace the **setup.lst** file of the \32bit folder with the one found in \64bit folder in order for the installer to copy the files to the correct destination folder (usually in \Windows\SysOW64 folder - and not in Windows\System32 that 32bit operating systems have). One possible exception is if your system was upgraded from 32bit to 64bit you still might have System32 working folder (on the newer Windows 10 versions I have seen such a folder to exist – you have to check for your PC). The default folder is C:\A8Eprog but you can change it during the installation. If any of the distributed libraries are older than the versions you have on your PC, it is mandatory <u>to keep your versions</u> (the newer ones), so no program will have any malfunctions after the setup (or removal) of A8Eprog. For 64-bit operating systems or for systems that have user/account control, you must have administrator privileges – as in all installations, in order for the installation to complete successfully. After successful install, you can start the software by it's icon. If Visual Basic Libraries are already installed (i.e by a previous software installation), then no updates would be done in your system files. You can edit the **setup.lst** file to change any option you want, or set the correct paths to your Windows folder.

If you build this project with the DB9-USB-D5-F module then you have to download and install the drivers for your operating system from the website of the manufacturer, Future Technology Devices International Ltd http://www.ftdichip.com/FTDrivers.htm otherwise you can connect the device to a free DB9 serial communication port of your PC and use a MAX232 or equivalent integrated circuit for the serial communication.

As with all software, some bugs may be found (I hope not!), please report them to be corrected as soon as possible.

Please note that the current software version, might appear a little different in what is shown in the pictures – these are for the first version of the software – whereas the functionality is still the same.

Visual Basic is a trademark of Microsoft Corporation
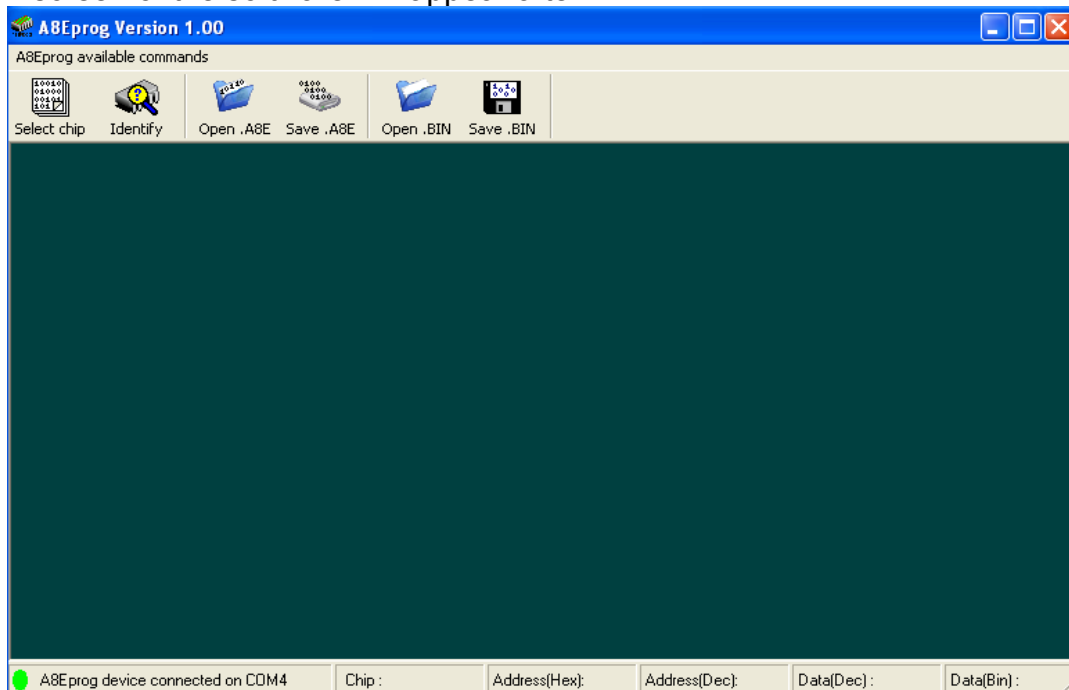
Starting & main screens

When starting the A8Eprog software for the first time, you must setup a communication port. Most likely the following window will appear:
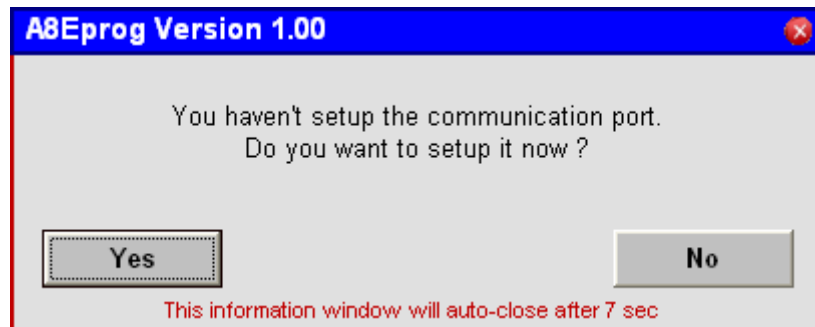
and by selecting Yes, you will be transferred to the setup window:
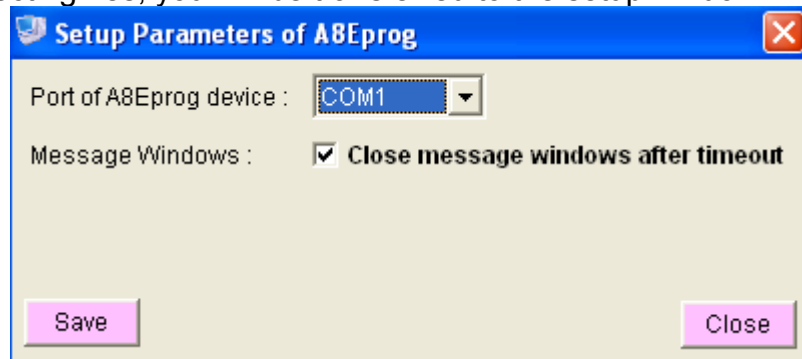
and you can select the communication port. Click **Save** to save settings. After that the main screen of the software will appear after:


If the software is unable to find the A8Eprog device a red circle will appear in the first panel of the status bar along with a message ("not found"), otherwise a green circle will appear indicating that the device was discovered by the software.

If for any reason the software was unable to communicate with the device on the defined port (i.e the port is already open by some other application), the following window will appear:
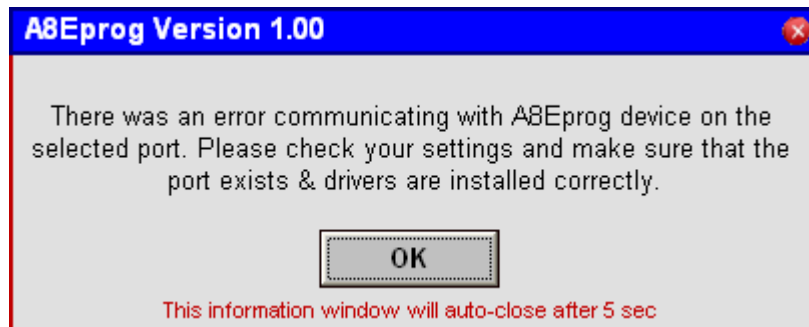
The message windows are automatically closed after a small period (5 – 10sec). To change this behavior you can go to **Setup parameters...** and uncheck the selection box of Message windows (Close by user - no timeout). A typical window without auto-close:
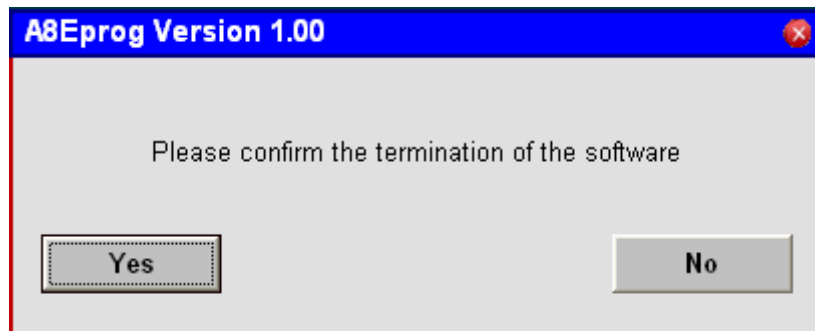
Will wait until the user selects Yes or No (or X from top-right) just as the standard message boxes of the operating system.

For decision-making windows in auto-close mode, after time-out **No** is selected.

The software is built with Multiple Document Interface (MDI), so you can have many different windows open at the same time. Some care however must be taken when commands are sent to the device (to have the right window active and right chip in socket). If you find difficult to work with multiple windows, just have one window open every time. On the bottom of the main window (the status bar) information about the active window (the chip selected) address and data byte in various formats (data in buffer is in Hexadecimal format, whilst on status bar the decimal and binary equivalents are shown) are displayed. The decimal value of the selected address is shown also. The following picture might be a little different in current version, but the functionality remains the same.
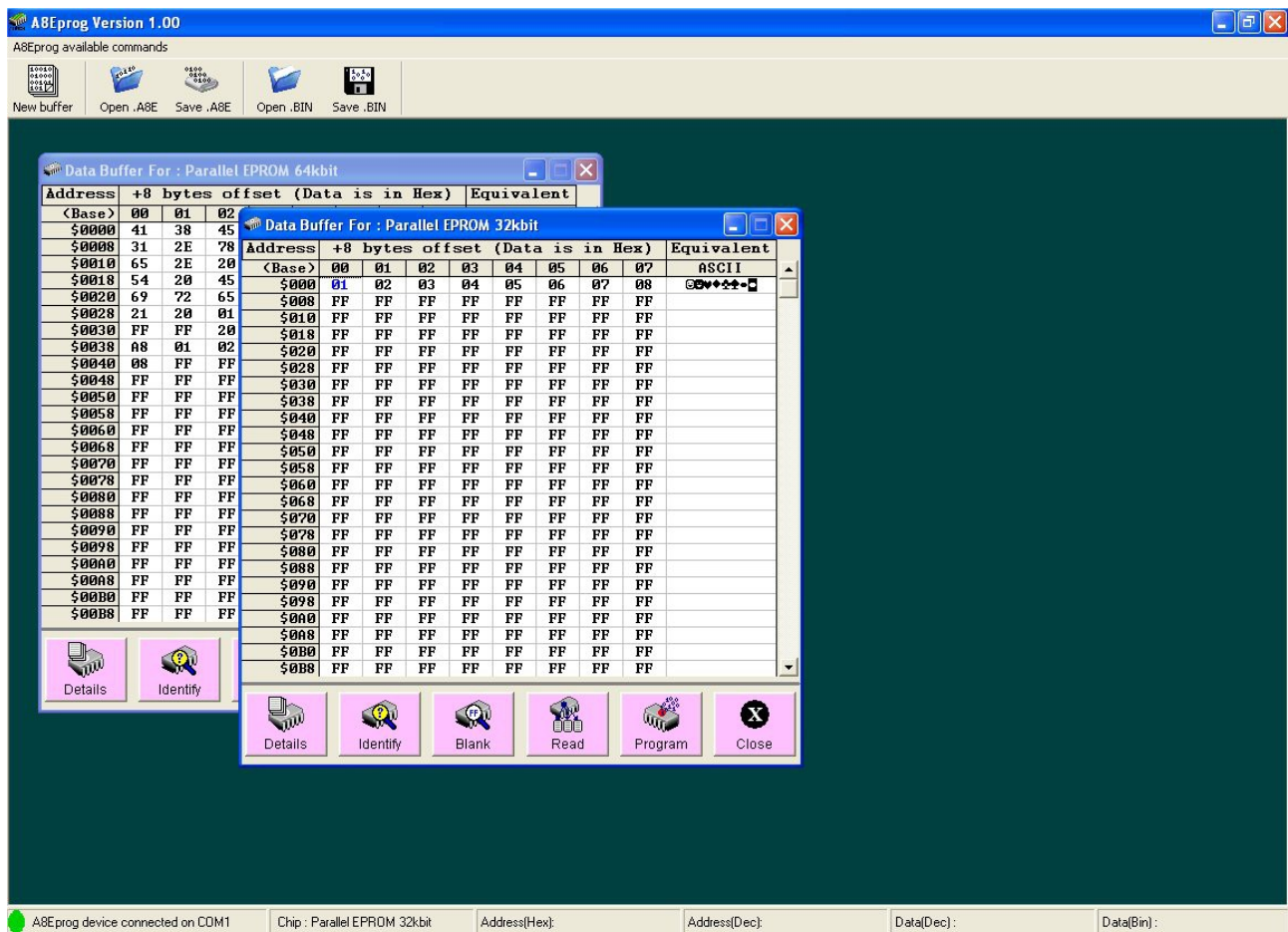
Multiple Document Interface (MDI) concept

The status bar changes when the active window changes (the data for the active window for a 32kB EPROM is shown in the above picture).
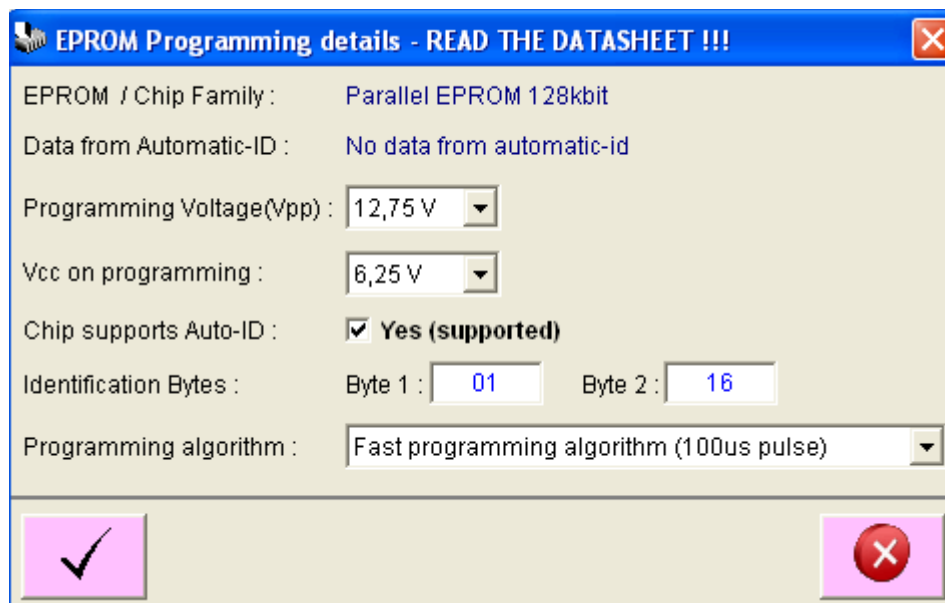
EPROM programming details & supported files
The software support two kind of files:
a)   binary files containing only the data (the ordinary type of binary files) and
b)   special files (A8Eprog format files) that contain both the data and the details of each chip (programming details).
      You can choose to work with any file format you like.

After you have created a data buffer/load a file, you can click on the **details** button, to set-up the programming parameters (if you want to program the chip). For reading a chip it's not necessary to set any programming parameters (but it is recommended to enter the programming details for future reference). Note that these settings are saved with the data buffer, if you choose to save the buffer to A8Eprog format file. If you want to save the data buffer as a binary file, these settings are not saved (only data saved).

A typical details window would look like the following:

If an automatic identification was made previously, then all the above settings are automatically entered by the software for the identified chip.

Please note (but you already know !)
For all UV-EPROM chips you can't program a byte in an address that contain data that have '0's on it's bits – only '1's - thus for a whole byte = $FF. As manufacturers state '0' are programmed to the chip, '1' can't be programmed, but are send to the chip ('1' only produced after a UV chip erasure). For example if one address contain the byte $2A (in binary this is 101010) then if you want to program the byte value $2E (101110 in binary) you can't because '0' can't be programmed to '1'. Instead if you want to program the byte $1E (which is 011110) the resulting byte on that address will be 001010 or $0A because '1's will change to '0's but not vise verse. That's why to avoid such instances always perform a UV erasure (thus having all bytes to '1') before programming, or program single bytes that already contain the value $FF.

May 2016 Updates - EPROM data buffer window (updated)

The main window that holds the data read/to program was updated:
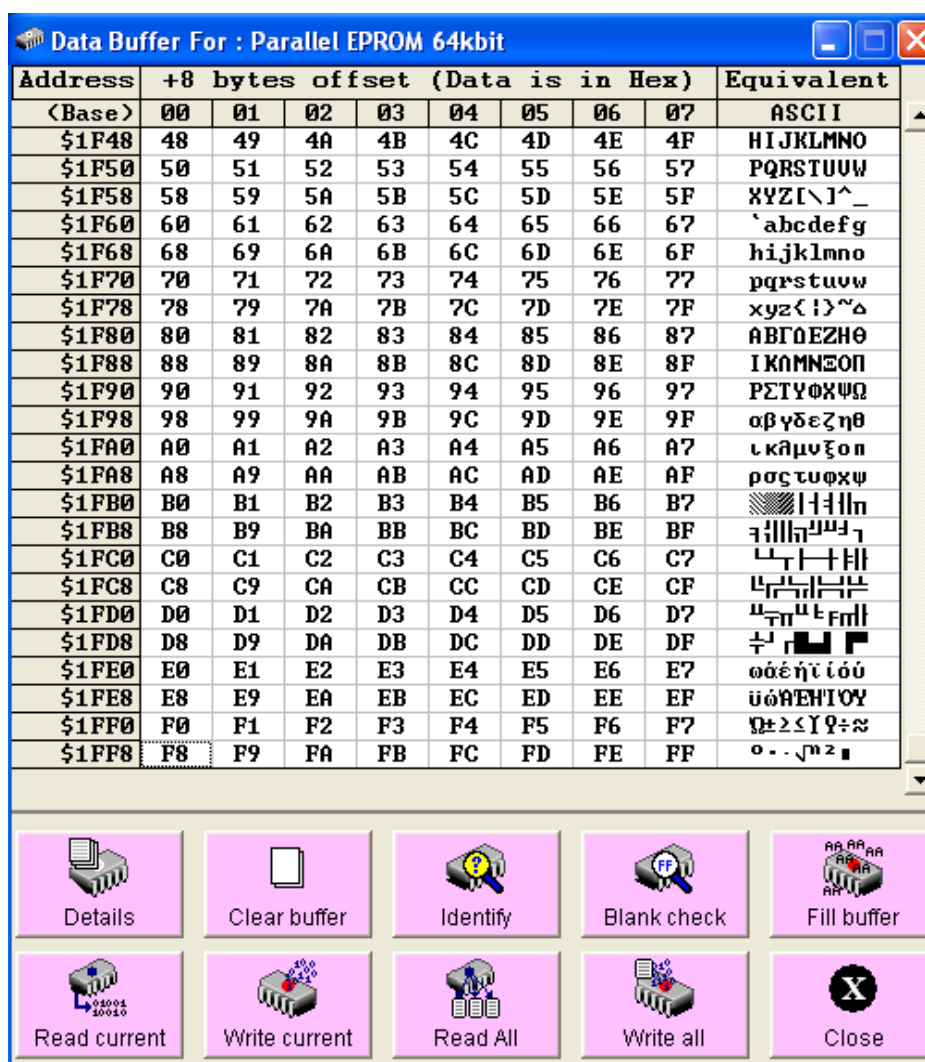
Addresses are shown on the left in hexadecimal format (with digits according to the maximum address value i.e the 64kbytes EPROM has maximum address $1FFF so 4 digits will be used for all addresses etc).

Navigate using the mouse, cursor keys, Page Up/Down Home, End and enter the value in hexadecimal format and hit [Enter] or click on any other field within the address range. Valid values are from 00 – FF obviously (0-255 decimal), if something else is entered outside these values an error window will pop-up. You can enter a value to any address using mouse or keyboard, you <u>can't</u> however enter a value in the ASCII field (I haven't enable it, just change the hex value and the ASCII field will be updated).

The various command buttons perform the actions mentioned, not very much to explain, as they are self-explanatory. These buttons are disabled if the A8Eprog device haven't been discovered on the selected port (and so you can load/save only). There is no need to explain all the functions, you may find them very easy. Just a few more words for the newer functions added on May...

Fill buffer and rest function buttons added

The "*Fill* buffer" button upon clicked will show the following window:
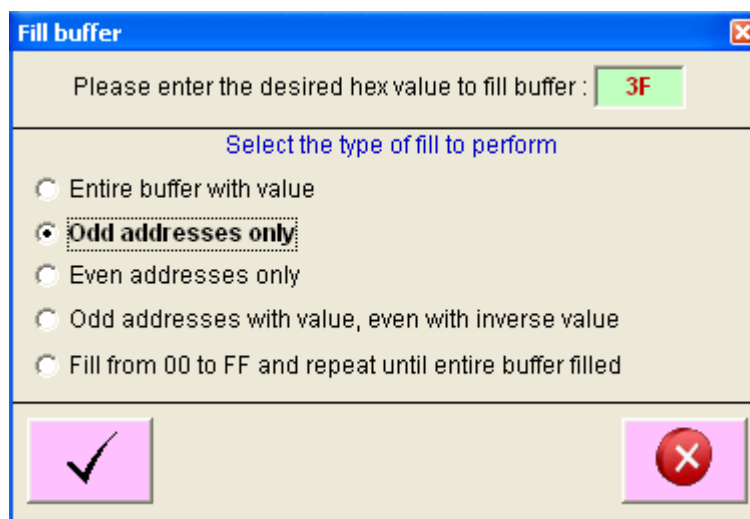
You simply enter a hex value to the text box and select the desired fill. You can fill entire buffer with the value specified (3F in the above window), fill odd addresses only, fill even addresses only (address $000 is considered even). By selecting the fourth option the half buffer (odd addresses) will be filled with the specified value and the rest half (even addresses) with the inverse of value. In the above example if 3F is entered the half buffer will be filled with 3F and the rest with C0 (the inverse value of 3F). By selecting the last option, the whole buffer will be filled starting at address 0 up to the last address, by applying incremented values from 00 to FF (this is shown in the previous page in the data buffer picture). All these 'fillings' are quite good for testing purposes.

The above buttons added are self-explanatory also... The **Clear buffer** button obviously clears the whole data buffer (applying the value FF to all addresses of buffer – NOT on chip) and the two other buttons are used to **read one address** and **write one address** respectively, this could be very useful especially if you want to alter few values in a chip. Before reading and writing the selected addresses a confirmation window will appear.

The **Blank check** button instructs the MCU to read the whole chip and report if it is blank or not (if all bytes have the value $FF).
You will notice some "technical delay" generated by the software (~2sec).
This is done to protect the relays from oscillating. The mcu needs way less than a second to read the entire chip, but during that small time, all voltages would have to applied and removed (because after any operation the A8Eprog returns to "Safe" mode). So to prevent oscillation of the relays (and to have stabilized voltages on the chip), a small delay is inserted between the various steps, which totally is about 2 seconds before **!BLK** command is sent to the MCU, which is practically no time...

Automatic identification – May 2016 Update

This was implemented as a seperate function. Still a similar function exist in the data buffer window, but this one gives more information and the ID bytes retrieved and creates a buffer for the identified chip. The following picture shows the auto-id window:
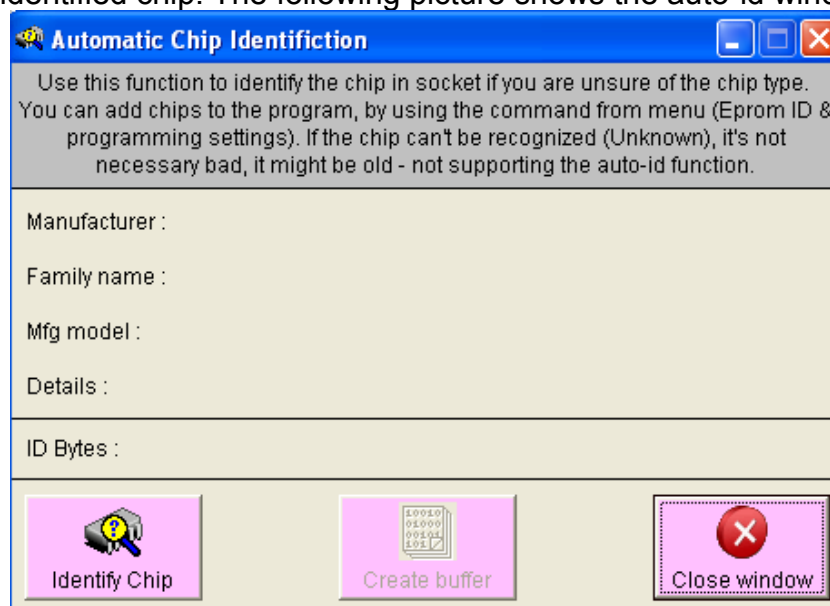

After successful identification of a known chip, the 'Create buffer' button becomes enabled. If you click on it, a data buffer of that size is created, otherwise you will be directed to the chip-selection window if the size is unknown. The following picture shows a chip-identification made on an Amd's 27C128 EPROM that was tested.
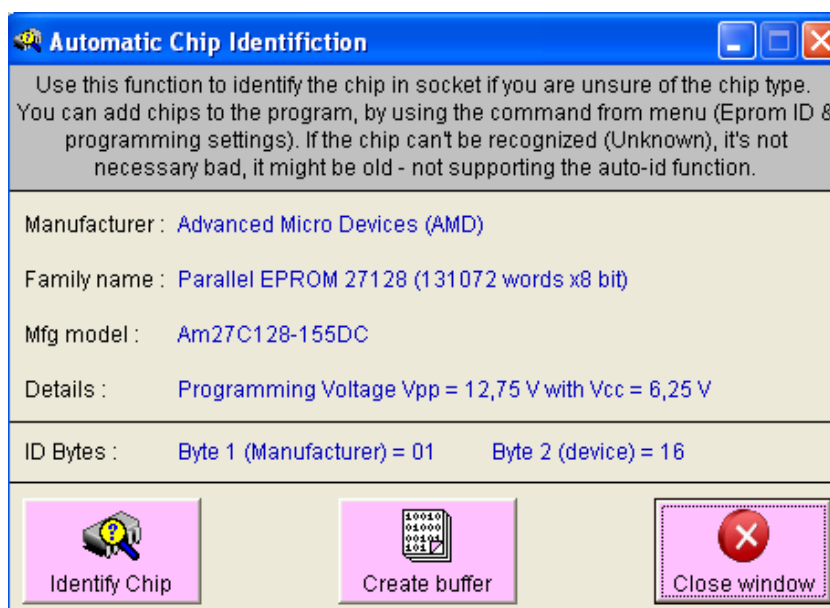

The identification data are retrieved from the **A8Eprog.Dat** file which is created by the software, by selecting the menu "Eprom ID & Programming Settings". Note you <u>must not</u> edit the file with text editor, it's a binary file. Some data already exist for my chips, you may add yours. The same stands true for the Manufacturers file **A8Eprog.Mfg** which holds the Manufacturers and one ID Byte. Also a binary file, add/edit/delete only with the A8Eprog software (you can do it with hex editor if you preserve record's length).

## Full programming and one address only programming

Of course reading a chip is only half the way. There are 2 buttons ('Write all' and 'Write current') that can be used to program bytes to the chip. As these buttons state, a whole chip programming or only one address is programmed. In fact it's a 3-way because in whole chip programming you can define an address range instead of full chip programming. For the whole-chip/Address range programming and for single address programming, a verification is performed after every byte programmed, according to the various algorithms (in fact a combination of the most used verification process is used for each of the selected programming algorithm). The following pictures show these functions:
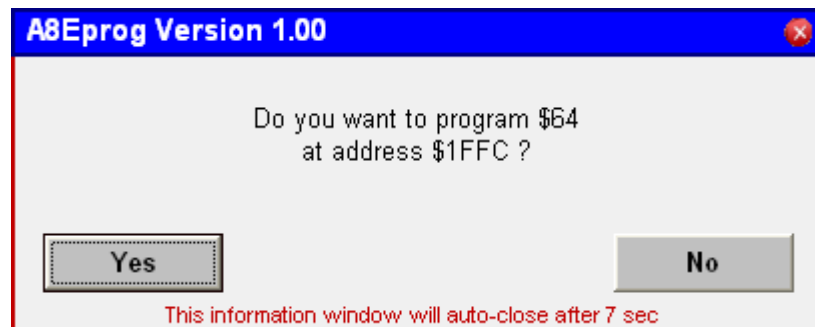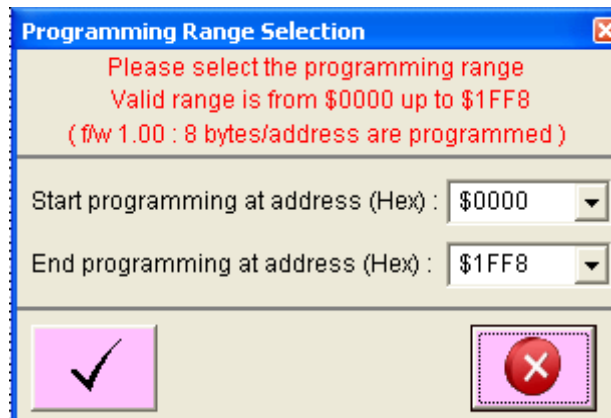

and after clicking on 'Yes', a programming / verifying is performed:
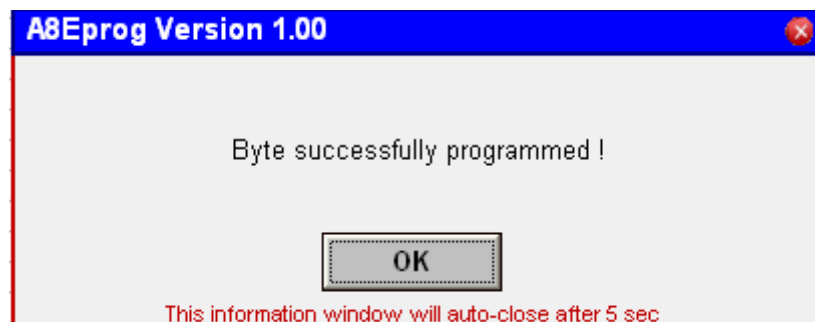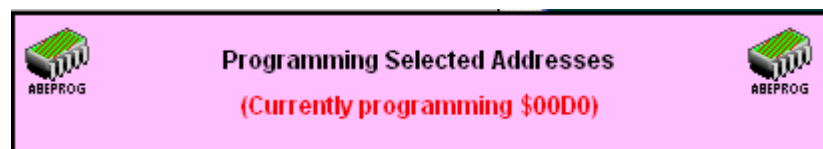

If for any reason (i.e address contain value already, bad chip etc) or the programming and verification can't be done, the following window will appear:
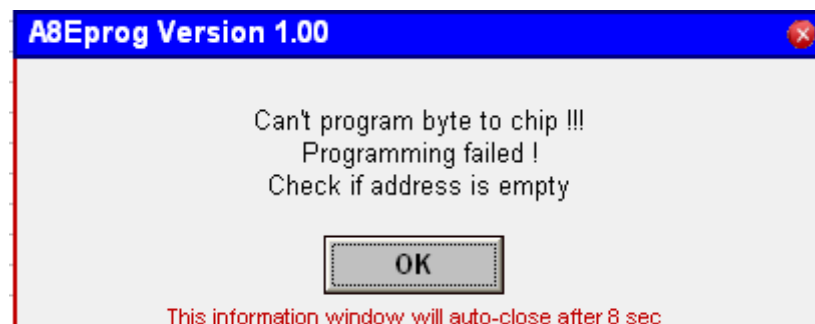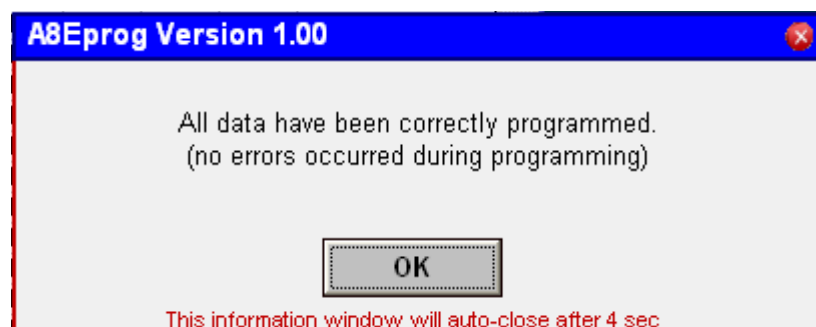
The Write-All button, shows the following window for range selection (the picture is shown for a 27C64 Eprom, whereas range selected is for whole chip programming. The addresses refer to the starting address of each group of 8 bytes).

Programming Range Selection

Please select the programming range
Valid range is from $0000 up to $1FF8
(f/w 1.00 : 8 bytes/address are programmed)

Start programming at address (Hex) : $0000 ▼

End programming at address (Hex) : $1FF8 ▼

✓   ✗

As shown above, last address for 27C64 is $1FF8 (+ 8 data bytes for the $1FFF final address). As for this version of the software 8 bytes are always programmed so the range is referred to the starting adresses of each 8 bytes group. If you don't want to program 8 bytes in the last address, you can program up to the previous one and program single addresses only (for the number of bytes required).

Programming Selected Addresses

(Currently programming $00D0)

Information window during programming

A8Eprog Version 1.00

All data have been correctly programmed.
(no errors occurred during programming)

OK

This information window will auto-close after 4 sec

And after successful programming

Programming is of course much slower than the reading process because each byte is programmed (up to 10 times if necessary - according to the selected programming algorithm) and then read from the programmed address for verification. If byte read is correct, it is reported to PC as correctly programmed. The PC software is instructing the MCU for the address and data byte to program, so the bigger the EPROM the more the programming time. A programming scheme that is a little bit slower than the most common supported was chosen (regarding the reading process, in the area of some micro-seconds instead of nano-seconds that most EPROM support), to be 100% sure that any chip will support it. Of course timings for the programming sequence is exactly as required by the various programming algorithms. As stated already time is not the main thing here (at least not for me), so programming timing is not the fastest, but does the job. If you don't want to full-program a chip, you can always program a selected area... The following table shows some programming times for whole chip programming using various chips.

Chip in target board	Action performed	Time	Notes on action performed
NSC NM27C64Q	Full programming	~ 50sec	Click to finish programming
Intel D27C256-1	Full programming	~200sec	Click to finish programming
ST M27C512-20XFI	Full programming	~ 400sec	Click to finish programming

Because many-many pages written already, I will stop here...
The rest functions you will discover by using the software....

I think I over-wrote...Again...

See ya.!