eprommer

COLLABORATORS			
	TITLE : eprommer		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 9, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 eprommer **1**

1.1 main . . . 1
1.2 history . . . 2
1.3 aboutfiles . . . 3
1.4 miscnotes . . . 3
1.5 todo . . . 4
1.6 hw-information . . . 4
1.7 rebuild . . . 5
1.8 programdescription . . . 6
1.9 mainwindow . . . 6
1.10 loadsave . . . 7
1.11 address . . . 7
1.12 epromtype . . . 8
1.13 action . . . 9
1.14 buffer . . . 9
1.15 optionswindow . . . 10
1.16 testwindow . . . 11
1.17 compile . . . 11
1.18 intelhex . . . 12

Chapter 1

eprommer

1.1 main

November ↩
, ↩
93
Paderborn/Germany

Amiga Eprommer V3.2d

Hardware and Software by Bob Blick and Udi Finkelstein.
Modifications to use WB 2.0 and Amiga's with more than 7MHz by Carsten Rose.

Table of Contents:

History

About these files

Some notes

To do first

Technical information

How to build the eprommer

Program description

Main window

Options window

Adjust hardware window

Compile the source

IntelHex description

1.2 history

Cause there are 3 people who worked on the eprommer over the years,
the documentaion is a littel bit confused. In order to seperate Bob,
Udi and Carsten, I (Carsten) wrote the author names in front of
some text parts.

A short history:

Bob:
     If you would like to reach me, you may send E-mail to PeopleLink,
my handle is BOFFO*BOB. You can also send E-mail to MCAB-BBS(a local
Amiga BBS) at 707-964-7114 to BOB BLICK (my real name). My address for
real mail is:
  Bob Blick
  Box 916
  Mendocino, CA 95460

The original project was done by Bob in 1989. This includes the complete hardware,
a first eprommer program (written in Basic) and the main part of this
documentation (especially the hardware part).

Some hardware changes and a new eprommer program (based on the old Basic program,
but written in SAS C)
was done by:

  Udi Finkelstein
  10 Glitzenstein st.
  64686 Israel.
  finkel@math.tau.ac.il

Espicially by Udi:

Hardware modifications:
He made a few hardware modifications to the circuit in order to make it
work better, and to make calibration easier.

*) He replaced the 4.3K resistor conected to the LM317, with a 5K trimmer.
   This way we can get accurate 25V for 2716.
*) He replaced the 2N3904 with 2N2464. He guess this is not necessary, but for
   some reasons he couldn't make his own programmer work until he used these
   transistors, which he saw on a friend's EPROMmer.

   Carsten: I put in BC 547, worked much better than 2N3904.

The newest changes on eprommer program (based on the source of Udi) with
a WB2.0 Look, ability to load, save Files with IntelHex format, were done by:

  Carsten Rose
  Leostr.35
  33098 Paderborn
  Germany

  EMail: caro@uni-paderborn.de

in 1992 and 1993. Carsten also changed,deleted and appended some parts of this

documentation file.

Program Versions:

1.0 Basic program by Bob
2.0 C-program by Udi - official release
3.0 C-program by Carsten (WB2.0)
3.2b First official release
3.2c Bugs during CIA initializing removed
3.2d The CIA delay time is now customable

1.3 aboutfiles

ABOUT THESE FILES

Eprommer - the executable (for WB2.0 and greater)
Eprommer.guide - this file.
Eprommer.pic - the schematic (IFF).
Personality.pic - schematics for the personality modules, (IFF).
src/* - the source code (compiled with DICE 2.07.54R)
src/boopsi.o - compiled picture bitmap of filerequester gadget
src/Dmakefile - DICE makefile
src/defs.h
src/Eprommer.c - Main program
src/Eprommer_protos.h - Prototypes for eprommer.c
src/Eprommer - Executable
src/Eprommer.h - Header file
src/IntelHex.c - Procedure's to handle IntelHex format
src/IntelHex.h - Header file
src/gad_eprommer.c - Automatically generated source of GadToolsBox
src/gad_eprommer.h - Header file
src/gad_eprommer.gui - Definitions in GadToolsBox format

1.4 miscnotes

Bob:

WHAT ARE WE HERE FOR?

I designed an EPROM programmer and wrote some software for it. Here we have the schematics and the source and compiled versions of the program. Since I've built it, it's been used for other EPROMs as well. It will read and program 2716 to 27512 EPROMs. It will also read 24 pin 8k roms as well. Also, it wouldn't take much to modify it for 32 pin EPROMs up to 4 megabit. By the way, this whole shebang is my property and you are only allowed to build it and use the software personally, no selling or profit-making without my written consent. You may send the entire arc-ed file to other networks as long as you do not remove anything from it. I also intend this programmer and the software for casual use around the house, and can not be responsible for damages from it, it's use, etc. The EPROMs it makes work fine for me, but don't put one in a heart-lung machine or an MX missile. Especially not in an MX missile (though I have heard that worse parts have gotten into a

few).

This programmer works pretty well, but if you'd like to improve it or the software, please feel free. I would love to hear from anyone with comments, as well. I have yet to find anyone to share hardware hacks on the AMIGA with. There are a few other projects I will probably upload eventually, so keep your eyes peeled. By the way, if you haven't built electronic projects from scratch before, check with an expert before attempting to build this circuit. The schematic makes certain assumptions about your electronic prowess.

1.5 todo

WHAT YOU SHOULD DO FIRST

Print out the pictures. When you look at them, the aspect ratio may seem a little squat, but they will print nicely.
The pics are 640x408, 2 color. Set preferences pitch to elite (important!). In the dpaint print menu, set margins at 1 and 64, page height 34 lines, %high=100, %wide=100. Use the old preferences printer drivers, they glitch less from dpaint. There's one on the boards called newepson.dri that doublestrikes and looks real good.

1.6 hw-information

Generally there are 2 solutions to handle different types of Eprom Pinning (and SRam's).

a) you build a lot of digital logic to solve all possibilities of IC-pin assignment.
b) you have a little hardware matrix (16 Pin socket) with personality modules for all different eprom's.

This eprommer use the low cost version (b) !

The SRam's use a special way to program. Cause they have a \WR pin (eprom's have a PGM pin with a different meaning) and no \WR was planed at the beginning of the project, A15 is used as \WR. The correct handling is given through the personality modules.

ABOUT THE CIRCUIT

The first part is the power supply. It requires an ac adapter with a 12vac rating, meaning more than 12v unloaded. Half wave rectification is used, and a voltage doubler for the programming voltage. There's a negative 1.2 volt bias supply so the 317 regulator can swing to 0 volts.

The 317 has three transistors attached to it so it can supply three programming voltages plus either high or low TTL level, depending on jumper 8-9 on the personality module. The programming voltage is software selectable. The first 74HCT374 turns on and sets the programming voltage. It also can set up one of the other 374's for

latching an address. It does it like this: When we make SEL high, the first 374 latches onto a byte from the parallel port. If the first bit(loaddr) is low, we have set up the third 374 so that when we then make SEL low, the third 374 is clocked, latching the lower address. This all assumes we have been stuffing the appropriate data out the parallel port. So we go like this: stuff a command out parallel port, clock SEL high, stuff a byte of the address out parallel port, clock SEL low, stuff a command out parallel port, clock SEL high, stuff other byte of address out parallel port, clock SEL low. Now you have the address set for the EPROM, and you can read or write it. BUSY is hooked to both OE on the EPROM and DIR on the 74HCT245, so if we want to read the EPROM we first should make the parallel port tri-state, then we can pull BUSY low and the EPROM's data will be sent to the port where we can read it. If we want to write to the EPROM, we would leave BUSY high, set our programming voltage, and pulse POUT low. POUT is connected to PGM on the EPROM, so that's all it takes. On 2732 and 27512 EPROMs, some pins do more than one thing, so we have to change the program slightly for them, but basically that's it. The personality modules adjust the pinout for the various EPROM types. By the way, you will notice there are 3 unused pins on the command 374. You can hook anything you want there, but I intended to use them for the extra address lines required by some of the new jumbo EPROMs in 32 pin packages. Keep your eyes peeled for updates.

Carsten: I replaced 1N914 with 1N4148 - it works.

1.7 rebuild

(Bob)

HOW TO BUILD IT

I wire-wrapped the thing and put it in a plastic box. I suggest using a double row header or some kind of connector right on the perfboard so you don't have a big cable hanging off the end while you are trying to build it. I also used zero insertion force sockets for the EPROM and the personality modules. I plugged and glued the ZIF sockets into wire wrap sockets on the board to get them in the air. I glued all the sockets to the board. For ICs, you will need three 74HCT374, one 74HCT245, and a 74HCT02. You can probably use LS parts, but most EPROMS have strong technical specification on LOW and HIGH level during programming and HCT are the best choice for them. HC will probably not work. You will need a 7805 and a 317 regulator, and if you use TO-220 packages you won't need to heatsink them. I used a 12 vac, 800ma AC adapter, you can probably use one rated at 500ma. The current requirements are not steep, but we only half wave rectify plus the voltage doubler, etc. So much for sentence fragments. Don't substitute 1N4000 types for the 1N914s, we need speed in one place and voltage drops in another. You must include the .01-5.1k termination on the three control lines to avoid false clocking when we change data on the parallel port. I smoked a few EPROMs before I figured that out. I had a real long cable, and the programming voltage would come on while I was reading EPROMs. 27512s die when they get 25 volts. You will need some 16 pin dip-type headers to make into personality modules. They are just jumpers, and most do a couple of EPROM types.

Also, to read 2364A ROMs, tell the program

you are reading a 27128 EPROM, but use the 2364A personality module.
Also you must add 4096 to all EPROM addresses, ie to read an entire
2364A, tell the program to start at EPROM address 4096 and to read 8192
bytes. I did it this way because there weren't enough pins on the
personality socket to include A12.

Carsten:
 - Don't put a capacitor on the LM317 between 'Out' and 'Adj' !
 - The cable between PARALLEL port and the Eprommer schould be short (max. 2 m).
 - Be sure you GROUND connection to your Amiga is good - one line in the whole
   cable could cause a lot of traffic, use more.

1.8 programdescription

PROGRAM DESCRITPION

Version: Eprommer 3.2d

Start the program from WB or CLI.

After startup, the hardware will be intialized, a 64kB databuffer allocated and
the latest user setting (eprom type, program algorythm) is loaded.
'Eprommer' also allocate the parallel port. If some other program use the port,
'Eprommer' will warn you. You can decide to continue or to abort the program.
If you continue your work in spite of the warning - your CIA's could be damaged ↩
   !!!

There are 3 windows in eprommer:
  a)
               Main window
                which will be openend after startup on
     the default screen.
  b)
               Options window
                to setup 'Eprommer'.
  c)
               Adjust hardware window
                to adjust and check the
     eprommer hardware.

No menus are supported.

1.9 mainwindow

Main Window

The window is divided in 6 rectengular parts. Each of them is surrounded by a
Bevelbox.

 -

```
Load / Save
 -
Address
 -
Eprom type
 -
Action
 -
Databuffer
 -
Options
 -
Adjust
```

1.10 loadsave

```
  Load or Save

Here you can load/save your datafiles to/from internal databuffer.
Two Types of Datafiles are allowed: Binary and IntelHex. Choose one of them.

Data format:
 IntelHex    All load/save actions will be done in IntelHex mode.
             IntelHex is often used to describe objectcode. Every byte is located
             on a constant address, the data doesn't have to be continous (very
             interesting to program modules) and a checksum is generated too.
 Binary      all load/save actions will be done in binary mode. Every byte is
             copied from and to eprom without any modifications.

Load and Save files:
 There are 3 different gadgets to load and to save:
 a) A Boopsi image on the left side of the string gadget. Click on it, to open
    a filerequester. Choose a directory and a file. After you clicked OK on the
    filerequester, the path and filename is copied to the string gadget and the
    data is loaded or saved.
 b) The string gadget to input the path and filename. Finish your input with
    'Carriage Return' and the data is loaded or saved. If you don't press return
    at the end, you have to click on the LOAD or SAVE gadget, to start action.
 c) The load or save button, to load or save the file which name is shown in the
    string gadget. If there is no filename inside the string gadget, a
    filerequester opens automatically.

See also: {"IntelHex" link IntelHex}

Your last used directoy is stored (load and save could be different) in ENV.
If you want to setup the directories static, you have to select the directories,
open the 'Options window' and click 'SAVE'.
```

1.11 address

Setup start addresses of eprom and internal databuffer.

These gadgets determine the postion on the internal databuffer and on the eprom where to read/write or program the data. Also the length (number of bytes) of data is specified.
You can input the number decimal or hexadecimal (see
Options
).

Buffer: The internal databuffer address. Range from 0 to 0xFFFF (64kB).
If you load/read data from a file or eprom, the data will be placed starting at this point.

Eprom : Similar to 'Buffer' but this time the eprom address is affected.

Length: The number of bytes which will be written or read.
After you load a file from disk, this shows the length of the file.
If you use the IntelHex mode, this is not the real length. In those case, 'Length' shows the highest used address.

Some examples:
Buffer = 0, Eprom = 0, Length = 10, Action: read or program.
100 Bytes will be read,written from address 0 upto 99.

Buffer = 0, Eprom = 2000, Length = 100, Action: program
100 Bytes will be written from buffer address 0 upto 99 to eprom from address 2000 upto 2099.

1.12 epromtype

Eprom type and program algorythm

There are serveral eprom and static RAM types supported. Be sure you use always the right personaltiy socket for the choosed eprom type ! Be carefully !

Eprom type:
Select 1 of the eprom types. Every eprom use a default program algorythm which is automatically set by choosing an eprom type. After you choosed an eprom type you can change the program algorythm. There is also a possibility to set the default length of any Eprom (see
Options
).

Program algorythm:
Every eprom has a default algorythm. You can change the algorythm after you choosed the eprom type.

-50ms Pulse algorythm for the original 2716,2732,2732A (the slowest)
-intel 's intelligent algorythm (perhaps the fastest for eproms)
-EPROMmer Original algorythm
-No delay A one pass write algorythm,just to use with static RAM's

Explanation:
"50ms" The first eproms have to be programmed with 50 ms pulses. Today this

method takes too much times for greater eproms, but older one (like the 2716,2732,..) need the 50ms to set up the data.

"intel" This works like the name (hmmmmhh ;-( ) - INTELligent !
Program the byte with a short time (I assume 1 ms) and check it. If the data isn't correctly, retry. If the data is correct, append a short burn (I assume again: 3ms).

Carsten: A little problem: Intel recommend that the Eprom VCC is set up to 6V during this operation. The Hardware can't do this. But Udi said it works, so I believe him (I don't checked this).

"Eprommer" Sorry, better you ask Bob (the Developer).

"No Delay" Only for static RAMs. This is my final mode during development. I take a 6264 SRAM (8kB Static RAM), solder an elctrolyt capacitor (1000uF) between VCC and GND on the back, bend the the VCC pin horizontally to insert a 1N4148 between +5V and VCC-Pin. Now you can remove the SRAM and set in the development circuit during voltage up - Yes, yes, yes, I know that's not Gentlemen like, but it works, until today I use my FIRST 6264 !!!

1.13 action

Action

During all actions, the values of address: 'Buffer', 'Eprom' and 'Length' are used.

Read: Read data from eprom into internal Databuffer.
Check: Check if an EPROM is empty.
Compare: Compare data between EPROM and internal databuffer.
Program: Program internal databuffer to eprom.

1.14 buffer

Modify internal databuffer

RAM Value: Displays or input a byte, affected is the address where 'Buffer' points ↩ to.
Read: Read a byte from address 'Buffer' and displays it
Write: Write 'RAM Value' to address 'Buffer' of internal databuffer.

If you want to change a value, you have to do 3 things:
- insert a buffer address
- insert the value in 'RAM Value'
- click on 'Write Byte'

If you want to read a value, you have to do 2 things:
- insert the buffer address

```
 - click on 'Read Byte'
```

1.15 optionswindow

```
                              Options

Numerical base:
 "Only Decimal" Inputs on 'Address' (Buffer/Eprom/Length) are in decimal notation.
 "Dec. & Hex. " Inputs on 'Address' are in decimal or hexadecimal notation.
  Hexadecimal numbers have to start with a '0x'. Decimal numbers will be
  immediately converted to hexadicmal.
 "Hexadecimal" Inputs on 'Address' (Buffer/Eprom/length) are in hexadecimal
    notation.

The next 2 options affect only data which is loaded in Dataformat="IntelHex".

 Buffer fragmented:
  Allows you to load IntelHex files with non-continous data. This is only
  implemented to remember the user what he does.
  Fragmented data is at this Version of 'Eprommer' not correctly supported
  (sorry - to much programing overhead). Eprommer programms always
  a continoues area of data. What does this mean: If you load an IntelHex file
  with 2 Databytes at 0x0010 and 0x00f1, Length is automatically set to 0x00f1.
  Programming at the next step will burn 0x00f1 Bytes , starting at Address 0
  (see Button
                Length
                ). The problem is, that there are undefined
  gaps in the data. To serve this problem, there is the next option.

 Clear Buffer 0xFF:
  Undescribed data ranges will be filled with 0xFF

 NOTE: If you switch 'Buffer fragmented' on you should always switch
       'Clear Buffer' on, except you really know what you do !
       With 'Clear Buffer' off, you are able to mix Data.

Info Requester:
 More Info-Requesters appears. This option is for users who like
 requesters like "Note, please turn BLAFASEL BLUBB NLUBB BLUBB ...."
 I don't like these Requesters.

Counting:
 If ON, the actual eprom addrress is displayed during the actions READ,
 EMPTY,COMPARE and PROGRAM. This takes a lot of CPU-Time, especially on 7MHz
 Amiga's.

Eprom default Size:
 If ON, every time you choose a new eprom type, the gadget 'Length' is set
 to the default size.

CIA Delay time:
 0    for any Amiga with 7MHz
 1-n  for any faster Amiga (my A4000/030 need 2)
```

```
 That's an internal delay time to slow down the CIA. Why is this needed ?
 You can program the CIA how fast you want - the output lines are slow !
 So after every write to CIA's you have to wait some micro seconds to be
 sure that the EPROMMER hardware noticed the new level.
 You can switch off the internal automatic speed detection if you call
 EPROMMER from CLI with a numeric argument:

 If EPROMmer don't find it's configuration File an automatic speedtest
 is performed.
OK:
 Sets the options to work with this. Don't save them in ENVARC:

Save:
 Sets the options to work with this. Writes the options to ENVARC:

Cancel:
 hmmmh,...
```

1.16 testwindow

```
            Adjust Hardware

Here you can check your hardware.

'Data' toggles your Datalines between 10101010 (0x55) and 01010101 (0xAA) .

'Address' does this also. Note that it is no Error that the Datalines
also switch if you toggle the addresslines !!! - This is given through the
eprommer hardware.

You can setup the programming voltage constant at the eprommer hardware.
This is to check it and to adjust them.

Be sure, you have removed any eprom from the socket ! Eproms will die, if
Vpp is setup a too long time - and a second isn't short for any IC :-(
To prevent damage, you have to click first:

      'Eprom removed'


 Then you can click on the voltage gadget. After you checked 'Eprom removed'
 or you choosed a new voltage, you should measure Vpp at your hardware.

If you leave this window, Vpp is automatically switched off !
```

1.17 compile

```
Compiling the source code:

I've generated the user interface with GadToolsBox 2.0c. The File is included.
```

```
If you use DICE (2.07.54R), call 'dmake', if you don't use DICE - your own mistake ↩
    .

You are free to change all, please send me a copy with a description and
an explanation of what you have done.

Thousands thanks to Bob and Udi,

  Carsten
```

1.18 intelhex

```
Diese Beschreibung basiert auf einer MÜNDLICHEN Erklärung eines Freundes,
also steinigt mich nicht, wenn was falsch ist - schickt mir lieber eine
Mitteilung was falsch ist (EMail: caro@uni-paderborn.de)

Beschreibung des IntelHex-Format's

Das IntelHex Format stellt Binärdatenfiles in ASCII-Hex dar.
In einer Textzeile können  (müßen aber nicht) bis zu 16 Datenbytes stehen.
Die Binärdaten müßen nicht fortlaufend oder aufsteigend in dem File stehen.
Dadurch ist das Format besonders geeignet zum Beschreiben von Objektcodes.

Jede Zeile besteht aus:
 - einem Anfangskennzeichen ':'
 - der Anzahl der Datenbytes
 - einer Hexadecimal 4 stelligen Adresse (gilt für das 1.Byte der Zeile)
 - einem Hexadecimal 2 stelligen Statusbyte
   (immer 0x00, ausser die Endekennzeichenzeile mit 0x01)
 - den jeweils Hexadecimal 2 stelligen Datenbytes
 - einer Hexadecimal 2 stelligen Prüfsumme

Also wie folgt:

:nnaaaassdddddd..pp

nn    Anzahl Datenbytes (2 Chars)
aaaa  Adresse       (4 Chars)
ss    Status        (2 Chars) immer 00, ausser am Ende
dddd..  Daten (nn Bytes)  (2*nn Chars)
pp    Pruefsumme      (2 Chars) 0x100 - Summe aller Bytes in der Zeile
                      (aber nicht das Anfangskennzeichen ':')

Das Ende der Datei wird mit einer besonderen Zeile markiert:

:00000001FF

Berechnung der Prüfsumme

Es werden alle Bytes der Zeile in 8 Bit Arithmetik aufaddiert. Überträge werden
ignoriert. Zu der Summe gehören also auch die 'Anzahl Datenytes', das Statusbyte
(nur Interessant für die 'Endekennzeichen'-Zeile) und die 'Startadresse' nicht
aber das Startzeichen ':'.
```

```
Anschließend wird noch eine Subtraktion durchgeführt:

   0x100 - Summe = Prüfsumme

Beispiel zum IntelHex Format:

Ab Adresse 0x1000 sind 12 Datenbytes von 1 - 12 und ab Adresse 0x1100 sind
5 Datenbytes mit 13 - 17 wie folgt:

0x1000 01020304 05060708 090A0B0C 0D0E0F10    ................
0x1010 1112

0x1100 13141516 17                            .....


Als IntelHex File sieht das dann so aus:

:101000000102030405060708090A0B0C0D0E0F1068
:021010001112CB
:05110000131415161792
:00000001FF

Viel Spass

  Carsten
```