Homebrew Computer RedBoard6809

By Favard Laurent, 2003/2013, Hobby project around 8 bits processor
Updated: 2013, May 24

Summary

Overview

The RedBoard 6809 homebrew computer is a small system Motorola 6809 8bits processor based. The system is made-up of two boards, the CPU board which hosts the processor, RAM, ROM, memory decoder and power supply regulation; the I/O board which hosts a 6821 PIA, a 6850 ACIA and clock generation for bauds speed.

RedBoard it's because PCBs are in red color.

History

These boards have been designed in 2003 years but then put aside, this project has been picked up in 2012. Because, I'm worked on a small 6809 Emulator as hobby under Mac OSX with Xcode, It was interesting to emulate these boards in the Emulator instead of any other more complicated 6809 based computer. When the Emulator started to works, I wrote a small Monitor in 6809 assembly for fun…Then, It became clear that the final step was to put this one in a real Eprom and run it on a real hardware. The loop was looped!

The CPU was working for the first time on February 1, 2013 and the full computer was alive on Februray 24, 2013.

Apple Mac Computer

I used a USB/Serial RS-232 adapter PL20xx chip based. Under OSX Lion, you need the following driver: **PL2303_Serial-USB_on_OSX_Lion.pkg**

To have a terminal program, you can use the built-in Unix command "screen /dev/tty.PL2203 9600" or any other tools like Zterm.

Beside the 6809 emulator running under OSX, I adapted a 6809 assembler from C sources to compile under Xcode .

Hardware

CPU Board #1

The goal of this board is to host the full computer system without any I/O. This board is made-up of:

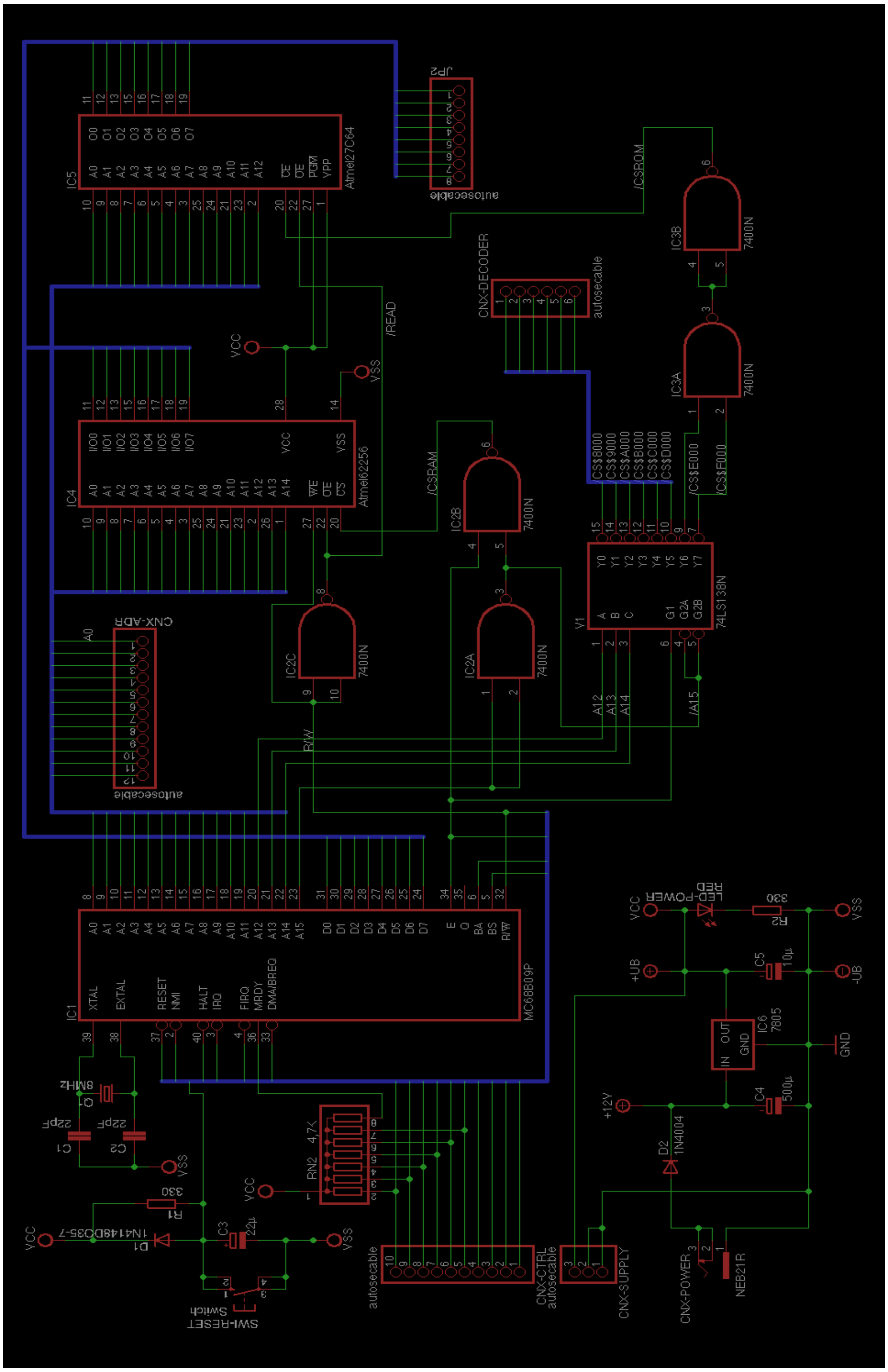
- Simple power supply regulator 7805 based
- Processor 68B09 with external crystal at 8MHz (For a bus frequency at 2MHz)
- Static RAM 62256, 32kB.
- EEPROM Atmel 28C64, 8kB.
- Memory decoder 74HCT138 (3 to 8)

I/O Board #2

This board allows the CPU to access to a peripheral parallel and serial interface.

- PIA 68B21 for general purpose usage
- ACIA 68B50 for RS-232 communication with a Max232
- Clock generator 4060 with crystal 2.4576 Mhz based for ACIA speed rate

CPU Board schematic



I/O Board schematic

Memory map

For the hardware point of view, the memory map is as follow:

$FFFF		
	EEPROM for Boot/Monitor	8kB
$E000		
$DFFF		
	I/O 6821, 6850 (Board#2]	4kB
$D000		
$CFFF		
	No hardware in this area	4kB
$C000		
$BFFF		
	No hardware in this area	4kB
$B000		
$AFFF		
	No hardware in this area	4kB
$A000		
$9FFF		
	No hardware in this area	4kB
$9000		
$8FFF		
	No hardware in this area*	4kB
$8000		
$7FFF		
	Static RAM	32kB
$0000		

*Monitor considers this area as the start of ROM expansion. Check the Monitor chapter to more information. However, for the hardware there is nothing specific.

Problem and troubleshooting

Wires and clock checking

Check that all wires are correctly done, not any unexpected wires between wrong signals. Then, with a scope, check the E signal clock on the processor (Or Q). The signal must be correct (See screenshot in Annexes)

E = crystal frequency / 4.

First program test:

We are lucky, 6809 has a great and useful instruction: **Sync**.

I used a small program (EK6809Boot1.bin) which contains a **sync** instruction**.** When the processor executes a **sync** instruction it stops its activity and wait for an external synchronization, i.e., an interrupt! In this case **BA** signal = 1 and **BS** signal = 0. I suggest having two LEDs to visualize the **BA/BS** status.

So, to check that processor is able to read the ROM, find the correct Reset vector and fetch some instruction before to stop, I burned an EEPROM with the following code:

```
            org         $E000
BootCode:   lds         #$100
            ldu         #$100

loop:       sync        ;       BA = 1 and BS = 0
            bra         loop

Vector:     rti

marque:     fcc "LAURENT BOOTCODE TEST #1, 20130201"
;       ---------------------------------------------------------------------------------------------------
            spaceto $FFF0                       ; special LFD directive: fill from last PC = * to here
            org     $FFF0
Vectors:    fdb     Vector
            fdb     Vector
            fdb     Vector
            fdb     Vector
            fdb     Vector
            fdb     Vector
            fdb     Vector
            fdb     $E000
```

<u>Troubleshooting:</u> I lived issues for a while until having BA/BS corrects. A7 address bus bit was tied to the VCC Power and A1 was not correctly connected to the RAM and ROM chips.

Second test program

I used EK6809Boot2.bin to check if the I/O board was ok. This program send a character on the serial RS-232 and the binary values %10101010 and %01010101 in order to have something that can be seeing with a scope.

Troubleshooting: When I did it, it doesn't work on the RS-232 but was Ok on the PIA. The problem was D1 data was not connected correctly between the both board and signal E was in short-circuit with D1 on the I/O board. This is why the program doesn't check the status register before to send a character.

Last tests programs

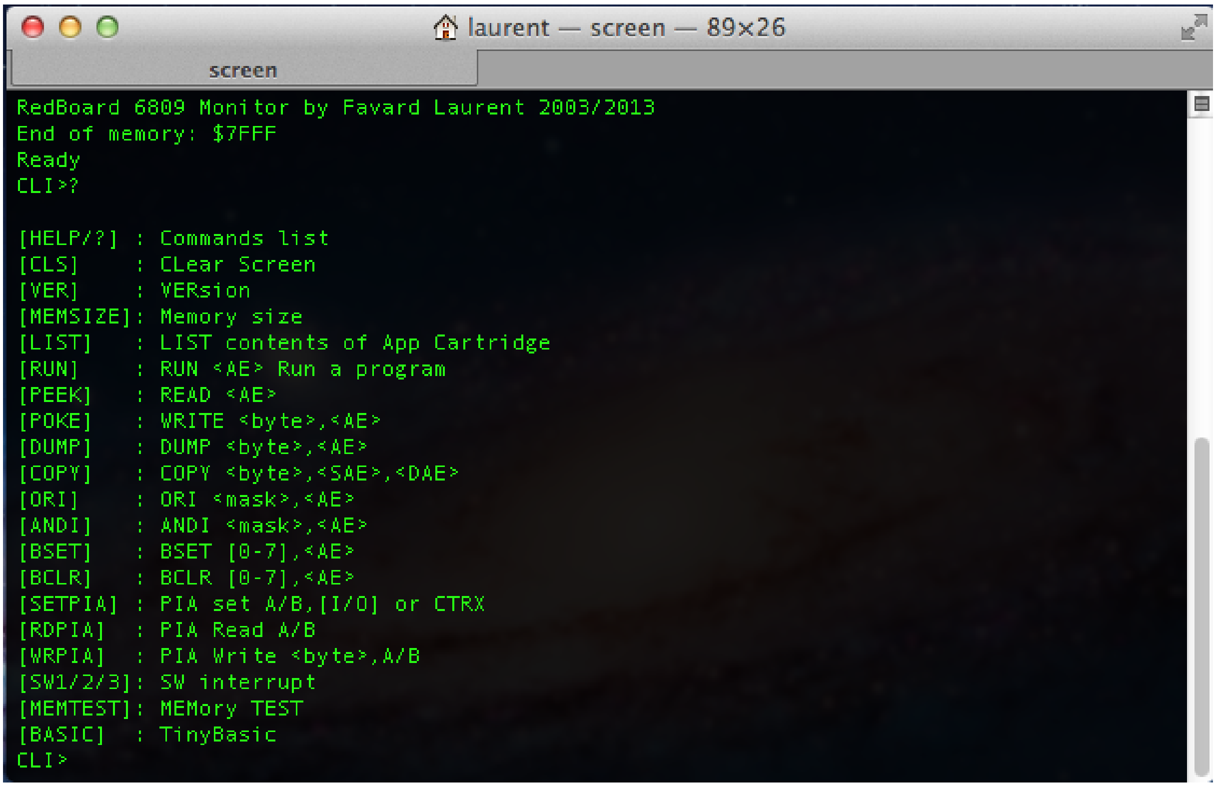
When previous issues were solved, I tried EK6809Boot3.bin to check that reading status register was ok. Then, I burned EK6809Boot4.bin which adds a simple RAM memory test.

At the end when seems to be ok, I burned the EK6809Monitor.bin and I had the great pleasure to see the monitor started exactly in the same way in the Emulator under OSX.

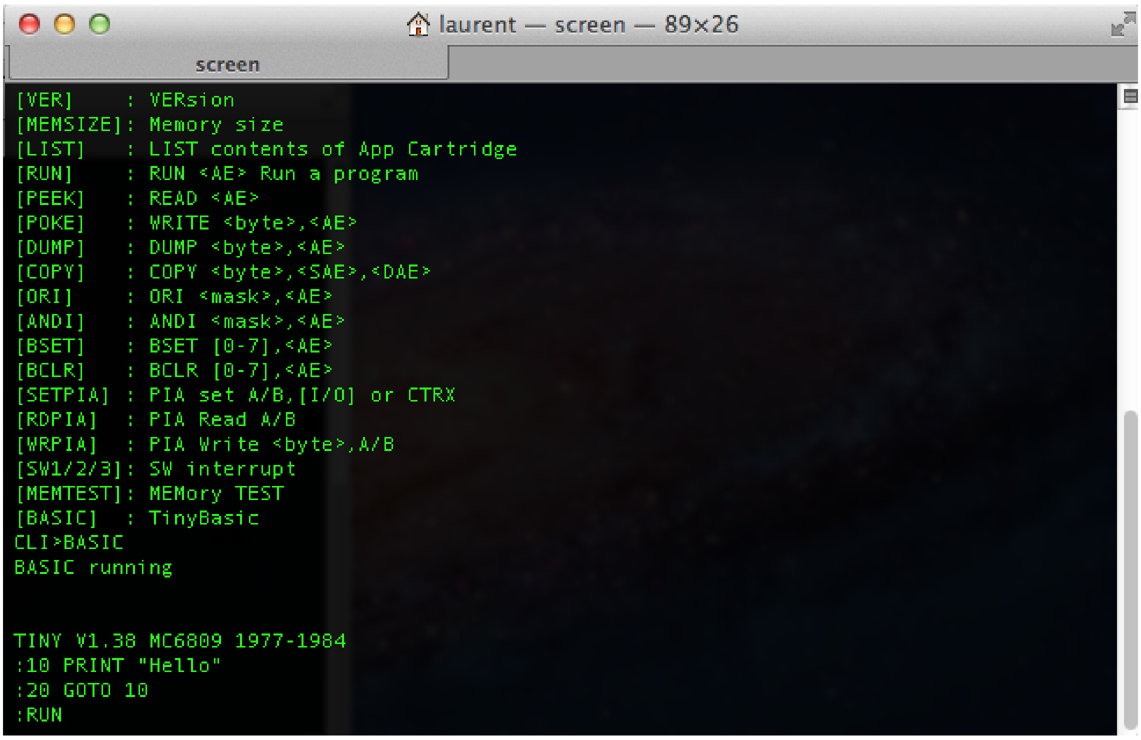
Software: EK6809Monitor

The monitor contents a minimal hardware initialization, memory checking and a set of commands in a small CLI. In addition, for fun, I added the tinyBasic in the same EPROM. Burn the EK6809Monitor.bin in a 8kB EEPROM 28C64.

TinyBasic is (C) Copyright 1977 by JOHN BYRNS



Screenshot of the Monitor started



Screenshot of the tiny basic started

Boot sequence and initialization

The monitor starts at **$E000** and immediately executes a **bra** to bypass the Monitor header:

```
            org     $E000
            bra     OSStart
            fcc     "6809"          processor code
            fcb     1,0             major,minor
            fcb     20,12,01,16     date in BCD (YY,YY,MM,DD)
            fdb     FunctionsTable  monitor functions table address
OSStart:    Boot code start here
```

1. Initialize system stack pointer to temporary value
2. Initialize user stack pointer to temporary value
3. Mask all interrupts (IRQ and FIRQ)
4. Copy in RAM the addresses of all interrupts vectors
5. Reset the ACIA
6. Set the ACIA 9600 baud, 8 bits, no parity, 1 stop
7. Check for **Diagnostic Cartridge** if [$8000] = 'D' and [$8001] = 'G'
   - If yes, load X with a return address
   - Jump at the address stored at $8002
8. Enable the interrupts (IRQ and FIRQ)
9. Check the memory from $0000 and compute the size
10. Store the RAM size at $0000
11. Store the end of RAM at $0002
12. Set the System stack point to the end of RAM
13. Set the User stack point to the below the system stack area

14. Set PIA port A and B in input mode
15. Check for **Automatic Cartridge** if [$8000] = ‘A’ and [$8001] = ‘T’
    - If yes, load X with a return address
    - Jump at the address stored at $8002
16. Check for **Applications** “cartridge” code if [$8000] = ‘A’ and [$8001] = ‘P’
    - If yes, through the list of descriptors and display each program available
17. Enter in the main CLI loop for await any user command

Monitor header

$E000	bra	OSStart	
$E002	fcc	"6809"	
$E006	fcb	1,0	Major, minor
$E008	fcb	20,13,02,01	BCD date YY,YY,MM,DD
$E00C	fdb	FunctionsTable	Monitor functions table

RAM system variables

$0000	RamSize	Size of the RAM
$0002	RamTop	Top RAM address (last address)
$0004	Swi3Vector	Vector address to SW3
$0006	Swi2Vector	Vector address to SW2
$0008	FirqVector	Vector address to FIRQ
$000A	IrqVector	Vector address to IRQ
$000C	SwiVector	Vector address to SWI
$000E	NmiVector	Vector address to NMI

Monitor functions table

FunctionsTable + 0	fdb	PutChar	
FunctionsTable + 2	fdb	PutHexChar	
FunctionsTable + 4	fdb	GetChar	
FunctionsTable + 6	fdb	GetCharUntil	
FunctionsTable + 8	fdb	WriteHexByte	
FunctionsTable + 10	fdb	WriteBinByte	
FunctionsTable + 12	fdb	WriteString	
FunctionsTable + 14	fdb	ReadString	
FunctionsTable + 16	fdb	ReadHexFromString	
FunctionsTable + 18	fdb	$0000	end of table

ROM Expansion at $8000

The monitor considers the $8000 area as a possible expansion. For that it will check the both address $8000 and $8001 for a magic number. If nothing is found, nothing it's done. The hardware set only this area to a size to 4kB in accordance to the 74HCT138 memory decoder. If more space is required, the hardware must be updated to change the default memory map decoding as done for example for the Monitor EEPROM at $E000 where two 74HCT138 outputs are combined via NAND gates.

The area isn't exclusively an EEPROM, but can be a ROM with any additional hardware…

Diagnostic Cartridge

The monitor executes an automatic code with a JMP instruction. Monitor stores in X register a return address if the program executed wants to return to the monitor.

		+0	+1
$8000	=	**'D'**	**'G'**
$8002	=	First 6809 instruction	

Go back to Monitor: **jmp 0,x**

Automatic Execution

The monitor executes an automatic code with a JMP instruction. Monitor stores in X register a return address if the program executed wants to return to the monitor.

		+0	+1
$8000	=	**'A'**	**'T'**
$8002	=	First 6809 instruction	

Go back to Monitor: **jmp 0,x**

Applications Expansion

The monitor via the RUN command will perform a **JSR** sub-routine call. So a program must finish with a **RTS** to return to the Monitor.

		+0	+1
$8000	=	**'A'**	**'P'**
$8002	=	First application descriptor	

Descriptor format:

Descriptor1 + **CA_Next**	2 bytes, address of the next descriptor or NULL if the last
Descriptor1 + **CA_Run**	2 bytes, address of the program entry (first 6809 instruction)
Descriptor1 + **CA_Init**	2 bytes, address of the init code (first 6809 instruction)
Descriptor1 + **CA_Date**	2 bytes, GEMDOS format: DDDDDDDM.MMMDDDDD
Descriptor1 + **CA_Time**	2 bytes, GEMDOS format: HHHHHMMM.MMMSSSSS
Descriptor1 + **CA_Name**	C string NULL terminated program name (Ended with '\0').

The code pointed by CA_Run and CA_Init must be terminated by a RTS instruction. CA_Init can be Null.

See in Annexes a source example.

Annexe: Oscilloscope screenshots

E clock screenshot

/CSROM waveform showing regular processor access

4060 Q4 signal: Crystal 2.457600 MHz / 16= 153600 Hz

Annexe: Source examples

Skeleton of Application Expansion ROM

```
RomCartidgeStart        equu    $8000
MonitorStart            equ     $E000
;       ------------------------------------------------------------------------------------------------
;       Offsets in ROM header
CPUCode                 equ     2
Version                 equ     6
Date                    equ     8
OffTableRoutines        equ     12
;       ------------------------------------------------------------------------------------------------
;       Offsets of subroutines in functions's Monitor
PutChar                 equ     0
PutHexChar              equ     PutChar+2
GetChar                 equ     PutHexChar+2
GetCharUntil            equ     GetChar+2
WriteHexByte            equ     GetCharUntil+2
WriteBinByte            equ     WriteHexByte+2
WriteString             equ     WriteBinByte+2
ReadString              equ     WriteString+2
ReadHexFromString       equ     ReadString+2
;       ------------------------------------------------------------------------------------------------
                        org     RomCartidgeStart
                        fcc     "AP"                    ; Applications Cartridge Header
;       ------------------------------------------------------------------------------------------------
CA_Next00:              fdb     CA_Next01
                        fdb     CARun00
                        fdb     $0000
                        fdb     %0100001001110110       ;       2013/03/22
                        fdb     %0111100000000000       ;       15h00:00
                        fcc     "Example00\0"
CARun00:                ldy     #MonitorStart           ;  find adr of functions table
                        ldy     OffTableRoutines,y      ;  Y = @ of functions table
                        ldy     WriteString,y           ;  add offset to point WriteString
                        ldx     #STRExample00           ; string to display
                        jsr     0,y
                        rts
STRExample00            fcc  "Application 00 started\015\012\0"
;       ------------------------------------------------------------------------------------------------
CA_Next01:              fdb     $0000
                        fdb     CA_Run01
                        fdb     $0000
                        fdb     %0100001001110110       ;       2013/03/22
                        fdb     %0111100111100000       ;       15h15:00
                        fcc     "Example01\0"
CA_Run01:               ldy     #MonitorStart
                        ldy     OffTableRoutines,y
                        ldy     WriteString,y           ; add offset to point WriteString
                        ldx     #STRExample01
                        jsr     0,y
                        rts
STRExample01            fcc  "Application 01 started\015\012\0"
```

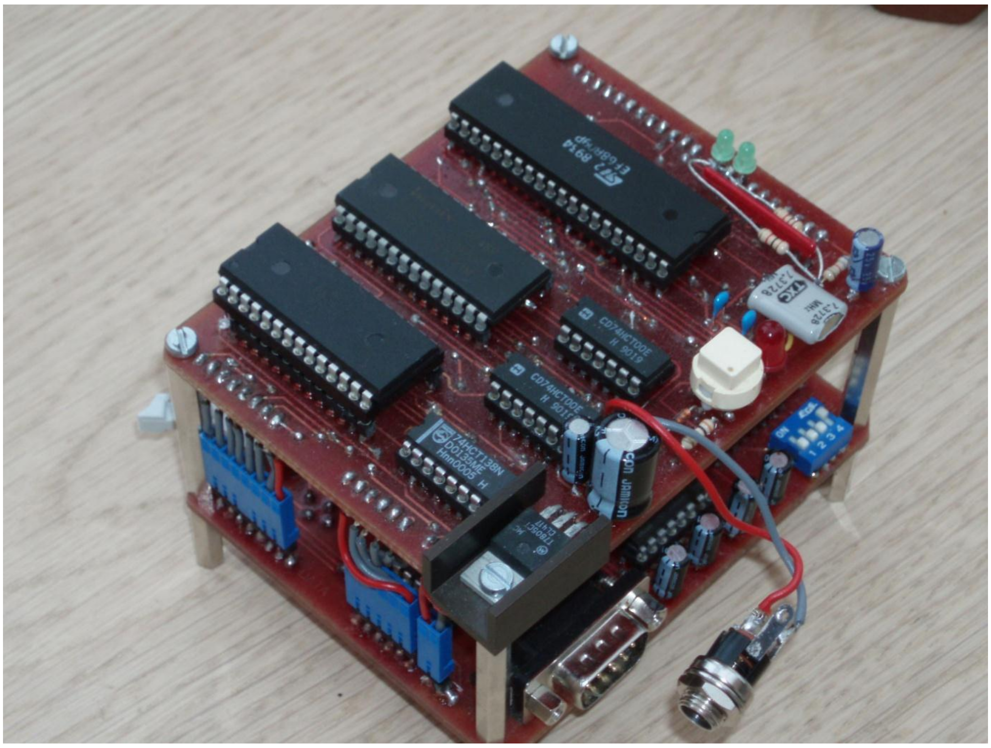
Skeleton of Automatic Expansion ROM

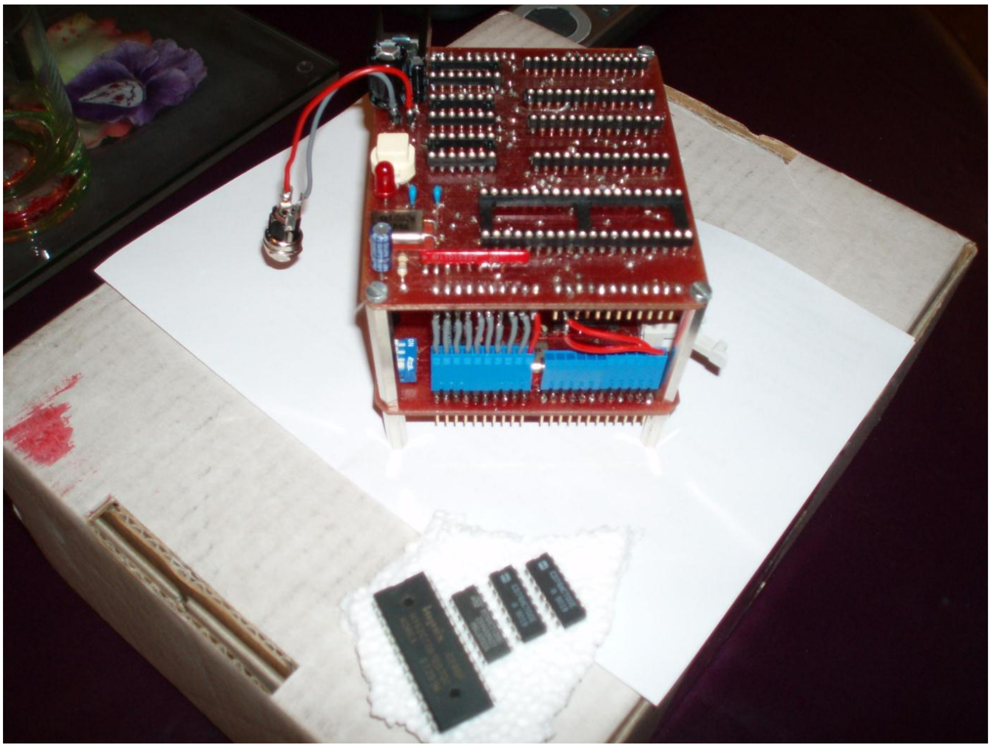
```
RomCartidgeStart        equ         $8000
MonitorStart            equ         $E000
;       -----------------------------------------------------------------------------------------------
;       Offsets in ROM header
CPUCode                 equ         2
Version                 equ         6
Date                    equ         8
OffTableRoutines        equ         12
;       -----------------------------------------------------------------------------------------------
;       Offsets of subroutines in functions's Monitor
PutChar                 equ         0
PutHexChar              equ         PutChar+2
GetChar                 equ         PutHexChar+2
GetCharUntil            equ         GetChar+2
WriteHexByte            equ         GetCharUntil+2
WriteBinByte            equ         WriteHexByte+2
WriteString             equ         WriteBinByte+2
ReadString              equ         WriteString+2
ReadHexFromString       equ         ReadString+2
;       -----------------------------------------------------------------------------------------------
                        org         RomCartidgeStart
                        fcc         "AT"                          ;Automatic Cartridge Header
;       -----------------------------------------------------------------------------------------------

Startup:                pshs    x                                 ;save return address

                        Ldy     #MonitorStart                     ;find adr of functions table
                        ldy     OffTableRoutines,y                ;Y = @ of functions table
                        ldy     WriteString,y                     ;add offset to point WriteString

                        ldx     #STRExample00                     ;string to display
                        jsr      0,y

                        puls    x                                 ;restore return address
                        jmp     0,x                               ;return to the monitor

STRExample00            fcc   "Automatic Cartridge started\015\012\0"
```

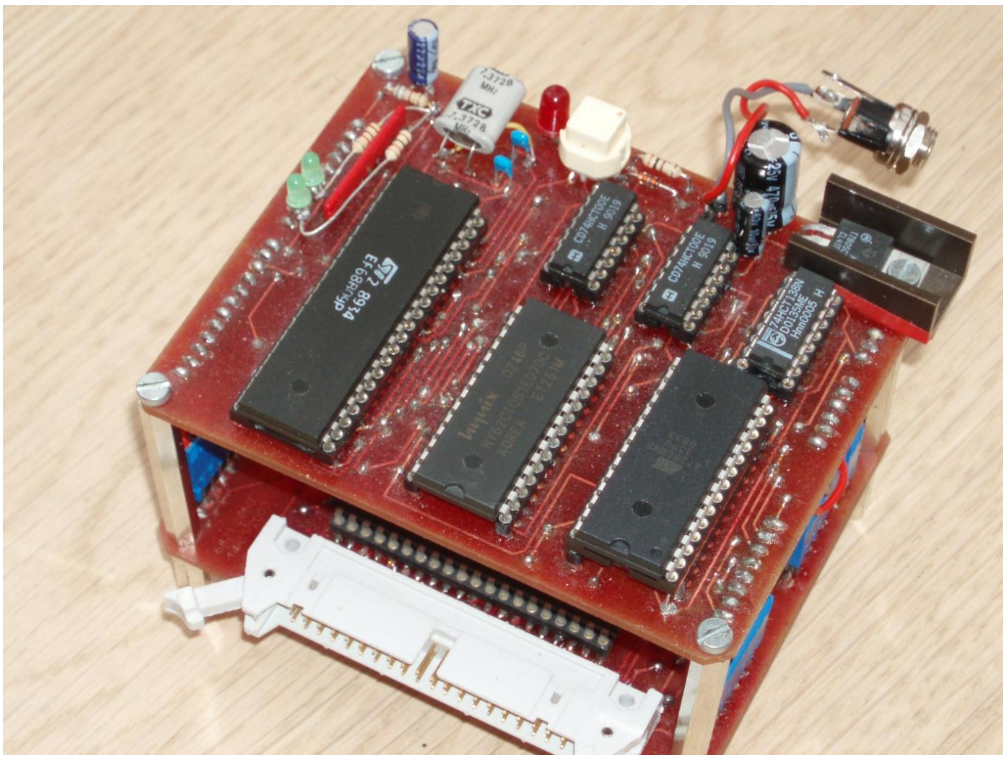
Skeleton of Diagnostic Expansion ROM

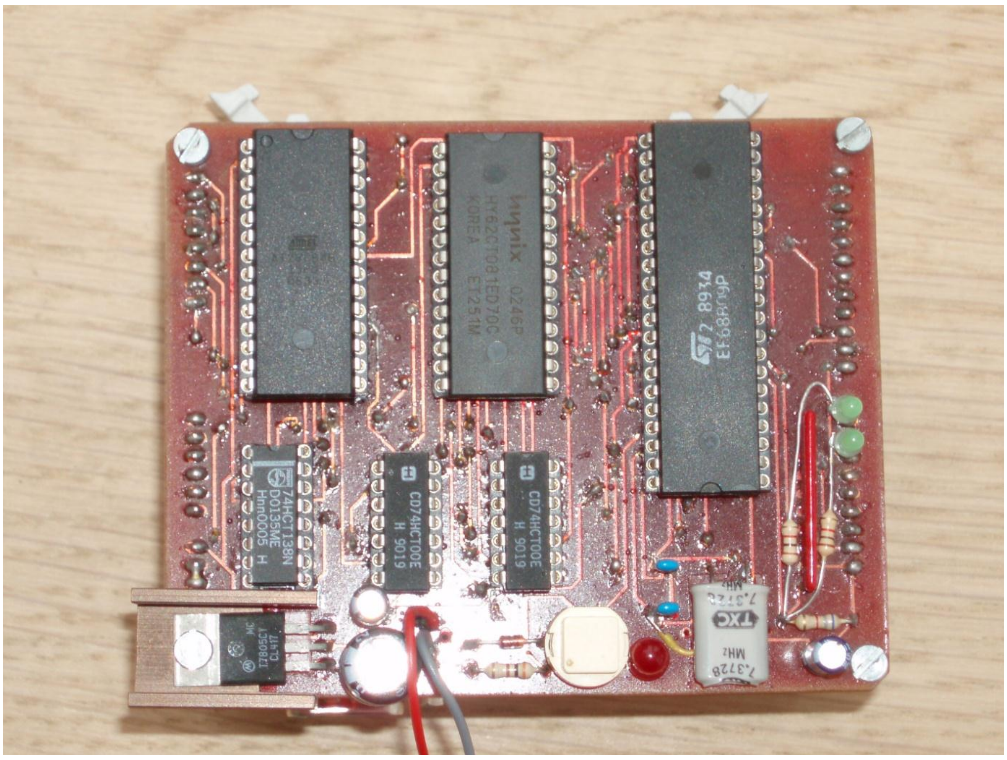
```
RomCartidgeStart        equ         $8000
MonitorStart            equ         $E000
;       ------------------------------------------------------------------------------------------
;       Offsets in ROM header
CPUCode                 equ         2
Version                 equ         6
Date                    equ         8
OffTableRoutines        equ         12
;       ------------------------------------------------------------------------------------------
;       Offsets of subroutines in functions's Monitor
PutChar                 equ         0
PutHexChar              equ         PutChar+2
GetChar                 equ         PutHexChar+2
GetCharUntil            equ         GetChar+2
WriteHexByte            equ         GetCharUntil+2
WriteBinByte            equ         WriteHexByte+2
WriteString             equ         WriteBinByte+2
ReadString              equ         WriteString+2
ReadHexFromString       equ         ReadString+2
;       ------------------------------------------------------------------------------------------
                        org         RomCartidgeStart
                        fcc         "DG"                            ;Diagnostic Cartridge Header
;       ------------------------------------------------------------------------------------------

Startup:                pshs    x                                   ;save return address

                        ldy     #MonitorStart                       ;  find adr of functions table
                        ldy     OffTableRoutines,y                  ;  Y = @ of functions table
                        ldy     WriteString,y                       ;  add offset to point WriteString

                        ldx     #STRExample00                       ;string to display
                        jsr      0,y

                        puls    x                                   ;restore return address
                        jmp     0,x                                 ;return to the monitor

STRExample00            fcc   "Diagnostic Cartridge started\015\012\0"
;       ------------------------------------------------------------------------------------------
```

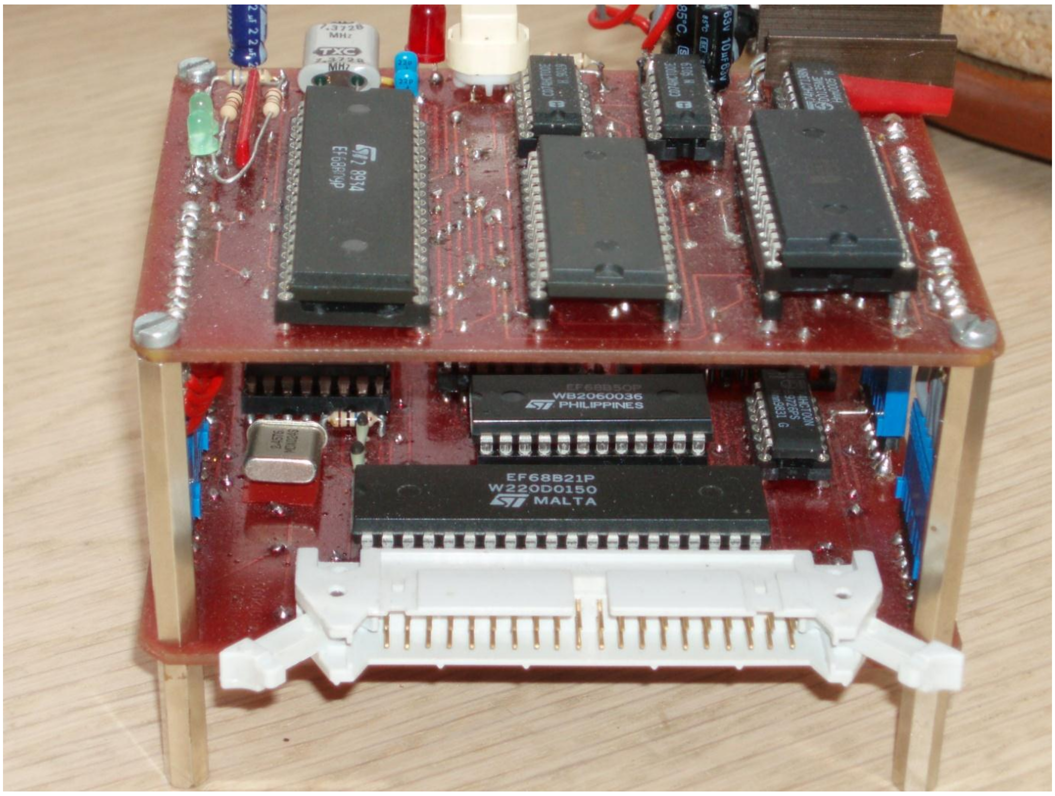
Annex: Hardware screenshots

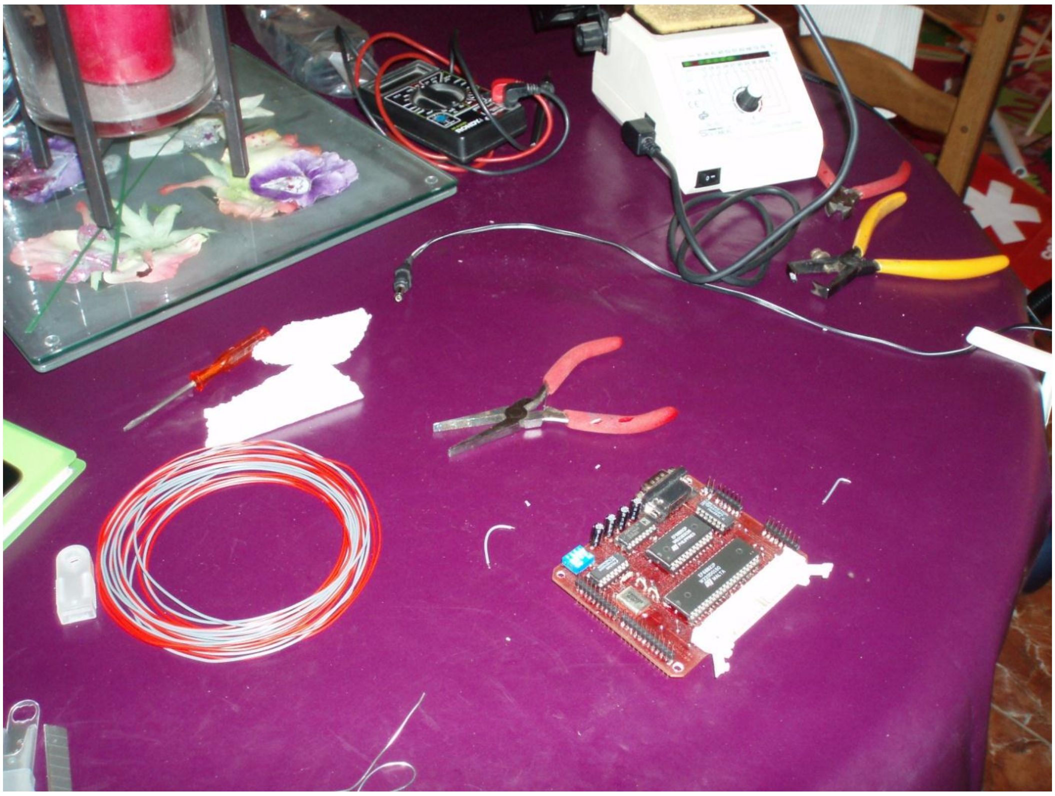


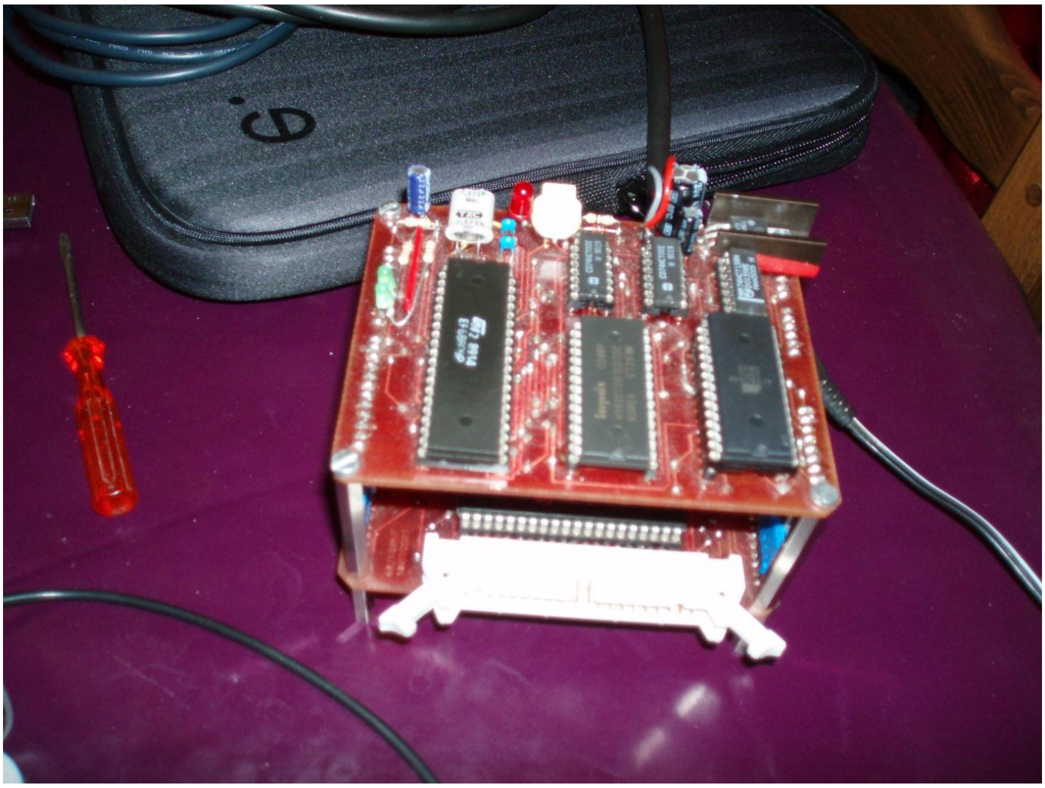


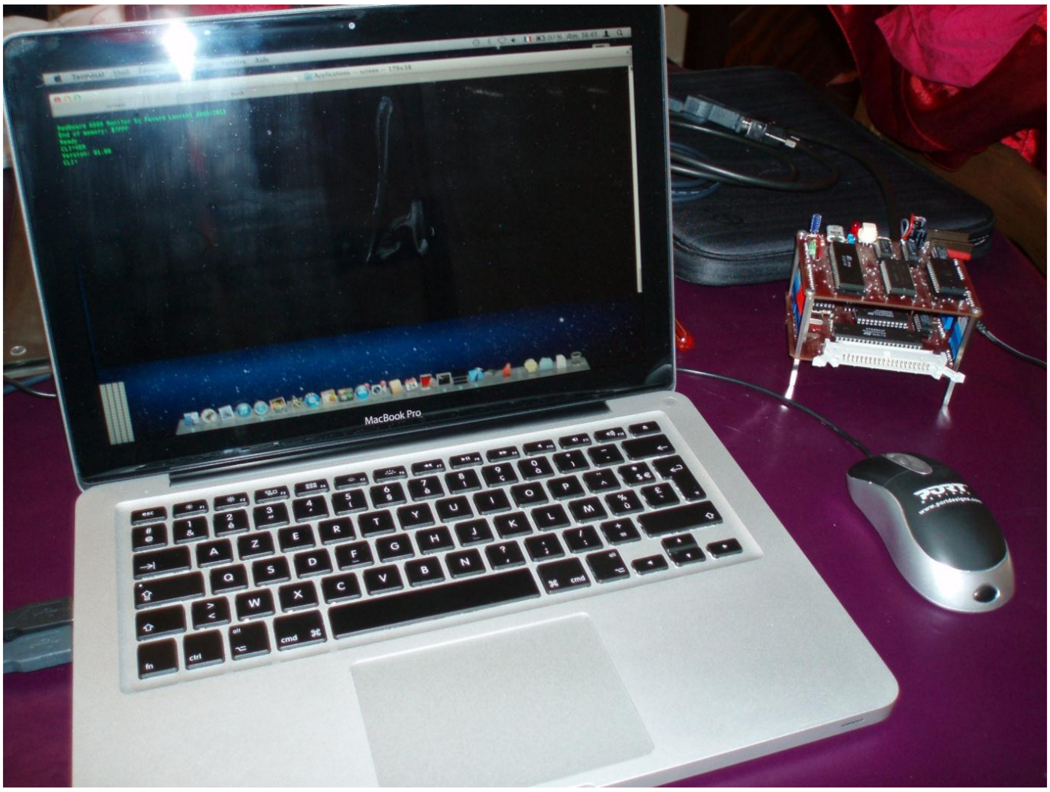



CD74HCT00E
H 9019
CD74HCT00E
H 9019
TXC

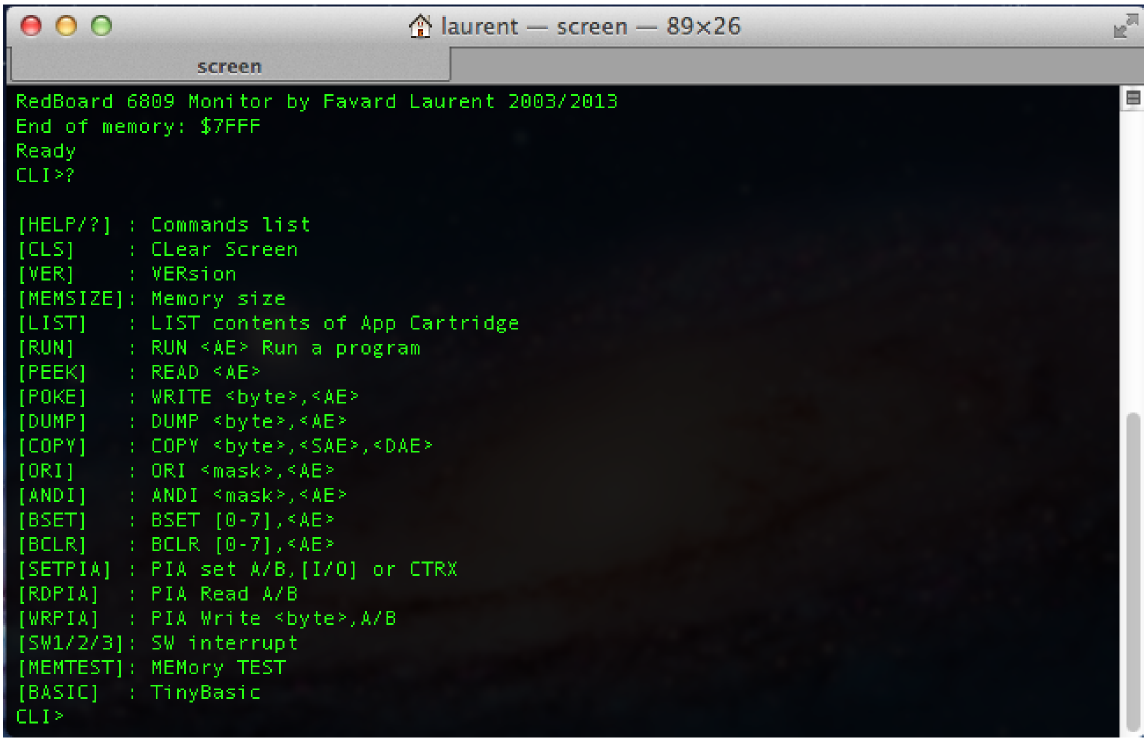

EF68B50P
WB2060036
PHILIPPINES
EF68B21P
W220D0150
MALTA



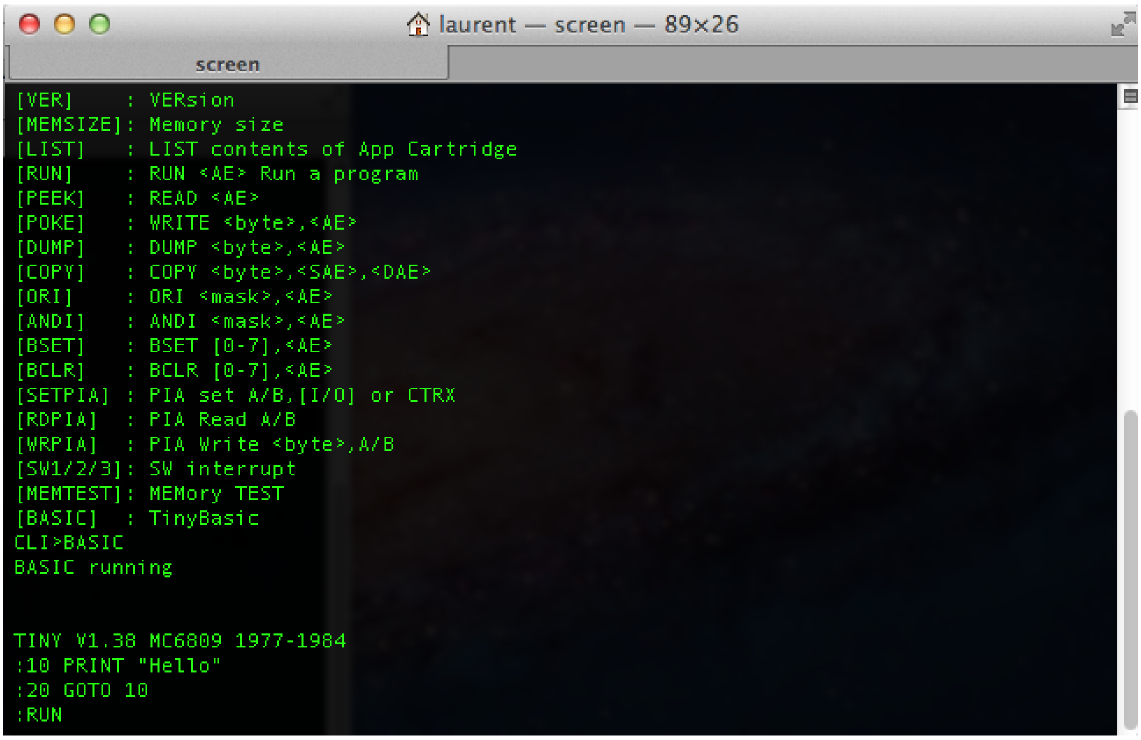



MacBook Pro

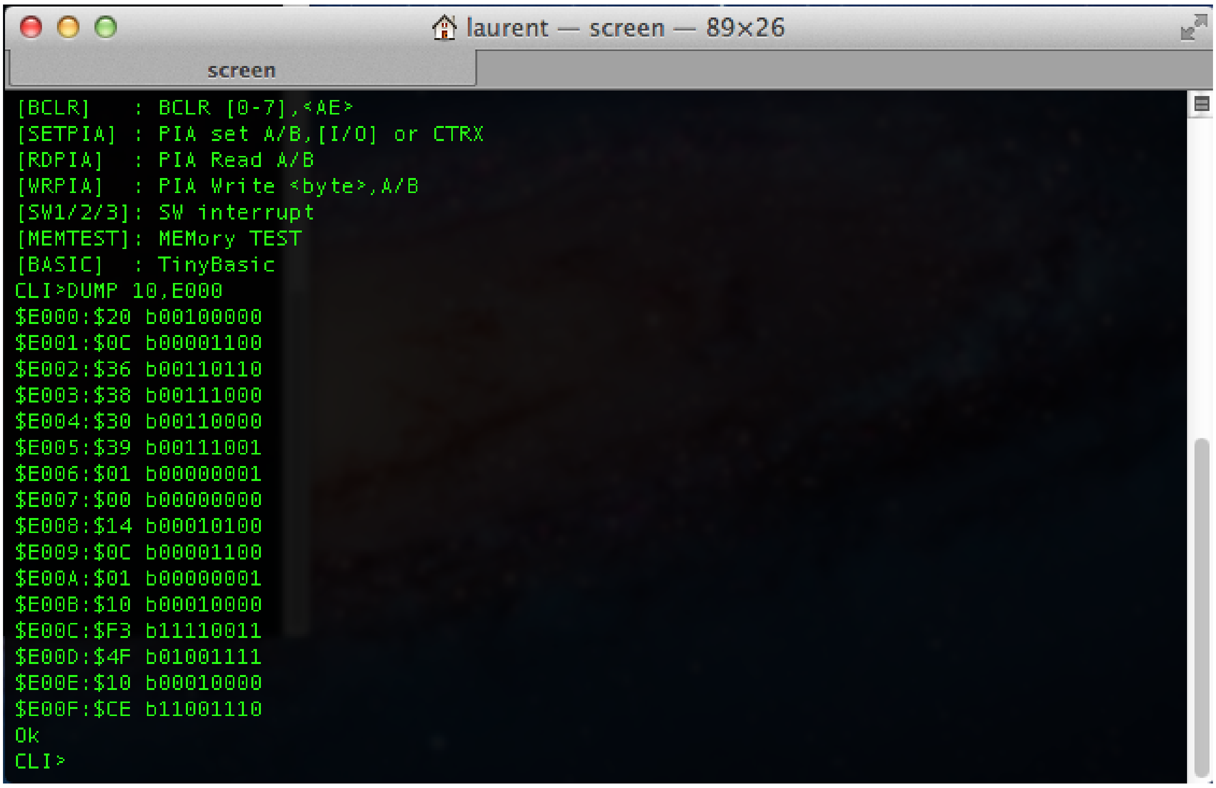
Annex: Monitor Screenshots



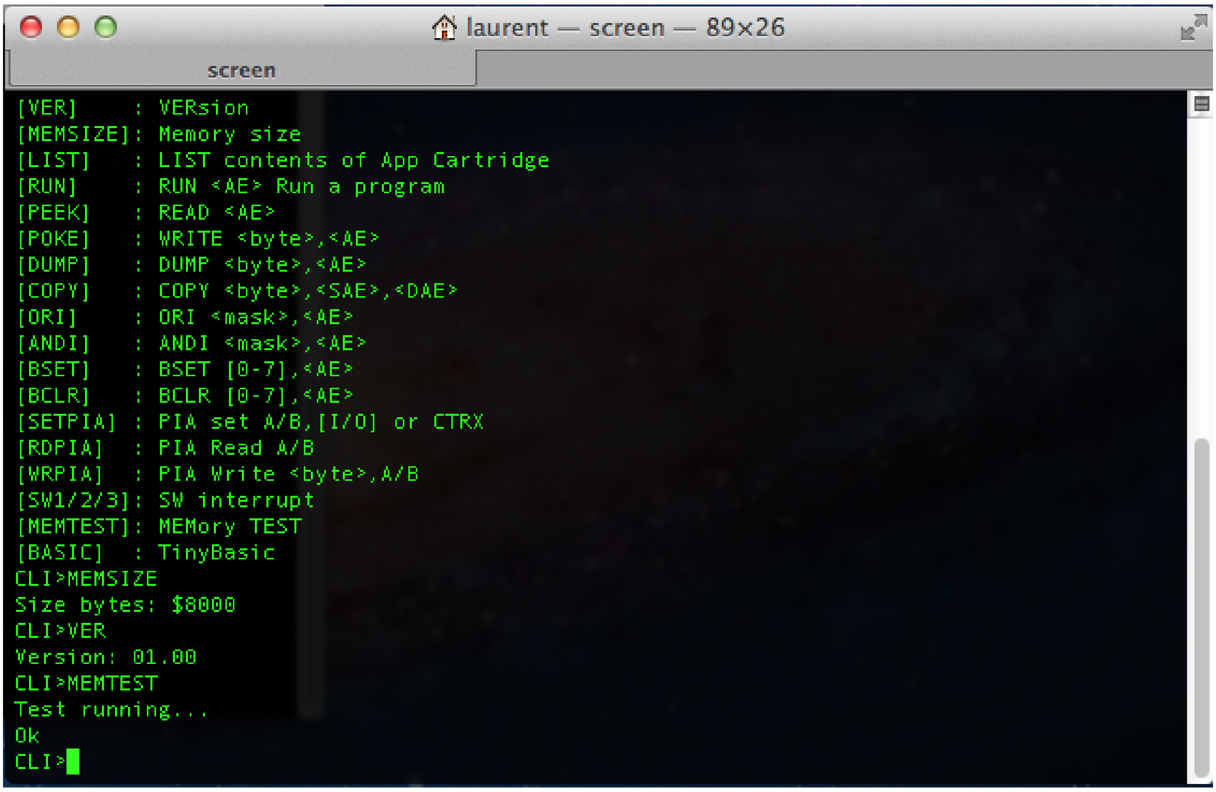
Main screen of the Monitor



Tiny Basic running



Dump command at $E000 for 16 values



Some command executed