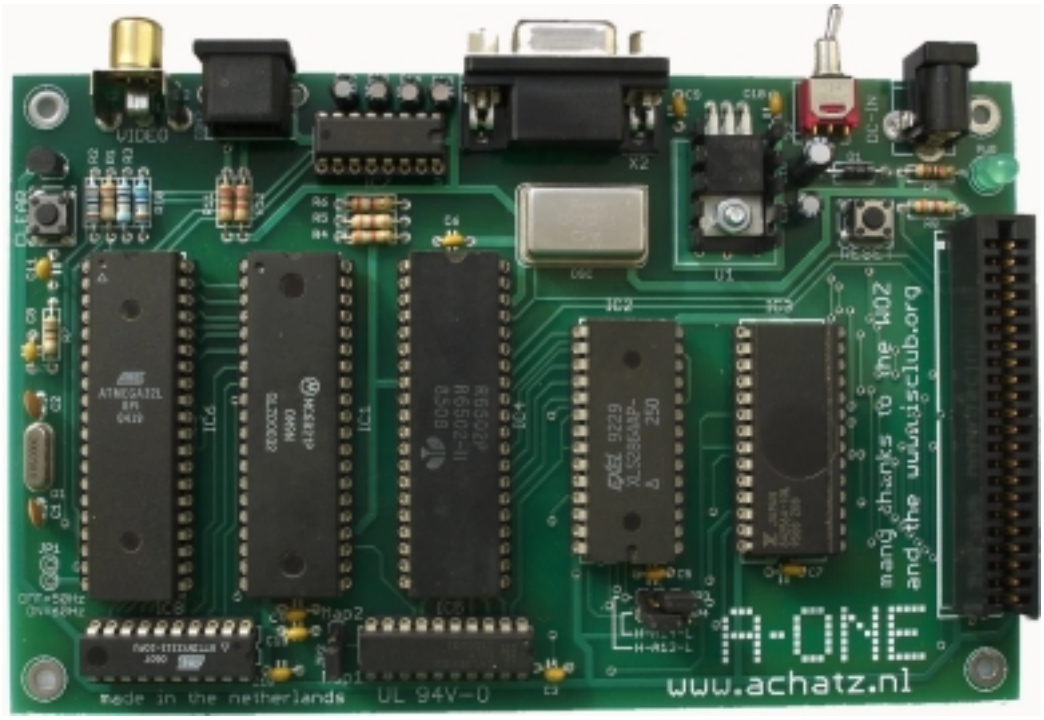
MANUAL

A-ONE

Apple 1 Replica Computer



Achatz Electronics
The Netherlands

Rel. 1.10

Published by Achatz Electronics Dec. 2006
Printed in the Netherlands

For specific information on the components mounted on the board, please refer to the Data Book of the builder or second sources.

Table of Contents

A Little Introduction To The Apple 1

Computers and humans do not speak the same language. Fortunately computers have come a long way toward learning to understand what we want them to do, thanks to graphical user interfaces, but back in 1976 computers were far from being human-friendly. They had to be spoken to in their own native language - binary numbers.

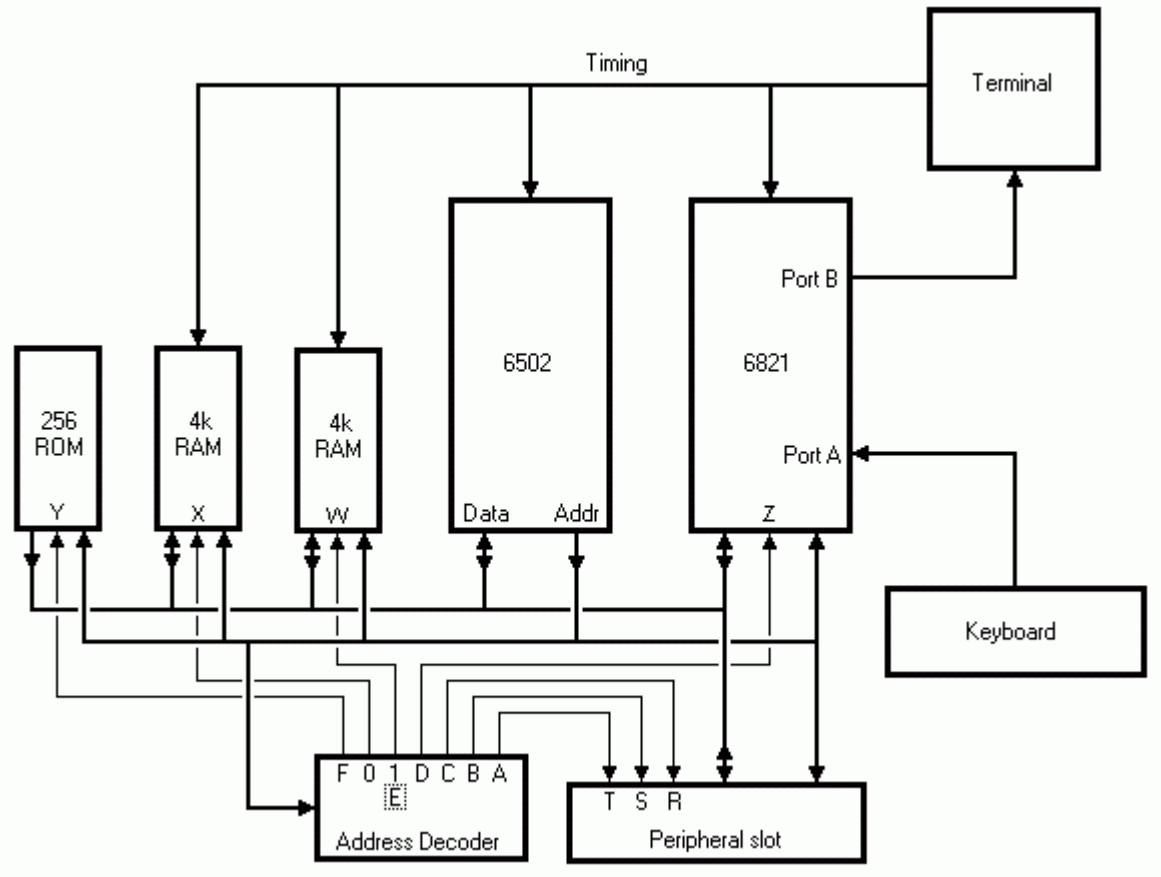
Steve Wozniak and his Apple Computer Company were among the first to change that. His Apple 1 computer came one step closer to us humans. Obviously computers still had a long way to go, but it was considered a giant leap for computer users back then.

Steve Wozniak and Steve Jobs

The Apple 1 was programmable with hexadecimal instead of binary numbers, it had a real keyboard instead of a large set of toggle switches and flashing lights, and it had a screen which could show 24 lines of 40 characters each instead of 6 seven-segment displays (at best). OK, we humans were still the ones who had to adapt the most, but from then on computers did their utmost to better integrate with us.

On the Apple 1 the human interface was controlled by a monitor program, better known as the Woz Monitor, named after its creator. A total of 256 bytes of ROM memory were responsible for holding the Woz Monitor and allowing the computer to do something sensible when the RESET switch was pressed. Doesn't sound like much, only 256 bytes, but keep in mind that memory was extremely expensive in those days. And hey, it's also 256 bytes more than Apple's main competitor at that time, the Altair 8800, which had no ROM at all and no terminal screen or keyboard!

Apple 1 Block Diagram



Here it is, the entire Apple 1 in overview. Not much of a computer by today's standards but quite reasonably sized back then. In fact it had more memory than the computer in the Apollo 11, which had landed the first man on the moon some 6 years before.

At the heart of the diagram we see the 6502, Apple 1's microprocessor. All timing is derived from the terminal circuitry. That's also where the processor's 1 MHz clock signal originates.The memory consists of 2 banks of 4k dynamic RAM and one tiny ROM with only 256 bytes, occupying another full 4k bank. The dynamic memory is refreshed by "stealing" clock pulses away from the processor. Explaining how this is done is a bit beyond the purpose of this page. Finally the "computer" part is completed by a PIA (Peripheral I/O Adapter), responsible for keyboard input and terminal output.

The keyboard is a standard ASCII keyboard. It produces a 7-bit ASCII character whenever a key is pressed. A key press is signalled by a short strobe pulse, which sets a flip-flop inside the PIA to trigger the software to read the new character from the keyboard.

The terminal is a character output device which is described at www.sbprojects.com.

Introduction To The A-ONE

My main intention was to build a personal computer that could be understood by anyone. Niklaus Wirth designed his Oberon operating system so that a person who understands programming can understand all the ins and outs of Oberon. My goal is similar - I wanted to build a computer that could be understood without having to consult many books and having to listen to many people.

So I designed the A-ONE Apple 1 replica with this thought in mind – keep it simple above all else.

There are also people who know just about everything about the original Apple 1 computer, but never got the chance to work with one. Well, here is your chance.

And for the lawyers among us: I got permission from Steve Wozniak (in writing) to use his original firmware (WOZ-MON and BASIC) for this project.

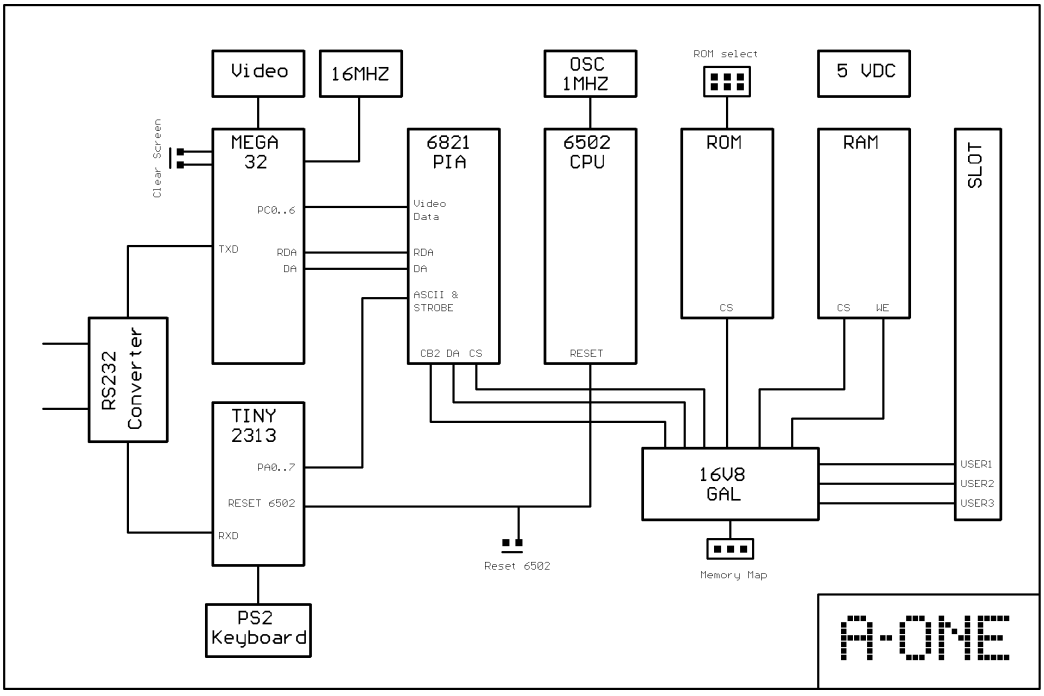
For the A-one project, I used as many original components as possible:
- 6502 CPU
- 6821 PIA
- ROM and RAM (although I used one big RAM chip instead of many small ones)

Some other generic parts were easier to redesign and produce with current technology chips. I used a 16V8 GAL for address decoding. The video display subsystem is now controlled with an Atmel ATmega32 processor and the PS/2 keyboard is handled by an Atmel Tiny2313 processor. This will enable the user to use the "old" computer with modern (and hence affordable) peripherals like a composite video screen and PS/2 keyboard.

Photo 1: The A-ONE replica

The A-ONE replica is upward compatible with the original. In addition, it has a fast RS-232 port onboard. Data transmission is handled by the ATmega32and receiving is done by the keyboard controller (Atmel Tiny2313).

A running machine



A-ONE Block Diagram

Memory Map

You can choose between two different memory maps:

Option 1:
+ 32 KB RAM located in the address range 0x000 - 0x7FFF
+ A1-Assembler in ROM starts at 0x9000
+ WOZ-BASIC in ROM starts at 0xE000
+ Krusader in ROM starts at 0xF000
+ WOZ-MON in ROM starts at 0xFF00

Option 2:
+ 4 KB of RAM, originally found at 0x6000 - 0x6FFF is moved to 0xE000 - 0xEFFF

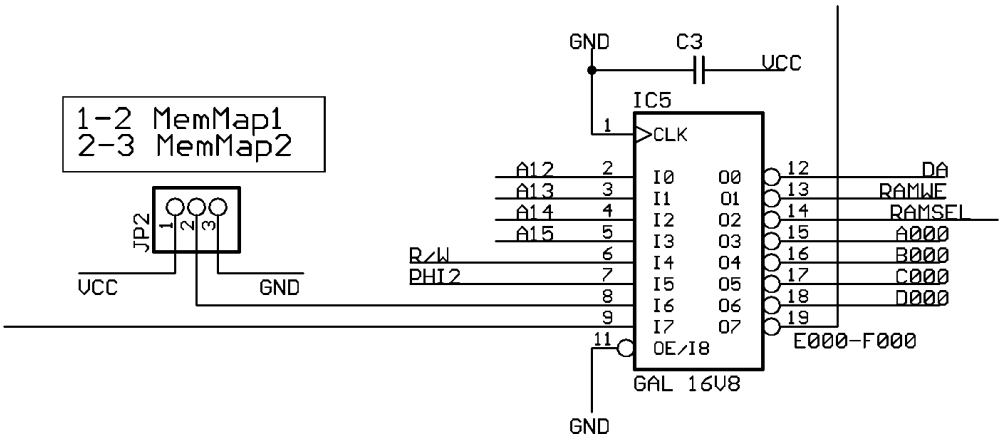
Option 2 corresponds with the original Apple-1 memory map. Since it had 4 KB of RAM at segment 0xE000, there was no room for ROM-BASIC at that address; BASIC had to be loaded through the WOZ-MON.

```
-----------------------------------------------
       JumperMode=HIGH (Map1)
-----------------------------------------------
       A15 A14 A13 A12

$0000   0   0   0   0   4K RAM
$1000   0   0   0   1   4K RAM
$2000   0   0   1   0   4K RAM
$3000   0   0   1   1   4K RAM
$4000   0   1   0   0   4K RAM
$5000   0   1   0   1   4K RAM
$6000   0   1   1   0   4K RAM
$7000   0   1   1   1   4K RAM
$8000   1   0   0   0   4K ROM USER
$9000   1   0   0   1   4K A1-ASM
$A000   1   0   1   0   4K USER
$B000   1   0   1   1   4K USER
$C000   1   1   0   0   4K USER
$D000   1   1   0   1   PIA I/O Control
$E000   1   1   1   0   4k ROM BASIC
$F000   1   1   1   1   4K Krusader & ROM WOZ-MON

-----------------------------------------------
       JumperMode=LOW (Map2)
-----------------------------------------------
       A15 A14 A13 A12

$0000   0   0   0   0   4K RAM
$1000   0   0   0   1   4K RAM
$2000   0   0   1   0   4K RAM
$3000   0   0   1   1   4K RAM
$4000   0   1   0   0   4K RAM
$5000   0   1   0   1   4K RAM
$6000   0   1   1   0   not used
$7000   0   1   1   1   4K RAM
$8000   1   0   0   0   4K ROM USER
$9000   1   0   0   1   4K A1-ASM
$A000   1   0   1   0   4K USER
$B000   1   0   1   1   4K USER
$C000   1   1   0   0   4K USER
$D000   1   1   0   1   PIA I/O Control
$E000   1   1   1   0   4k RAM
$F000   1   1   1   1   4K Krusader & ROM WOZ-MON
```



GAL 16V8 Source Code for address decoding and memory map select

```
A-ONE GAL (C) F.X.ACHATZ Version 291106

*IDENTIFICATION
    ad_dec;
*TYPE
    GAL16V8;
*PINS
    % Inputs %
    A12   =   2,
    A13   =   3,
    A14   =   4,
    A15   =   5,
    RW    =   6,
    PHI2  =   7,
    JMP   =   8,
    CB2   =   9,

    % Outputs %
    DA     =   12,
    RAMWE  =   13,
    RAMSEL =   14,
    A000   =   15,
    B000   =   16,
    C000   =   17,
    D000   =   18,
    EF00   =   19;

*BOOLEAN-EQUATIONS
    /DA = CB2;
    /RAMWE = PHI2 & /RW;
    /EF00  = (A15 & /A14 & /A13 + A15 & A14 & A13 & A12 + JMP & A15 & A14 & A13);
    /D000  = A12 & /A13 & A14 & A15;
    /C000  = /A12 & /A13 & A14 & A15;
    /B000  = A12 & A13 & /A14 & A15;
    /A000  = /A12 & A13 & /A14 & A15;
    /RAMSEL =  JMP & /A15 +
              /JMP & A15 & A14 & A13 & /A12 + /JMP & /A15 & /A13 + /JMP & /A15 & /A14 +
              /JMP & /A15 & A12;
;

*END
```

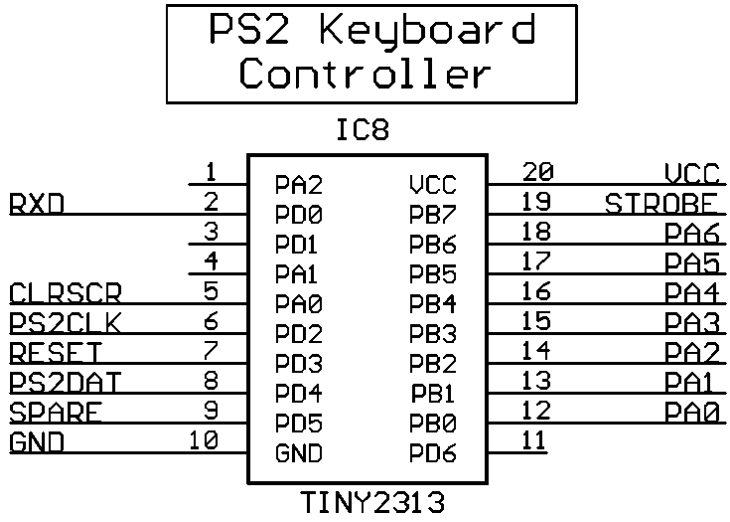
PS/2 Keyboard

The A-ONE uses an industry standard PS/2 keyboard. It is controlled via the ATtiny2313 processor. The PS/2 clock line is controlled by PortD.2 and the data line is controlled by PortD.4. PortD.3 is used for generating a RESET condition for the A-ONE processor (the 6502) either when a Ctrl-R is received via RS-232 or when the user presses the F12 function key.

PortB, bits 0 to 6, output the 7 bit keyboard data to the 6821 PIA chip.

All data that is received via RS 232 (at 2400 bps) is directed to PortB, just like the keyboard data.

Additionally, a strobe signal is generated and implemented as PortB.7 This means that a terminal session on a PC connected to the A-ONE via serial cable is treated as direct keyboard input by the A-ONE.



Video Controller

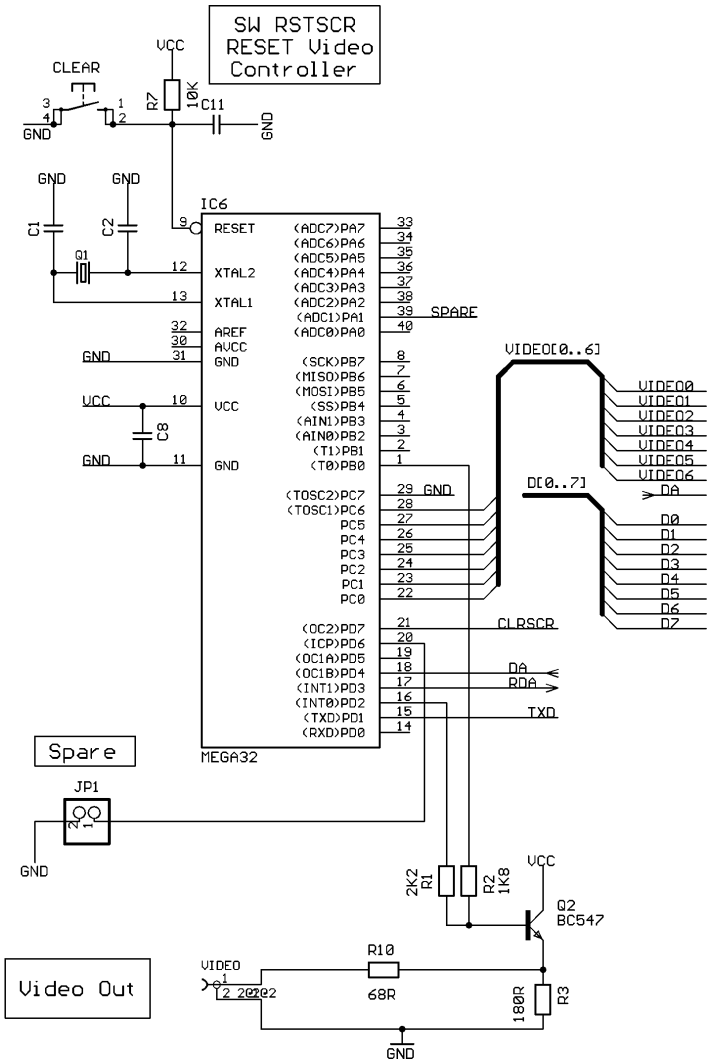
The ATmega32 serves two purposes:

- generating video data
- outputting serial data via the TxD pin (PortD.1, pin 15)

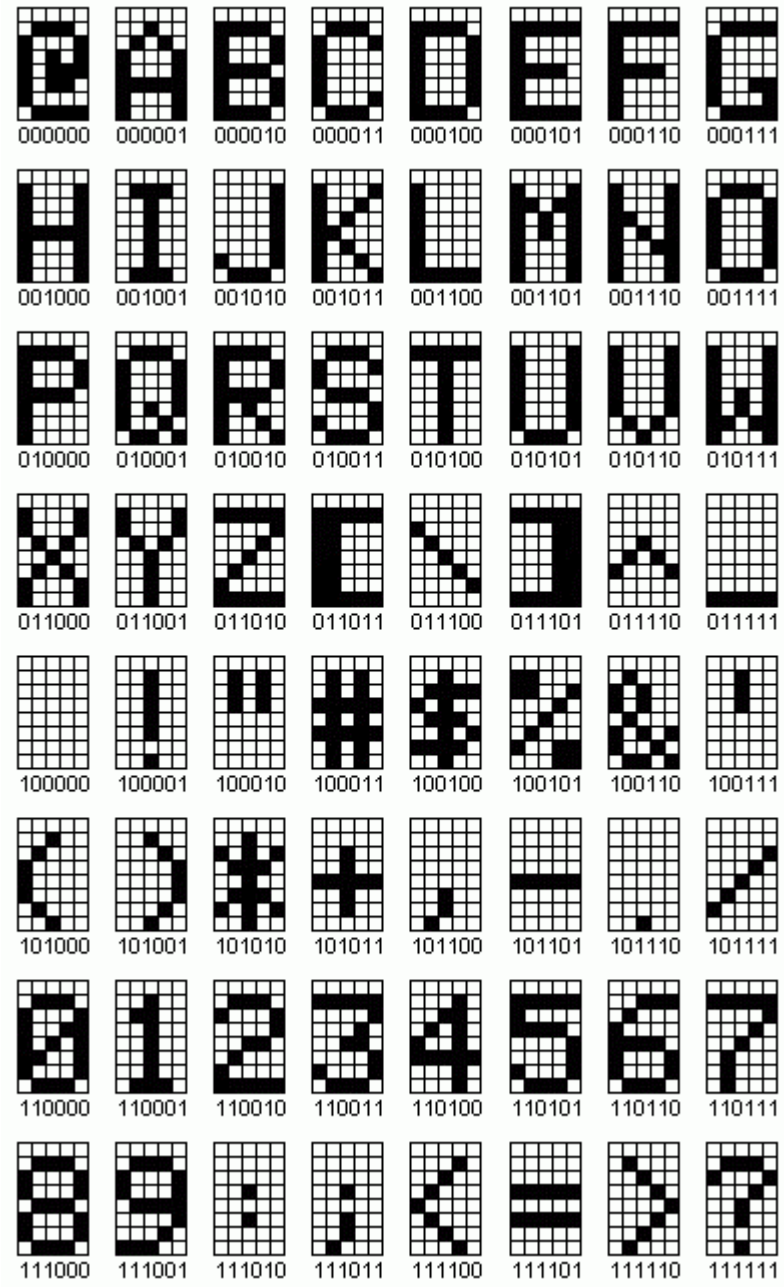
Data coming from the keyboard is retransmitted instantly via the TxD pin of the ATmega32.

The V-sync routine controls the DA and RDA handshake signals which are required for the 6821 peripheral.

Every 16.6 or 20 ms, a new character can be sent to the videocontroller, which means that maximum 50 or 60 characters per second are allowed.



Character Generator:



Working with the A-ONE

WOZ MONITOR

The original Apple 1 did not come with a reset circuit, which means that the user has to press the RESET switch in order to get the machine started. Once you do that a back slash '\' is printed on the screen and the cursor drops down one line. The cursor position is represented by a flashing '@' symbol.

You can now type address, data and commands which will be executed as soon as you press the RETURN key. The input buffer can hold up to 127 characters; if you type more characters before pressing the RETURN key the input line will be erased and will start again from scratch. This overflow situation is indicated by a new back slash after which the cursor again drops one line.

Because of the primitive nature of the terminal there are not many line-editing features available. You can press the back arrow key to erase characters from the input buffer, but the erased characters will not be erased from the screen nor will the cursor position back-up. You'll have to keep track of the changes yourself. It's obvious that you can easily get confused when a line contains too many corrections or when an error is detected all the way at the other end of the input line. In that case it would be easiest to cancel the input and start all over again. Cancelling the input is done by pressing the ESC key.

Address inputs are truncated to the least significant 4 hexadecimal digits. Data inputs are truncated to the least significant 2 hexadecimal digits. Thus entering 12345678 as an address will result in the address 5678 being used. This feature can be used to your advantage to correct typing errors instead of using the back arrow key.

If an error is made while parsing the input line then the rest of the line is simply ignored without warning even though commands executed before the error are executed normally.

Examining memory (memory dump)

You can examine the contents of a single memory location by typing a single address followed by a `RETURN`:

```
4F

004F: 0F
```

Note: The **bold** typed characters are what the user types. All other characters are responses from the Apple 1.

Now let us examine a block of memory from the last opened location to the next specified location:

```
.5A
```

```
0050: 00 01 02 03 04 05 06 07
0058: 08 09 0A
```
Note: `004F` is still considered the most recently opened location.

We can also combine the previous two examples into one command:
`4F.5A`

```
0040: 0F
0050: 00 01 02 03 04 05 06 07
0058: 08 09 0A
```
Note: Only the first location of the block `4F` is considered opened.

You can also examine several locations at once, with all addresses on one command line:
`4F 52 56`

```
004F: 0F
0052: 02
0056: 06
```
Note: `0056` is considered the most recently opened address.

Let's take this concept to the extreme and combine some block and single address examinations on one command line:
`4F.52 56 58.5A`

```
004F: 0F
0050: 00 01 02
0056: 06
0058: 08 09 0A
```
Note: By now you won't be surprised that `0058` is considered the most recently opened location.

Finally let's examine some successive blocks of memory. This can be handy if you want to examine a larger block of memory which will not fit on one monitor screen. Remember that there is no way to halt a large examine list other than hitting the RESET button!
`4F.52`

```
004F: 0F
0050: 00 01 02
.55

0053: 03 04 05
.5A

0056: 06 07
0058: 08 09 0A
```

Depositing memory (changing memory contents)

This is how to change a single memory location (provided it is RAM memory of course):
`30:A0`

```
0030: FF
```
Note: `FF` is what location `00300` contained before the operation, from now on it contains `A0`.
Location `0030` is now considered the most recently opened location.

Now we're going to deposit some more bytes in successive locations, starting from the last deposited location:

```
:A1 A2 A3 A4 A5
```

Note: Location `31` now contains `A1`, location `32` contains `A2` and so on until location `35` which now contains `A5`.
Combining these two techniques will give us the next example:

```
30:A0 A1 A2 A3 A4 A5
```

```
0030: FF
```

Note: Location `0030` used to contain `FF` in this example.
Breaking up a long entry into multiple command lines is done like this:

```
30:A1 A2
```

```
0030: FF
:A2 A3
```

```
:A4 A5
```

Note: A colon in a command means "start depositing data from the most recently deposited location, or if none, then from the most recently opened location.

Now we're going to examine a piece of memory and then deposit some new data into it:

```
30.35
```

```
0030: A0 A1 A2 A3 A4 A5
:B0 B1 B2 B3 B4 B5
```

Note: New data deposited beginning at most recently opened location, which is `0030` in this example.

Running a program

To run a program at a specified address:

```
10F0R
```

```
10F0: A9
```

Note the cursor is left immediately to the right of the displayed data; it is not returned to the next line. It's the program's responsibility to control the rest of the output.

From now on the user program is in control of the Apple 1. If the user program does not return to the Woz monitor (by jumping to address $FF1F) you'll have to press the RESET key to stop your program and return to the Woz Monitor.

You can also enter a program and run it all from the same command line. Please note that this only works for very short programs of course.

```
40: A9 0 20 EF FF 38 69 0 4C 40 0 R
```

```
40: FF
```

Note: `FF` is the previous contents of location `0040`.

This little program will continue printing characters to the screen. It can only be stopped by pressing the RESET key.

The Woz Monitor's RAM Use

The monitor needs some RAM memory to perform its tasks. When a user program is running all bytes used by the monitor may be recycled since the monitor doesn't care about the contents of any of the memory locations when it regains control again.

Here's the complete list of all the memory the Woz Monitor requires while it is running:

Zero page	**$24 to $2B**	General purpose storage locations. None of the bytes are very important and they may all be changed by the user program.
Stack page	**$0100 to $01FF**	Although the Woz Monitor only uses 3 bytes of stack space at most there is no way of telling where the stack actually is. This is because the stack pointer is not initialized by the Woz Monitor. A user program may use the entire stack for its own purposes. However be careful when entering code on page $01 before the stack pointer is initialized, the monitor may overwrite your code again.
Input buffer	**$0200 to $027F**	This space is used as input buffer. User programs may use this area. However you can not enter code here manually because it will be overwritten by the monitor.

The next addresses are not exactly RAM locations but they aren't any less important. They are the 6821 PIA control registers.

KBD	**$D010**	Keyboard input register. This register holds valid data when b7 of KBDCR is "1". Reading KBD will automatically clear b7 of KBDCR. Bit b7 of KBD is permanently tied to +5V. The monitor expects only upper case characters.
KBDCR	**$D011**	The only bit which we are interested in in this register is the read-only bit b7. It will be set by hardware whenever a key is pressed on the keyboard. It is cleared again when the KBD location is read.
DSP	**$D012**	Bits b6..b0 are the character outputs for the terminal display. Writing to this register will set b7 of DSP, which is the only input bit of this register. The terminal hardware will clear bit b7 as soon as the character is accepted. This may take up to 16.7 ms though!

$D013 This register is better left untouched; it contains no useful data for a user program. The Woz Monitor has initialized it for you. Changing the contents may kill the terminal output until you press RESET again.

Useful Routines

Apart from the monitor program itself the Woz Monitor contains only a few useful routines which can be called by user programs.

$FF1F GETLINE This is the official monitor entry point. If your program is finished and you want to return to the monitor you can simply jump to this location. It will echo a CR and from then on you are back in the monitor.

$FFEF ECHO This simple routine prints the character in the Accumulator to the terminal. The contents of the Accumulator are not disrupted; only the flag register will be changed.

Although this is a fairly short routine it may take up to some 16.7 ms before it returns control to the user program. For more details about this behaviour please read the page about the terminal.

$FFDC PRBYTE This routine prints the byte which is held in the Accumulator in hexadecimal format (2 digits). The contents of the Accumulator are disrupted.

$FFE5 PRHEX Prints the least significant 4 bits of the Accumulator in hexadecimal format (1 digit). The contents of the Accumulator are disrupted.

If you want to read a single character from the keyboard from within your own machine language program you can use the following piece of code:

```
KBDIN       LDA     KBDCR       See if there is a character available
            BPL     KBDIN       Not as long as b7 remains low
            LDA     KBD         Get the character and clear the flag
```

WOZ BASIC

I think it goes without saying that you should make sure that Apple 1 BASIC is loaded into memory before it can be started. If it's not provided in ROM you'll have to load it into memory locations from $E000 to $EFFF. The BASIC interpreter can be started by typing E000R in the Woz Monitor. After printing the contents of address E000 you'll see **">"** symbol followed by the flashing cursor, which is the prompt to enter BASIC commands.

E000 is the so-called "cold" entry point to the BASIC interpreter, which means that the program is initialized before we can really get going. This also means that there will be no runnable program in memory when we enter BASIC this way.

If you go back to the Woz monitor and want to return to the BASIC interpreter without losing anything, you can use the "warm" entry point by typing E2B3R in the Woz Monitor.

Starting BASIC using its cold entry point initializes the LOMEM and HIMEM pointers to $0800 and $1000 respectively (HIMEM is set to 4k + 1). This means that you'll have a total of 2k of memory for your BASIC program and its variables.

On a standard Apple 1 with 8k of RAM, you could lower LOMEM to $0300, giving you an extra 256 bytes of program memory. If you have more than 8k of RAM, which is probably the case with all modern day replicas and Apple 1 emulation programs, then you can raise HIMEM to whatever memory size you've got. You must have a contiguous block of memory between LOMEM and HIMEM.

Changing LOMEM and HIMEM is done with two similarly named commands. Please note that these two commands only accept decimal values and that they do NOT test whether your entries are valid. Also note that these commands will erase your existing Basic program from memory!

To run BASIC, first make sure jumper JP2 selects Map1 (we use the BASIC in ROM).

Reset the A-ONE by pressing the Reset-switch or "F12" on the keyboard.

We are ready to start BASIC. We use the same conventions used earlier in the machine language section – user typed information is shown in **boldface** while the computer response is shown in normal weight text.

You type
E000R

The A-ONE responds:
E000: 4C
>

The A-ONE is showing the BASIC-Prompt ">"

You type:
PRINT 15+35

BASIC responds:
50
>

You type:
10 A=1
20 PRINT A
RUN

BASIC responds

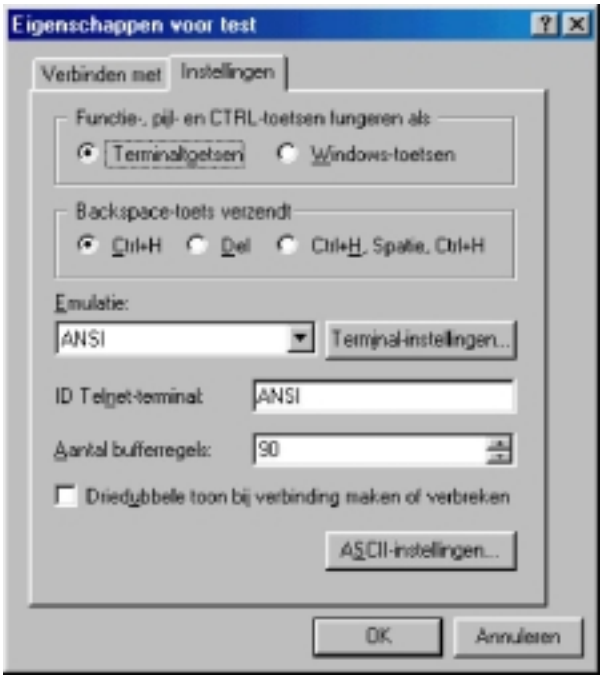
1
***END ERR

For a detailed description of all BASIC commands and for using the Monitor please consult the links at the end of this document

Hyperterminal settings:

Select the correct COM port on your personal computer

2400 baud, 8 bits, no parity, 1 stop bit, no handshake



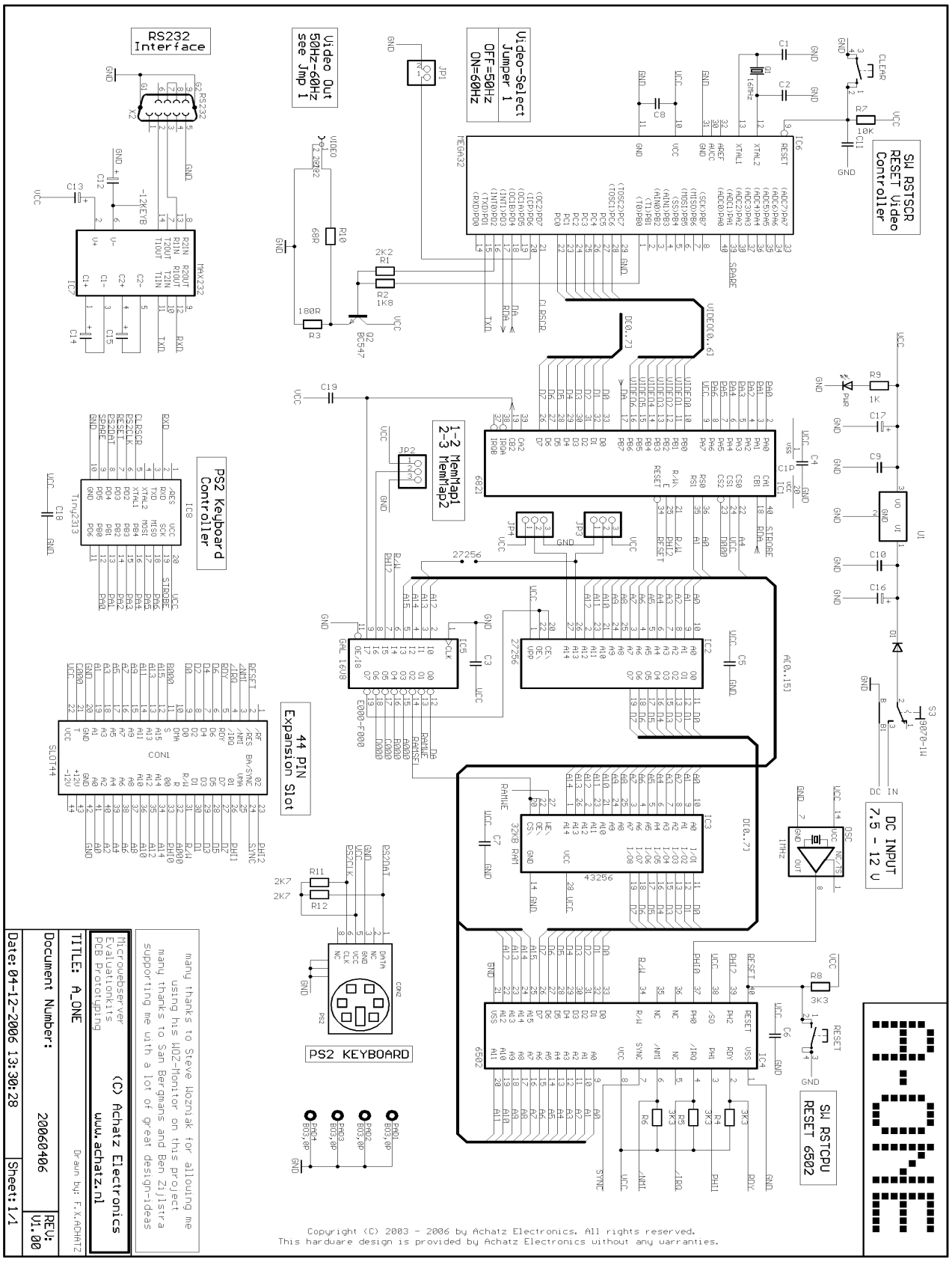
Important: line-delay and char-delay must be set to 500ms and 5ms

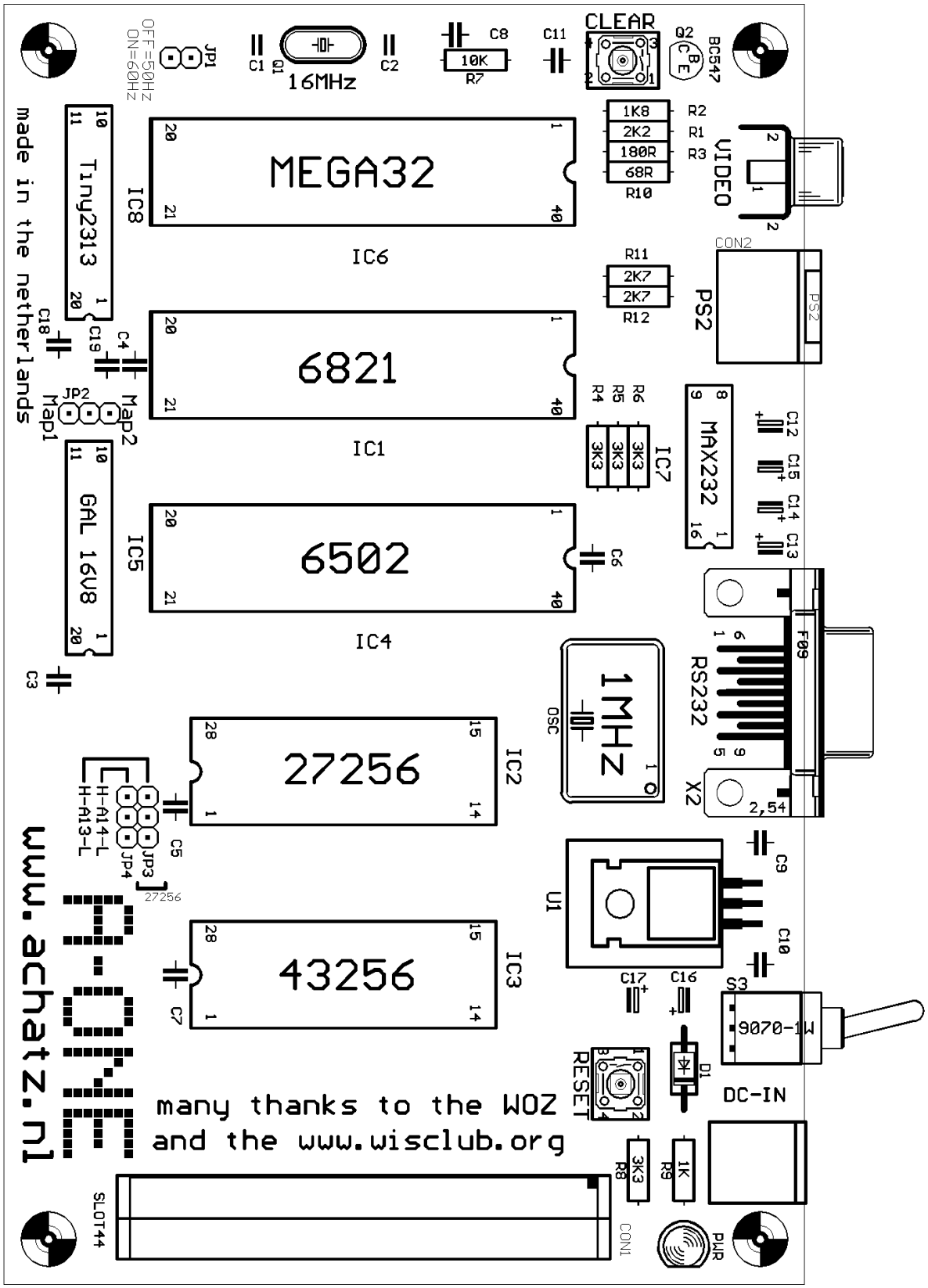
Make sure to use a normal 1:1 serial cable male-female with 9 pins. You can also operate the A-ONE via a terminal in exactly the same way as using a PS/2 keyboard.

Technical Data:

CPU:	MOS Technology 6502
System-Clock:	1 MHZ
RAM Memory:	32 KB SRAM standard (optional 32 KB ZeroPowerRAM) (optional 32 KB Time-Keeper-RAM)
ROM Memory:	32 KB standard 8/16/32KB types supported using jumper select
Video Controller:	ATMEGA32-16 40 chars/line at 24 video lines, 50/60 chars/per second at max, with automatic scrolling and Clear-Screen function
Video Output:	Composite positive video, 75 ohms, frame rate 50 Hz or 60Hz
Keyboard Controller:	ATTiny2313 (8MHZ internal) Software designed for a standard PS/2 keyboard using a US keyboard layout, 6502 Reset-Function when receiving a CTRL-R
Address Decoding:	GAL16V8 Two memory maps available using jumper select
RS232:	9-pin SubD female connector, 2400 baud transmit/receive
Expansion Connector:	2x22 edge connector (Apple 1 compatible)
Switches:	RESET switch for 6502 CPU, CLEAR switch for Video
Power Supply:	Universial AC-DC Adapter
Voltage:	DC 9 – 12 VDC
Supply Current:	250 mA
PCB-Size:	160 x 100 mm (Euro-Format)

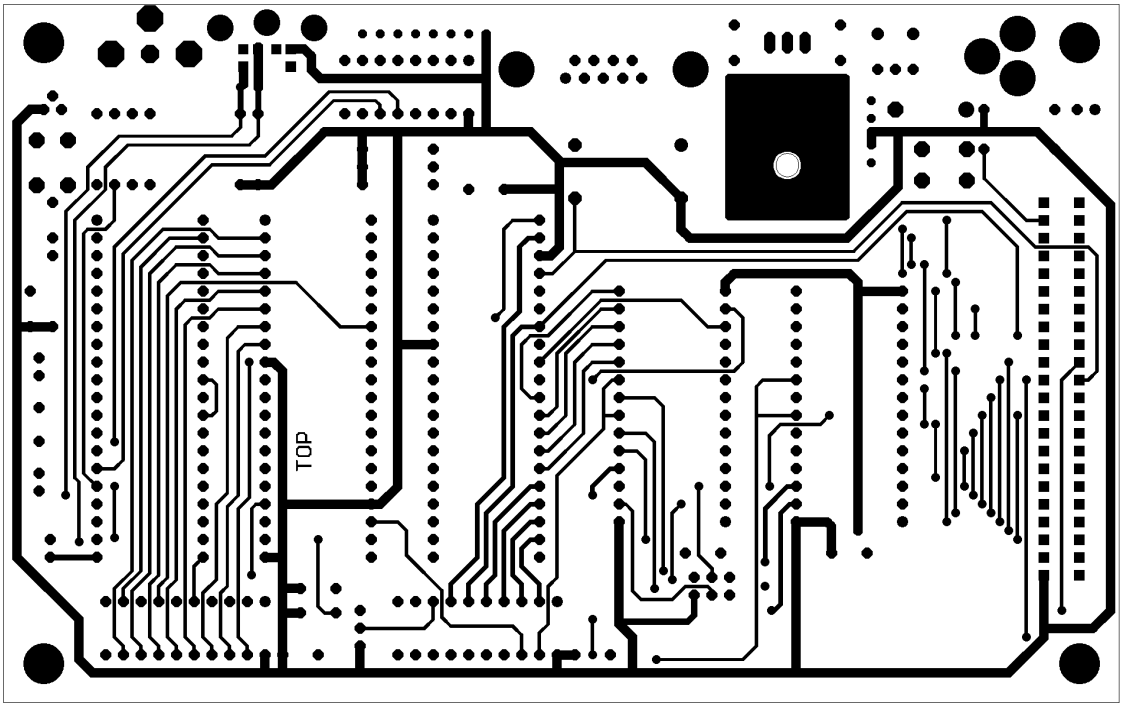
A-ONE Schematic

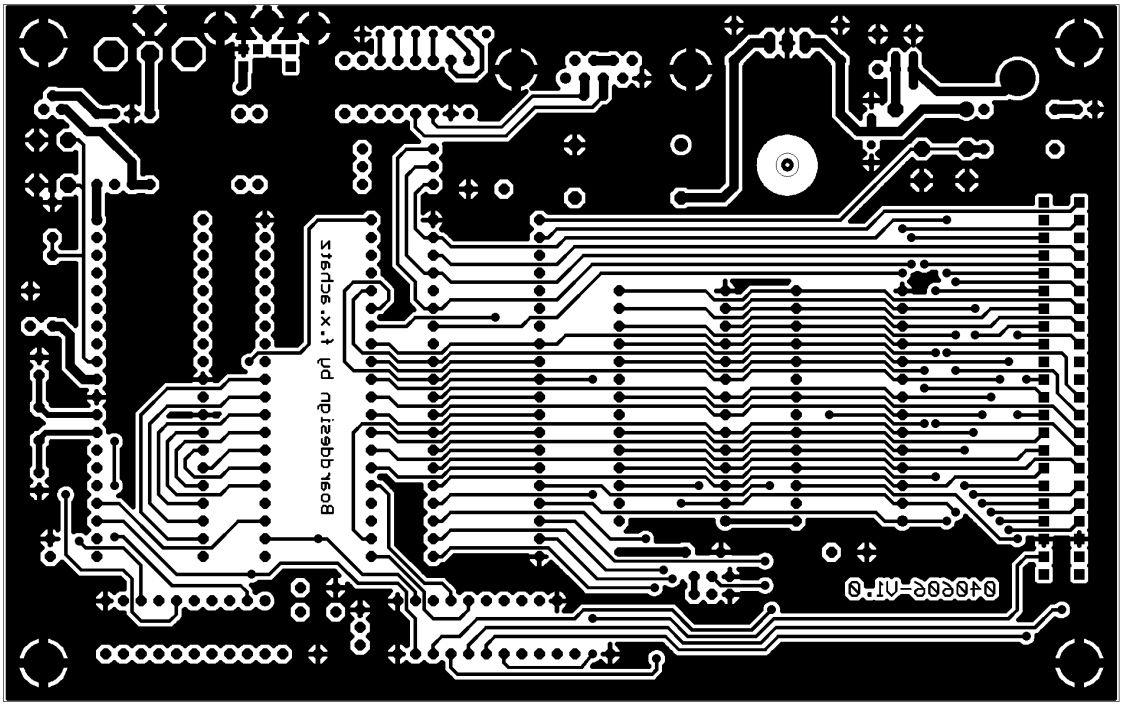




A-ONE COMPONENT SIDE

A-ONE PCB:





Parts list:

C1	22pf
C2	22pf
C3	100nf
C4	100nf
C5	100nf
C6	100nf
C7	100nf
C8	100nf
C9	100nf
C10	100nf
C11	100nf
C12	10uf
C13	10uf
C14	10uf
C15	10uf
C16	47uf
C17	10uf
C18	100nf
C19	100nf
R1	4K7
R2	2K7
R3	100R
R4	3K3/3K9
R5	3K3/3K9
R6	3K3/3K9
R7	10K
R8	3K3
R9	1K
R10	68R
R11	2K7
R12	2K7

IC1	6821 PIA
IC2	27256 EPROM
IC3	43256 RAM
IC4	6502 CPU
IC5	GAL 16V8
IC6	ATMEGA32-16
IC7	MAX232
IC8	ATTiny2313
U1	7805 with heatsink
D1	1N4004
Q1	16MHz
Q2	BC547
OSC	1MHZ
PWR	LED 5mm
JP1	1X02
JP2	1X03
JP3	1X03
JP4	1X03
CON1	SLOT2x22
CON2	PS2
VIDEO	TOBU3
X2	F09HP
RESET	Key OMRON
CLEAR	Key OMRON
S3	SWITCH ON/OFF
IC socket	40 pins (3pcs)
IC socket	28 pins (2pcs)
IC socket	20 pins (2pcs)
IC socket	16 pins (1pcs)

The A-ONE is shipped with SRAM (IC3). In place of the SRAM, a NVRAM could be used. One such RAM is the M48Z35Y-70. Alternatively a M48T35Y-70 (TimeKeeperRAM) could be installed to give you a realtime clock on the A-ONE.

Please handle the A-ONE board and/or the micro-chips with care.

Follow the instructions for using Electrostatic Sensitive Devices (ESD)

Assembling the Kit version:

Step 1:
Locate all delivered parts. Check that everything is in the bag (use the parts list).

Step 2:
Install and solder the resistors (R1 - R12) and the diode (D1).

Step 3:
Install and solder the IC sockets.

Step 4:
Solder the 100nf and the 22pf capacitors (C1 – C11,C18,C19).

Step 5:
Install the pin headers and the crystals. Take care with the 1 MHZ oscillator. The direction of this part is important.

Step 6:
Complete the board by installing the connectors, switches, regulator with heatsink, electrolytic capacitors (C12-C17) and the LED.

Step 7:
Check your work. Re-check the right values of the parts.

Step 8:
Before installing the IC's use an AC-DC power adapter, switch it to 9-12VDC and power-up the A-ONE board. The LED should light and nothing should get hot or smoke. Be sure to disconnect the power adapter before continuing!!!

Step 9:
Carefully install the IC's in the correct sockets.

Step 10:
Re-check your work again. Be sure that the chips are installed with the correct orientation (pin 1 on the chip must match pin 1 on the socket).

Step 11:
Connect a PS/2 Keyboard to the PS/2 connector and connect a composite TV or a composite video monitor to the cinch connector (video-connector).

Switch on the monitor and power up the A-ONE board.

A welcome-message should appear on the video-screen. Now press the A-ONE Reset-button and a "\" should come up on the first line, followed by a blinking "@" on the second line.

It's time to say, "Congratulations, you did a great job"!

Special Functions:

PS/2 KEYBOARD:

[ESC] = exit auto line numbering (SB Assembler)
[SHIFT]+[ESC} = exit auto line numbering (WOZ-BASIC)
[F1] = CLEAR SCREEN
[F12] = RESET 6502 CPU

HYPER TERMINAL:

[ESC] = exit auto line numbering (SB Assembler)
[CRTL]+[D] = exit auto line numbering (BASIC)
[CTRL]+[L] = CLEAR SCREEN
[CTRL]+[R] = RESET 6502 CPU

CLEAR SCREEN

The Apple 1 does not support a software-controlled clear-screen function and the video-output is scrolling. We have implemented a clear-screen function in the video-controller by sending a HEX [0C].

Example:

Load this sequence into RAM:

0000: A9 0C 2C 12 D0 30 FB 8D
0008: 12 D0 60

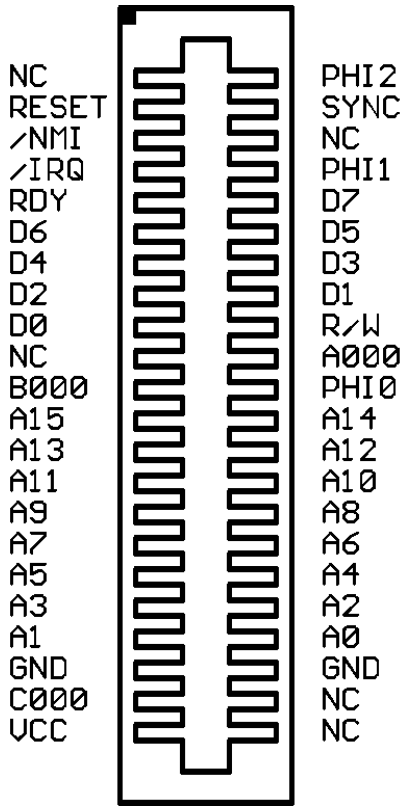
Make a [CALL 0] within BASIC for Clear Screen and Cursor Home

JUMPER SETTINGS

JP1 Memory Map
JP2 Video select 50Hz or 60 Hz
JP3 not used for EPROM 27256
JP4 A14–L (A1-ASM and Krusader running with a 6502 CPU)
JP4 H–A14 (A1-ASM and Krusader running with a 65C02 CPU)

Expansion Connector:

The A-ONE can be expanded with I/O devices via a 44 pin edge connector.
All address lines, data lines, control signals and clock signals are available.



Top view

There are 3 User Select Lines available:

Signal A000 uses address range $A000 - $AFFF
Signal B000 uses address range $B000 - $BFFF
Signal C000 uses address range $C000 - $CFFF

System Monitor:

The Hex Monitor is a program in location FF00 to FFFF (hex) and is entered by pressing RESET. A backslash is displayed when the monitor loads. The monitor performs front panel functions such as examining memory and running programs.

```
FF00: D8 58 A0 7F 8C 12 D0 A9
FF08: A7 8D 11 D0 8D 13 D0 C9
FF10: DF F0 13 C9 9B F0 03 C8
FF18: 10 0F A9 DC 20 EF FF A9
FF20: 8D 20 EF FF A0 01 88 30
FF28: F6 AD 11 D0 10 FB AD 10
FF30: D0 99 00 02 20 EF FF C9
FF38: 8D D0 D4 A0 FF A9 00 AA
FF40: 0A 85 2B C8 B9 00 02 C9
FF48: 8D F0 D4 C9 AE 90 F4 F0
FF50: F0 C9 BA F0 EB C9 D2 F0
FF58: 3B 86 28 86 29 84 2A B9
FF60: 00 02 49 B0 C9 0A 90 06
FF68: 69 88 C9 FA 90 11 0A 0A
FF70: 0A 0A A2 04 0A 26 28 26
FF78: 29 CA D0 F8 C8 D0 E0 C4
FF80: 2A F0 97 24 2B 50 10 A5
FF88: 28 81 26 E6 26 D0 B5 E6
FF90: 27 4C 44 FF 6C 24 00 30
FF98: 2B A2 02 B5 27 95 25 95
FFA0: 23 CA D0 F7 D0 14 A9 8D
FFA8: 20 EF FF A5 25 20 DC FF
FFB0: A5 24 20 DC FF A9 BA 20
FFB8: EF FF A9 A0 20 EF FF A1
FFC0: 24 20 DC FF 86 2B A5 24
FFC8: C5 28 A5 25 E5 29 B0 C1
FFD0: E6 24 D0 02 E6 25 A5 24
FFD8: 29 07 10 C8 48 4A 4A 4A
FFE0: 4A 20 E5 FF 68 29 0F 09
FFE8: B0 C9 BA 90 02 69 06 2C
FFF0: 12 D0 30 FB 8D 12 D0 60
FFF8: 00 00 00 0F 00 FF 00 00
```

Hardware notes:

Page 0 Variables

XAML	24
XAMH	25
STL	26
STH	27
L	28
H	29
YSAV	2A
MODE	2B

Other Variables PIA

IN	200-27F
KBD	D010
KBD CR	D011
DSP	D012
DSP CR	D013

```
;-------------------------------------------------------------------------
;
;  The WOZ Monitor for the Apple 1
;  Written by Steve Wozniak 1976
;
;-------------------------------------------------------------------------

                .CR     6502
                .OR     $FF00
                .TF     WOZMON.HEX,HEX,8

;-------------------------------------------------------------------------
;  Memory declaration
;-------------------------------------------------------------------------

XAML            .EQ     $24             Last "opened" location Low
XAMH            .EQ     $25             Last "opened" location High
STL             .EQ     $26             Store address Low
STH             .EQ     $27             Store address High
L               .EQ     $28             Hex value parsing Low
H               .EQ     $29             Hex value parsing High
YSAV            .EQ     $2A             Used to see if hex value is given
MODE            .EQ     $2B             $00=XAM, $7F=STOR, $AE=BLOCK XAM

IN              .EQ     $0200,$027F     Input buffer

KBD             .EQ     $D010           PIA.A keyboard input
KBDCR           .EQ     $D011           PIA.A keyboard control register
DSP             .EQ     $D012           PIA.B display output register
DSPCR           .EQ     $D013           PIA.B display control register

; KBD b7..b0 are inputs, b6..b0 is ASCII input, b7 is constant high
;     Programmed to respond to low to high KBD strobe
; DSP b6..b0 are outputs, b7 is input
;     CB2 goes low when data is written, returns high when CB1 goes high
; Interrupts are enabled, though not used. KBD can be jumpered to IRQ,
; whereas DSP can be jumpered to NMI.

;-------------------------------------------------------------------------
;  Constants
;-------------------------------------------------------------------------

BS              .EQ     $DF             Backspace key, arrow left key
CR              .EQ     $8D             Carriage Return
ESC             .EQ     $9B             ESC key
PROMPT          .EQ     "\"             Prompt character

;-------------------------------------------------------------------------
;  Let's get started
;
;  Remark the RESET routine is only to be entered by asserting the RESET
;  line of the system. This ensures that the data direction registers
;  are selected.
;-------------------------------------------------------------------------

RESET           CLD                     Clear decimal arithmetic mode
                CLI
                LDY     #%0111.1111     Mask for DSP data direction reg
                STY     DSP              (DDR mode is assumed after reset)
                LDA     #%1010.0111     KBD and DSP control register mask
                STA     KBDCR           Enable interrupts, set CA1, CB1 for
                STA     DSPCR            positive edge sense/output mode.

; Program falls through to the GETLINE routine to save some program bytes
; Please note that Y still holds $7F, which will cause an automatic Escape
```

```
;-------------------------------------------------------------------------
; The GETLINE process
;-------------------------------------------------------------------------

NOTCR           CMP     #BS             Backspace key?
                BEQ     BACKSPACE       Yes
                CMP     #ESC            ESC?
                BEQ     ESCAPE          Yes
                INY                     Advance text index
                BPL     NEXTCHAR        Auto ESC if line longer than 127

ESCAPE          LDA     #PROMPT         Print prompt character
                JSR     ECHO            Output it.

GETLINE         LDA     #CR             Send CR
                JSR     ECHO

                LDY     #0+1            Start a new input line
BACKSPACE       DEY                     Backup text index
                BMI     GETLINE         Oops, line's empty, reinitialize

NEXTCHAR        LDA     KBDCR           Wait for key press
                BPL     NEXTCHAR        No key yet!
                LDA     KBD             Load character. B7 should be '1'
                STA     IN,Y            Add to text buffer
                JSR     ECHO            Display character
                CMP     #CR
                BNE     NOTCR           It's not CR!

; Line received, now let's parse it

                LDY     #-1             Reset text index
                LDA     #0              Default mode is XAM
                TAX                     X=0

SETSTOR         ASL                     Leaves $7B if setting STOR mode

SETMODE         STA     MODE            Set mode flags

BLSKIP          INY                     Advance text index

NEXTITEM        LDA     IN,Y            Get character
                CMP     #CR
                BEQ     GETLINE         We're done if it's CR!
                CMP     #"."
                BCC     BLSKIP          Ignore everything below "."!
                BEQ     SETMODE         Set BLOCK XAM mode ("." = $AE)
                CMP     #":"
                BEQ     SETSTOR         Set STOR mode! $BA will become $7B
                CMP     #"R"
                BEQ     RUN             Run the program! Forget the rest
                STX     L               Clear input value (X=0)
                STX     H
                STY     YSAV            Save Y for comparison

; Here we're trying to parse a new hex value

NEXTHEX         LDA     IN,Y            Get character for hex test
                EOR     #$B0            Map digits to 0-9
                CMP     #9+1            Is it a decimal digit?
                BCC     DIG             Yes!
                ADC     #$88            Map letter "A"-"F" to $FA-FF
                CMP     #$FA            Hex letter?
                BCC     NOTHEX          No! Character not hex

DIG             ASL
                ASL                     Hex digit to MSD of A
                ASL
                ASL

                LDX     #4              Shift count
HEXSHIFT        ASL                     Hex digit left, MSB to carry
```

```
                ROL     L               Rotate into LSD
                ROL     H               Rotate into MSD's
                DEX                     Done 4 shifts?
                BNE     HEXSHIFT        No, loop
                INY                     Advance text index
                BNE     NEXTHEX         Always taken

NOTHEX          CPY     YSAV            Was at least 1 hex digit given?
                BEQ     ESCAPE          No! Ignore all, start from scratch

                BIT     MODE            Test MODE byte
                BVC     NOTSTOR         B6=0 is STOR, 1 is XAM or BLOCK XAM

; STOR mode, save LSD of new hex byte

                LDA     L               LSD's of hex data
                STA     (STL,X)         Store current 'store index'(X=0)
                INC     STL             Increment store index.
                BNE     NEXTITEM        No carry!
                INC     STH             Add carry to 'store index' high
TONEXTITEM      JMP     NEXTITEM        Get next command item.

;-------------------------------------------------------------------------
;  RUN user's program from last opened location
;-------------------------------------------------------------------------

RUN             JMP     (XAML)          Run user's program

;-------------------------------------------------------------------------
;  We're not in Store mode
;-------------------------------------------------------------------------

NOTSTOR         BMI     XAMNEXT         B7 = 0 for XAM, 1 for BLOCK XAM

; We're in XAM mode now

                LDX     #2              Copy 2 bytes
SETADR          LDA     L-1,X           Copy hex data to
                STA     STL-1,X          'store index'
                STA     XAML-1,X         and to 'XAM index'
                DEX                     Next of 2 bytes
                BNE     SETADR          Loop unless X = 0

; Print address and data from this address, fall through next BNE.

NXTPRNT         BNE     PRDATA          NE means no address to print
                LDA     #CR             Print CR first
                JSR     ECHO
                LDA     XAMH            Output high-order byte of address
                JSR     PRBYTE
                LDA     XAML            Output low-order byte of address
                JSR     PRBYTE
                LDA     #":"            Print colon
                JSR     ECHO

PRDATA          LDA     #" "            Print space
                JSR     ECHO
                LDA     (XAML,X)        Get data from address (X=0)
                JSR     PRBYTE          Output it in hex format
XAMNEXT         STX     MODE            0 -> MODE (XAM mode).
                LDA     XAML            See if there's more to print
                CMP     L
                LDA     XAMH
                SBC     H
                BCS     TONEXTITEM      Not less! No more data to output

                INC     XAML            Increment 'examine index'
                BNE     MOD8CHK         No carry!
                INC     XAMH

MOD8CHK         LDA     XAML            If address MOD 8 = 0 start new line
                AND     #%0000.0111
                BPL     NXTPRNT         Always taken.
```

```
;-------------------------------------------------------------------------
;  Subroutine to print a byte in A in hex form (destructive)
;-------------------------------------------------------------------------

PRBYTE          PHA                     Save A for LSD
                LSR
                LSR
                LSR                     MSD to LSD position
                LSR
                JSR     PRHEX           Output hex digit
                PLA                     Restore A

; Fall through to print hex routine

;-------------------------------------------------------------------------
;  Subroutine to print a hexadecimal digit
;-------------------------------------------------------------------------

PRHEX           AND     #%0000.1111     Mask LSD for hex print
                ORA     #"0"            Add "0"
                CMP     #"9"+1          Is it a decimal digit?
                BCC     ECHO            Yes! output it
                ADC     #6              Add offset for letter A-F

; Fall through to print routine

;-------------------------------------------------------------------------
;  Subroutine to print a character to the terminal
;-------------------------------------------------------------------------

ECHO            BIT     DSP             DA bit (B7) cleared yet?
                BMI     ECHO            No! Wait for display ready
                STA     DSP             Output character. Sets DA
                RTS

;-------------------------------------------------------------------------
;  Vector area
;-------------------------------------------------------------------------

                .DA     $0000           Unused, what a pity
NMI_VEC         .DA     $0F00           NMI vector
RESET_VEC       .DA     RESET           RESET vector
IRQ_VEC         .DA     $0000           IRQ vector

;-------------------------------------------------------------------------

                .LI     OFF
```

Power Supply:

To operate the A-ONE you can use a universal AC-DC Adapter. Before plugging the adapter into the wall socket, please follow the instructions shown below:

VOLTAGE: Set the voltage selector switch to the required voltage (9-12VDC).

CURRENT: Ensure that power supply can provide 350mA of current.

INPUT PLUG: Select the correct size input plug (2.1mm barrel connector).

POLARITY: Make sure that the center of the barrel connector is set to +12VDC.

PROTECTION: The A-ONE is protected against wrong polarity.

Links and downloads:

http://www.sbprojects.com
http://www.applefritter.com/apple1
http://www.achatz.nl
http://www.brielcomputers.com
http://dreher.net/CFforAppleII/

A-ONE hardware design:	Franz Achatz	www.achatz.nl
Video controller Software:	San Bergmans	www.sbprojects.com
Keyboard controller Software:	Ben Zijlstra	www.benshobbycorner.nl

Many thanks to Jack Rubin and Jan Verhoeven for helping me on this Manual

Thanks to Steve Wozniak because he is a nice guy and genius.

Franz Achatz
Baanstraat 134
6372 AJ Landgraaf
The Netherlands

FAX: +31(0) 84 743 4128
Email: info@achatz.nl

http://www.achatz.nl

Achatz Product Warranty

Thank you for purchasing an Achatz product

Two Year Parts and Labour Warranty

Achatz Electronics warrants this product against any defects in materials and workmanship for a period of two years from the date of invoice. In the event of a malfunction during the warranty period, Achatz Electronics will repair or replace this product to its original operation conditions. To assure the highest level of service, a return authorization number must be obtained from Achatz Electronics before products are returned for service.

This unit has been thoroughly tested and inspected to assure proper performance and operation

Achatz Electronics

www.achatz.nl

Baanstraat 134
6372 AJ Landgraaf
The Netherlands
Fax: +31 (0) 84 743 4128

Project News:

Coming up, soon