A-One Terminal

Wouldn't you want to own your very own Apple 1? I know I would! Unfortunately the original Apple 1 is hard to come by and above all very expensive.
The next best thing is to buy yourself one of its replicas. Franz Achatz built one which he had for sale for a while. Unfortunately he has stopped selling them.
To be honest Vince Briel (http://www.brielcomputers.com) was the first one to create an Apple 1 replica. But Franz's machine is a little smaller, contains less chips and has a selectable 50Hz/60Hz video output which is rather important here in Europe.

Original parts for the Apple 1 terminal are very hard to get these days. On top of that they require a much bigger board space and a lot of extra power. An alternative had to be found, which comes in the form of two Atmel AVR processors.
An ATTiny2313 acts as a keyboard controller, while an ATMega32 is responsible for the video output.
The keyboard controller translates PS/2 keyboard signals to ASCII, which can be read by the Apple 1 as if it were an original ASCII keyboard.
The video controller accepts the characters from the Apple 1 in exactly the same way as the Apple 1 terminal would. That way absolutely no compromises have been made to the Apple 1's behaviour.
As a bonus these two micro controllers add a serial input and output to the system, which makes it a lot easier to enter and save programs using a modern PC as storage device.

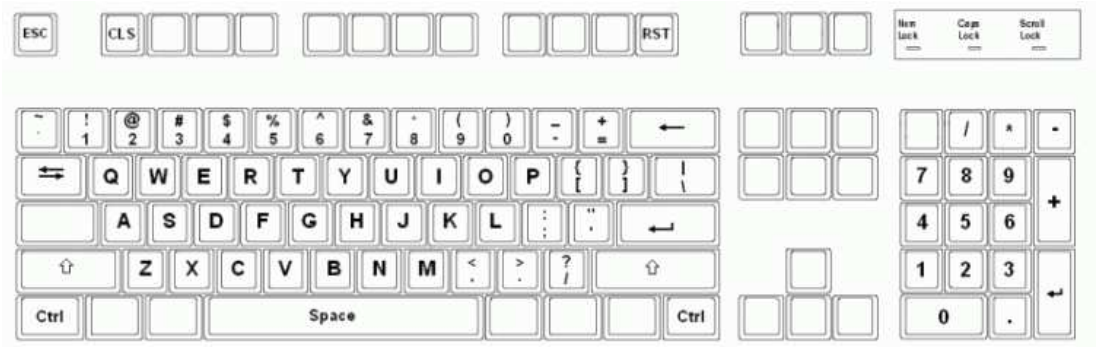
The A-One terminal enables you to run the A-One using a normal 50Hz or 60Hz TV as video screen and a US layout PS/2 keyboard as input device.
Alternatively you can use a terminal program like Hyperterm, connected to the A-One's serial interface, to run the A-One entirely from a modern PC, with or without a TV attached to the video output.

You can download the source and HEX files for both controllers from the downloads (download.php) page. The keyboard controller software is written in Bascom (https://www.mcselec.com) by Ben Zijlstra (http://www.benshobbycorner.tk). The video controller software is written using my SB-Assembler (../../sbasm/index.php) by me, myself and I.

Keyboard Controller

First I'll describe the keyboard controller. Its task is to translate PS/2 keyboard signals to a ASCII codes which can be presented to the Apple 1. As a bonus you can also enter the characters through the processor's serial input, at 2400 baud.

The program for the keyboard controller was written in Bascom (https://www.mcselec.com/). Since I'm a complete newby to Bascom I found Ben Zijlstra (http://www.benshobbycorner.tk), our Bascom expert, willing to help making this program to perform the way it does today.



The keyboard controller expects a US layout PS/2 keyboard. Not all keys are used because they wouldn't make much sense to the Apple 1. As a bonus you can type some keys which could not be typed on an original Apple 1 keyboard (e.g. the square and curly brackets).
In order to avoid confusion the keyboard controller can only generate upper case characters.

Because of the used Bascom keyboard library the operation of the Ctrl key may be a bit peculiar. If you need a Ctrl character you'll have to press Ctrl once (no need to hold it) and then type the appropriate character. In case you press the Ctrl key by accident simply press it once more to cancel it.

Two of the function keys have a special function. F1 will clear the screen, just like pressing the CLS button on the A-One board. F12 will reset the A-One, just like pressing the Reset button on the A-One.

The original Apple 1 keyboard input routine GETLINE will mess up the input of special characters a little. Therefore if you want all the keys, including the Ctrl characters, to be interpreted the way they are supposed to be you'll have to write your own input routine. However don't expect such a program to run the same on an original Apple 1.

Video Controller

To the A-One's point of view the video controller behaves exactly like the original Apple 1's hardware terminal output. A character is sent to the video controller, where it even has to wait for its turn to be displayed. When it is finally accepted it is put in video RAM and at the same time the character is also sent out to the serial output of the ATMega32 at 2400 baud (8 data bits, no parity, 1 stop bit).

As opposed to an original Apple 1, which displays garbage after it has been switched on, the A-One will show a splash screen with some interesting URL's you can visit. You are then prompted to press the RESET switch, which will start the Apple 1, clears the screen as usual and shows the flashing cursor.
The rest of the operation is quite the same as with an original Apple 1. Although..... We couldn't resist the temptation to make the software a bit cleverer than absolutely necessary. A CTRL-L sent to the video controller will clear the screen and put the cursor in the upper left hand corner. At the same time most terminal programs connected to the serial output will also clear their screen when a CTRL-L is received.

I would like to add one little word about the artificial character delay. As you can read in the description about the Apple 1 terminal each printed character has to wait until the character under the cursor is displayed before it can be accepted. Our software version can accept characters much faster than that if it wants to. Normally it doesn't want to and it will accept the characters at an approximate 16ms interval. This interval is not related to the cursor position like on the real Apple 1, it is simply an interval of 256 video lines. Only a very carefully written Apple 1 program could tell the difference!
The beauty of having a serial I/O on the A-One is that you can send text files to the A-One which are simply treated as key strokes. But at 2400 baud the characters are coming in too fast if we hang on to this artificial 16ms delay.
Therefore I have arranged for the keyboard controller software to assert a dedicated line between the two controllers which can temporarily disable the artificial delay. This ensures that the file can be sent at top speed (2400 baud) to the A-One, without missing any characters.

All's well that ends well. Or is it? Not exactly. The A-One can accept all characters at top speed indeed. However the Apple 1 has the tendency to add some more characters to its output. For instance if you enter a line of hex bytes, pressing CR will generate an extra CR, the address is repeated, followed by a colon and the first data byte.
This means that during the upload of a text file more characters have to come out than there are coming in. Without flow control this will inevitably lead to loss of output characters. However the A-One will not miss any characters, so the program is transferred correctly. And the output to the screen is also correct. Only the serial output will become a little messy during a file transfer and should simply be ignored.

Video Controller Software

We are in a hurry, at least we are if we want to transform an ATMega32 into a single chip video controller. The software is very time critical and sometimes it has to run at top speed to get the pixels out in time. Failing to do so will inevitably be visible as horizontal jitter, producing an unpleasant and unstable picture. Vince Briel's Replica 1 (SE) uses a hardware shift register to shift out the pixels, which makes the software less time critical. However we'll have to do without a hardware shift register, which required me to use all my programming skills to create a stable video signal.

You can find a copy of the video controller software, including the source file, on the Downloads page (download.php). The program was written using my own SB-Assembler (../../sbasm/index.php).
I will explain some of the software features in more detail here for educational purposes.

The entire program runs in one single interrupt routine. The main program is empty, well almost empty. Interrupts are spaced exactly 64µs apart for 50Hz systems, or 63.5µs for 60Hz systems. Each interrupt routine starts by generating an H-Sync pulse. A state machine then determines what is to be done during the rest of the line. Care should be taken that the interrupt routine ends before it's time to start a new one.

What needs to be done?

- Generate a V-Sync pulse every field.
- Generate an H-Sync pulse every line.
- Generate pixels according to character bit map.
- Display a flashing cursor.
- Accept characters from the Apple 1 and put them on the screen.
- Introduce an artificial character input delay every 16ms or so.
- Echo incoming characters to the serial output.
- Clear the screen if necessary.
- Scroll the screen if necessary.

All these tasks are performed inside the interrupt routine. A simple state machine determines what task needs to be done during each particular video line. Some lines leave no time to do anything else but control the video/sync outputs. Other lines allow plenty of time (>55µs) to do other tasks. These other tasks are: accepting characters from the Apple 1, sending the characters to the serial output, scrolling the screen, flashing the cursor, etc, etc.

Differences between 50Hz and 60Hz systems:

Before we continue with the explanation of the video controller software I should start by summing up the differences between the two main video systems. We're talking black and white only here, so that simplifies the differences. You can select which video system you prefer on the A-One by placing or removing a Jumper.

The main differences are the number of lines per field and the length of each video line. Officially a 50Hz system uses 312½ lines per field, while a 60Hz system uses 262½ lines. These values create interlaced pictures of 625 and 525 lines respectively. For the A-One we are only interested in non-interlaced pictures, so we simply forget about the half lines.
A single video line is 64µs long for 50Hz systems, whereas it is only 63.5µs long for 6oHz systems.

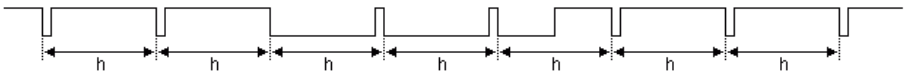
These are the main differences between the two video systems. It goes without saying that there are some other differences regarding the number of blank lines above, between and below the text lines. These difference are simply caused by the fact that we have 50 lines per field more to spend on 50Hz systems.

Generating a video signal is a very time critical business, every nano second counts. Fortunately the ATMega32 can execute most of its instructions within 62.5ns when running at 16MHz. We're definitely going to need that speed.

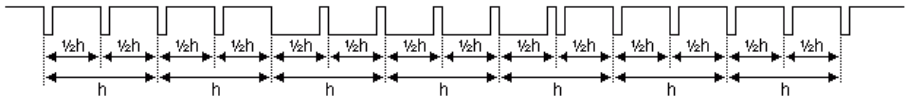
First of all we need a very accurate and stable 64µs (63.5µs) time base. This is achieved by using Timer 0 to count from 0 to 127 (126) with a pace of 0.5µs per step.
Every time Timer 0 reaches TOP it is automatically resets to 0 and an interrupt is generated. The first thing we do during this interrupt is generate a falling edge

on the Sync output of the ATMega32. From then on it is a matter of transferring control to the appropriate routine, based on a state machine.
After performing all appropriate tasks for the current video line control is returned to the main program by executing a final RETI instruction.

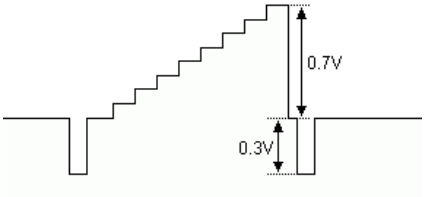
By the way, the main program is a bit boring perhaps. All we do there is put the machine to sleep, where it will stay until the next Timer 0 interrupt occurs. This is done to assure a constant interrupt latency to avoid horizontal jitter.



The picture above shows what a basic Vsync signal should look like. Please notice that the falling edges of the sync signal are always spaced exactly **h** apart! This ensures that the Hsync separator in the TV can accurately lock onto the incoming video signal.
Since we are generating a non interlaced picture you won't find a half line during this Vsync period. The presence of a half line would require us to generate equalizing pulses during the Vsync period to ensure that there will be a falling edge every **h** at all times. Technically speaking we don't really have to generate these equalizing pulses for a non-interlaced picture. However to ensure compatibility with even the most demanding TV set I decided to include them anyway. The picture below will show you what the A-One's Vsync pulse will finally look like.



Believe it or not, but the Vsync signal was the most complicated signal of the entire project. The rest of the picture consists of empty video lines and video lines containing the generated text.
An empty video line is merely a 4.7µs lasting sync puls with the rest of the line remaining at black level. This gives us more than 55µs time to handle the terminal jobs during these empty lines. These jobs will be explained later.
The text lines are very time critical, and must be executed at top speed. In order to achieve top speed I had to carefully align the video buffer and the character generator lookup tables in memory. This is done to minimize the required computation time of the index pointer for the lookup tables. The character generator lookup table is copied to RAM because the processor can read data faster from RAM than from ROM.



Here you see a typical video line with a gray staircase signal. We won't be generating any shades of gray, only black or white. Black being approximately 0.3V above the sync tip, and white approximately 1V above the sync tip.
This means that we only need 3 analog levels:

PortD.2	PortB.0	Video level
low	low	Sync tip
high	low	Black level
high	high	White level

```
;---------------------------------------------------------------------
;  Constants
;---------------------------------------------------------------------

LINES_TOP50     .EQ     40              # of empty lines above pic (50Hz)
LINES_TOP60     .EQ     26              # of empty lines above pic (60Hz)
LINES_INT50     .EQ     3               # of lines between text (50Hz)
LINES_INT60     .EQ     2               # of lines between text (60Hz)

CTRLL_OPT       .EQ     1       or 0    Ctrl-L = Clear screen option

CURSOR          .EQ     '@'             Cursor character
CURSOR_RLD      .EQ     16              Cursor reload timer value
```

From the source file I will only describe some "tweakable" constants here. The source code is littered with comments, which should explain the entire operation of the software. You may change these values to suit your personal preference if you like. However in normal situations there is no need to change any of these values.

The LINES_TOPxx values are the number of empty video lines between the end of the VSync and the first text line. I've tuned these values to center the picture on my TV set. In case the picture on your TV is totally out of center you may change these values.

The LINES_INTxx values are the number of blank lines in between two rows of text. The value should not be set lower than 1. I've chosen these values to create an acceptable distance between the text lines on both video systems.

Whatever you set the line values to, the assembler automatically calculates how many empty lines remain between the last text line and the beginning of the next VSync. You will be warned if there are no lines left, which requires you to lower the values for LINES_TOPxx and/or LINES_INTxx before you can create working software again.

CTRLL_OPT is a flag which allows you to disable the CTRL-L = Clear Screen interpretation of the software. We all have the tendency to overload the system with "nice to have features" even though the original Apple 1 didn't have them.
If you don't want the A-One to interpret a printed CTRL-L as Clear Screen, simply set CTRLL_OPT to 0 and reassemble the file.

The last two "tweakable" values are CURSOR and CURSOR_RLD. CURSOR is simply the cursor character. It is set to be an @ symbol because the original Apple 1 used it too.

You can also change the flash rate of the cursor by changing CURSOR_RLD. The value counts the number of fields before the state of the cursor is changed from space to @ or vice versa. This means that the cursor will flash slightly faster on a 60Hz system, but who cares. If you do care you can change this value to whatever you like.