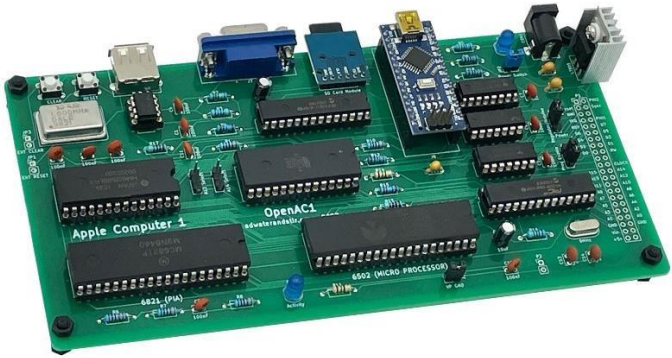
Open AC-1 Assembly and Operation



This project is a combination of multiple open-source and public domain efforts. I would like to acknowledge and thank the authors and creators.

Integer BASIC - Steve Wozniak – Steve hand assembled the BASIC interpreter and Wozmon included in ROM.

RC6502 Apple 1 SBC – Copyright Tor-Eirik Bakke Lunde GPL-3.0 license – Much of this project is an extension of his work.

VT100 Terminal – Copyright Geoff Graham GPL-3.0 license – His work provides the VGA output and character generation, which I modified to use the Apple 1 font.

USB Keyboard processing for the VT100 Terminal program – David Hansel GPL-3.0 license

Table of Contents

Why Did I Create This Kit? **3**

Parts List **4**

Building Your OpenAC1 ..... **5**

Jumper Settings **10**

USB Keyboard and VGA Monitor **11**

Initial Power Up **12**

Wozmon **13**

Using Apple BASIC **15**

Using the SD Card **16**

The Rubout Character **17**

Saving Programs **18**

Appendix A **19**

WARNING: You are creating a kit using technology from the mid70s. While the components are modern, they are still as susceptible to static discharge as they were almost 50 years ago. Please use precautions to protect against electrostatic discharge (ESD).

You can protect against ESD and discharge static electricity from your body by touching a metal grounded object (such as an unpainted metal surface on your computer's I/O panel) before you interact with anything electronic. When unpacking static-sensitive components, do not remove the component from the antistatic packing material until you are ready to install the component. Just before unwrapping the antistatic package, be sure to discharge static electricity from your body.

Why did I create this kit?

The world certainly did not need another "work-alike" Apple 1. In fact, it's possible nowadays to fully source and build your own exact replica of an Apple 1 (here's mine):

While building an exact replica is fun and rewarding, it is a lot of work, expensive (around $1,000), and requires a lot of troubleshooting and maintenance to keep it running.

I also read the book "Apple I Replica Creation: Back to the Garage" by Tom Owad and enjoyed the in-depth discussion of Steve Wozniak's design and inspiration for the Apple 1, as well as Vince Briel's work in creating the Replica 1.

My goal was to create a through-hole kit that can be assembled in a weekend, allow you to get a taste of what using an Apple 1 was like, load real Apple 1 software quickly and easily, and use modern easily-obtained peripherals (USB keyboard, VGA monitor, SD card). My real interest is building affordable kits, since I grew up in the generation of Heathkit, Altair 8800, COSMAC Elf, and the ZX-80/81 kits.

If you want to build a "more authentic" Apple 1 replica, please see the resources listed in Appendix A.

Parts List

I would strongly suggest comparing the parts you received with the list below. Let me know if you are missing anything and I will send a replacement.

Bag #1

- 5 x 3.3kΩ Resistor
- 1 x 1MΩ Resistor
- 1 x 47kΩ Resistor
- 1 x 1kΩ Resistor
- 1 x 330Ω Resistor
- 2 x 10kΩ Resistor
- 2 x 4.7kΩ Resistor
- 2 x 150Ω Resistor
- 1 x 220Ω Resistor
- 1 x 470Ω Resistor

Bag #2

- 2 x 40-Pin Wide DIP Socket
- 2 x 28-Pin Wide DIP Socket
- 2 x 28-Pin Narrow DIP Socket
- 2 x 14-Pin DIP Socket
- 1 x 16-Pin DIP Socket
- 1 x 8-Pin DIP Socket
- 2 x 15-Pin Single Female Header
- 2.54” Male Headers

Bag #3

- 2 x 10µF Capacitor
- 11 x 100nF Capacitor
- 2 x 27pF Capacitor
- 1 x 10nF Capacitor
- 1 x 8MHz Crystal
- 1 x 1MHz Oscillator

Bag #4

- 1 x 65C02 Processor
- 1 x 68C21 PIA
- 1 x HM62256 RAM
- 1 x 28C64 ROM
- 1 x MCP23S17
- 1 x PIC32MX250
- 1 x 74HCT138
- 1 x 74HCT00
- 1 x 74HCT04
- 1 x NE555

Bag #5

- 2 x 5mm LED
- 2 x Tactile Button
- 1 x DB15 Connector
- 1 x USB-A Connector
- 1 x DC Barrel Jack
- 1 x Micro SD Card Module
- L7805 Regulator
- TO-220 Heatsink
- 1 x 100nF Capacitor
- 1 x 330nF Capacitor
- 4 x M3 8mm Standoffs
- 4 x M3 Nylon Nuts
- 6 x Jumpers
- 1 x MicroSD Card
- Male Headers

Unbagged Parts

- 1 x PC Board
- 1 x 9v Power Adapter
- 1 x Arduino Nano

Building your Open AC1:

Building the Open AC-1 is a pretty straight-forward process. In fact, if you're an experienced kit builder you can probably do it without consulting the instructions very much, as all component values are clearly indicated on the circuit board. But if you prefer step-by-step instructions, here we go!

In general, I like to build kits from the lowest profile component to the highest. This makes it easier to flip the board over and solder the components you are working on.

Take bag #1 with the resistors. You can identify the resistors with the following color codes, but I find it faster to use a multimeter.

3.3kΩ x 5		Orange-Orange-Black-Brown
1MΩ x 1		Brown-Black-Black-Yellow
47kΩ x 1		Yellow-Violet-Black-Orange
1kΩ x 1		Brown-Black-Black-Brown
330Ω x 1		Orange-Orange-Black-Black
10kΩ x 3		Brown-Black-Black-Red
4.7kΩ x 2		Yellow-Violet-Black-Brown
150Ω x 2		Brown-Green-Brown
220Ω x 1		Red-Red-Black-Black
470Ω x 1		Yellow-Violet-Black-Black

Place the resistors in the correct locations as labeled on the circuit board. Resistors are not polarized so the orientation is not important. Solder them in place.

Bag #2 contains the DIP sockets. Solder the sockets with the orientation indicator facing to the right as indicated on the circuit board. Solder these carefully so there are no solder bridges.

Add the two fifteen pin female headers to position J6. It will help to use the Arduino to hold the headers vertical while soldering these connectors.

Cut the male pin headers into the following segments: four 3-pin segments and two 2-pin segments. These will be soldered in locations A1-A4, JP2 and J5.

Next is bag #3. Solder the 8MHz crystal in position Y1. The crystal is not polarized, so orientation is not important.

Look closely at the capacitors. There are eleven 100nF capacitors (marked with "104") and two 27pF capacitors (marked with "27"). They look similar, so be sure you place them correctly.

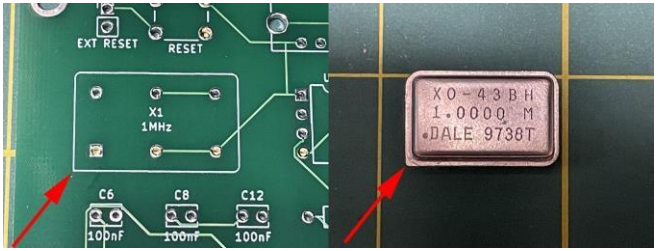
Solder the two 27pF ceramic disc capacitors in locations C13 and C14. The orientation is not important.

There are eleven 100nF capacitors (marked with "104") that are soldered into locations C1, C4-C12, and C18. Orientation is not important.

Solder the 10nF capacitor (marked with "103") in position C2. Orientation is not important.

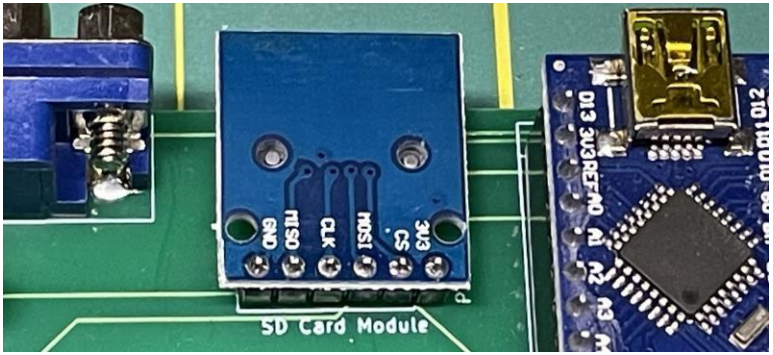
Add the two 10µF electrolytic capacitors to locations C3 and C17. These capacitors are polarized, so place the longer leg in the "+" hole on the left side.

Solder the 1MHz oscillator in location X1. The orientation of this is important. You'll notice the graphic on the circuit board has three rounded corners with one right-angle corner on the lower left. Be sure this matches the corners on the oscillator.



There are a few miscellaneous components to add from bag #5. Solder the LEDs in position D1 and D2. Make sure the long lead is inserted in the right hole, with the flat edge facing left.

Next you will add the SD card module to P3 as shown.



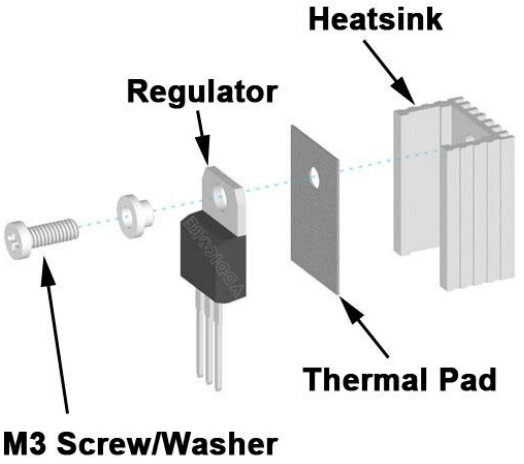
Add the tactile pushbuttons in position SW1 and SW2.

Solder the VGA connector in position and the USB-A connector in position P4.

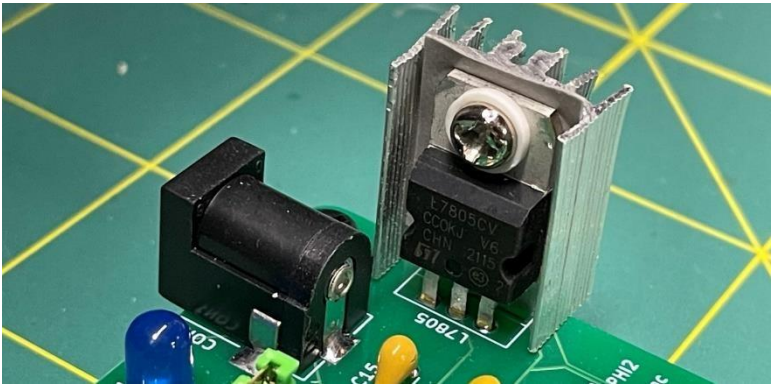
The DC power jack is soldered in position CON1.

Solder the 100nF capacitor (marked with “104”) in position C15 and the 330nF capacitor (marked with “334”) in position C16. The orientation of the capacitors is not important.

Assemble the L7805 regulator and heatsink assembly

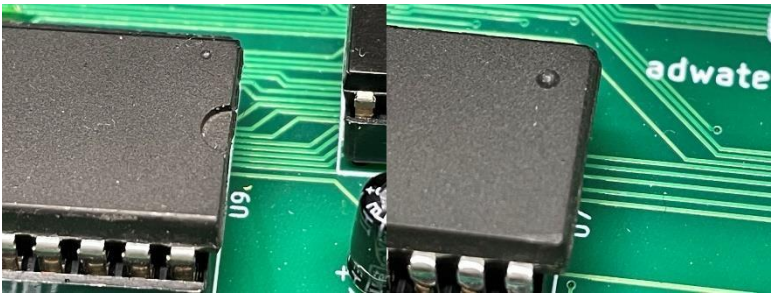


and solder it as shown in position U10.

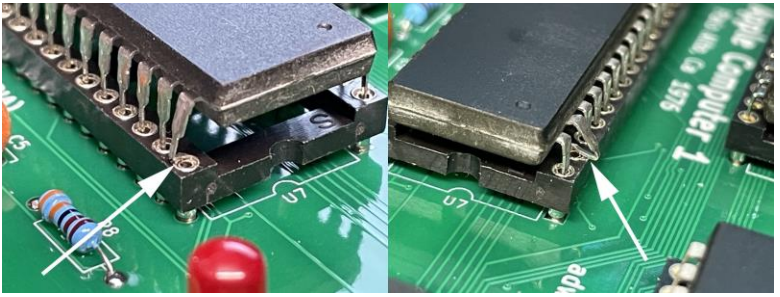


Now you can plug the pre-programmed Arduino Nano into the female headers in position J6.

From bag #4, place the ICs in the correct locations as indicated by the labels on the circuit board. Be sure to pay attention to the polarity indicators. The polarity indicator is typically a notch in the side of the IC, but sometimes can be a dot.



Be very careful that the pins go in straight and you don't have a pin bent under the chip. I have had that happen a couple times.



Finally, place the four nylon standoffs in the four corner holes and secure with four nylon nuts.

(You'll notice that a 44-pin double male header is conspicuously absent from the parts included – that is because I have nothing yet designed to utilize that connector. Maybe someday… For now, it is available for your experimentation.)

Jumper settings

Location A3 controls the connection between the Arduino and the PIC32 processor. When in the lower position, the OpenAC1 operates normally. When in the upper position it allows new programming to be uploaded to the Arduino through its onboard USB connection. When the jumper is in the upper position, this also allows you to use a computer and terminal emulator for input/output via the USB connection.

Location A4 determines the power source for the 6502 and other chips on the circuit board. Typically, this will be placed on the lower position which enables power from the L7805 regulator. In the upper position, the chips will be powered from the Arduino's onboard voltage regulator. **If you want to power the device from a USB power source connected to the Arduino, place this jumper in the upper position. If powered from an external power adapter, place this jumper in the lower position.**

JP1 is a connector for an external reset switch and JP3 is a connector for an external clear switch.

JP2 is a connector if you would like to add an external power switch, for example if you create a case for this project. Otherwise you can place a jumper across these two connectors, then the OpenAC1 will be powered whenever 9v power is supplied.

J5 connects CPU pin 1 (VPB) to GND. This is required for most CPUs (Rockwell, Synertek, UMC, MOS, etc.) but is left off for WD65C02 (which I don't recommend using).

Jumpers A13 and A14 control which 8k page of ROM is mapped to $E000 - $FFFF (typically the location of BASIC and Wozmon). When a 28C64 ROM is installed, there is only one 8k page available, so **no jumpers are required on A13 and A14.** If you have a 28C256 ROM installed, you have four 8k pages to choose from with the following table:

This is only useful if you have multiple ROM images for the Apple 1. Of course there is the version that is supplied with this kit (with Wozmon, Apple BASIC, and Krusader Assembler), but there is at least one more available on the internet with a version of Applesoft BASIC for the Apple 1.

USB keyboard and VGA monitor

Your OpenAC1 is almost ready. You'll need to connect a VGA monitor and a USB keyboard. Unfortunately, all USB keyboards are not created equal. Why? The PIC processor that controls the video and keyboard is, of course, limited in its memory and processor speed. USB protocols can be complicated and memory intensive. For example, the USB keyboard protocol is different than the USB hub protocol, and there is not enough memory in the PIC to support both protocols (and the video emulation).

So it was decided that the video emulator would support only the standard USB keyboard protocol.

Well, most "fancy" USB keyboards with multiple features, or wireless USB keyboards, actually have internal USB hubs which require the host computer to have separate USB protocols implemented.

So what USB keyboards will work with the OpenAC1?

The easy answer is almost any standard, directly wired USB keyboard, with no additional features (the cheap no-frills keyboards). Of course, a lot of people would prefer to pair a retro Apple 1 kit with a retro-looking mechanical keyboard. So I am assembling a list of verified working keyboards you may be interested in. Please see the resources listed in Appendix A.

Initial Power Up

Now that everything is plugged in a ready to go, finally insert your Micro SD card in the slot (label facing down and gold contacts facing up), connect the 9v power supply and voila!

Is everything working? If not, check to see if you have a jumper on JP2. If the screen does not clear on its own after about three seconds, press the small reset button **on the top** of the Arduino (not the reset button on the upper left of the main circuit board).

The original Apple 1 did not have a reset circuit and required you to press reset before using it. This is optional in your kit, but go ahead and press it if you like. The cursor on an Apple 1 was a flashing "@" symbol.

You are in the Wozmon (or what Steve Wozniak interchangeably called the "System Monitor" and "Hex Monitor"). Go ahead and try a few commands. Wozmon is very easy and has very few commands. It's mainly used for reading/entering values and launching applications.

Wozmon

The following examples are from Steve Wozniak's original user manual for the Apple 1. In these examples, (RET) means to press the Return or Enter key. Returned values are Woz's original examples, your memory contents may be different.

• Examining the contents of a single address:

```
USER TYPES/             4F (RET)
MONITOR TYPES/          004F: 0F (contents of 4F)
```

• Examining a block; from the last examined location, to a specific one.

```
USER TYPES/             .5A (RET)
MONITOR TYPES/          0050: 00 01 02 03 04 05 06 07
                        0058: 08 09 0A
```

Note: 4F is still considered the most recently opened location.

• Combining examples 1 and 2 to print a block of memory in a single command.

```
USER TYPES/             4F.5A (RET)
MONITOR TYPES/          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".

• Examining several individual locations at once.

```
USER TYPES/             4F 52 56 (RET)
MONITOR TYPES/          004F: 0F
                        0052: 02
                        0056: 06
```

• Examining several blocks of memory at once.

```
USER TYPES/             4F.52 56 58.5A (RET)
MONITOR TYPES/          004F: 0F
                        0050: 00 01 02
                        0056: 06
```

```
0058: 08 09 0A
```

• Depositing data in a single location.

```
USER TYPES/              30: A0 (RET)
MONITOR TYPES/           0030: FF (prior contents)
```

Note: Location 30 is considered opened and not contains A0.

• Depositing data in successive locations from the last location used in a deposit command.

```
USER TYPES/              : A1 A2 A3 A4 A5 (RET)
```

(This deposits A1 in location 31, A2 in location 32, and so on.)
Note: A colon in a command means “start depositing data from the most recently deposited or opened location.”

• Run a program at a specified address.

```
USER TYPES/              10F0 R (RET)
MONITOR TYPES/           10F0: A9 (contents)
```

• Run at the most recently examined location.

```
USER TYPES/              10F0 (RET)
MONITOR TYPES/  10F0: A9 USER TYPES/
R (RET)
```

Note: RAM locations 0024 to 002B are used as index pointers by the monitor, and are invalid for user use when using the monitor. Also, locations 0200 to 027F are used as input buffer storage, and are also invalid for use when using the monitor.

Using Apple BASIC

To enter BASIC, type E000R (RET). The computer will respond by printing the prompt character >. This prompt character (>) is used throughout BASIC to signify that it is ready for additional commands or statements.

To exit BASIC hit the "RESET" button. This will return control to the monitor. To re-enter BASIC from the monitor without losing the previous program, type E2B3R (RET) instead of E000R. This is extremely useful when you have unintentionally hit the reset button while in BASIC. Entering BASIC at E000 clears and previous programs. You can also exit BASIC to the monitor by issuing the command CALL -151.

Apple BASIC (which is what this version of BASIC was called at the time) is very similar to Integer BASIC for the Apple II, and very similar to other versions of BASIC, but there are a few differences worth mentioning.

HIMEM = (expr) Sets the high memory boundary for user programs (in decimal). Initializes to 4096.

LOMEM = (expr)	Sets the low memory boundary for user Programs (in decimal). Initializes to 2048. Both HIMEM and LOMEM destroy any current user programs.
SCR	Scratches (deletes) the entire BASIC program. Nothing is saved.
END	All programs must have an END statement.

Using the SD card

Of course the original Apple 1 did not have an SD card, but I have added one to my kit. The SD card has text files on it that can be read and written by a modern computer. Be sure to insert the SD card before powering on your OpenAC1, as the card is not "hot swappable". If the SD card is not inserted when powering up, the screen will display the message "SD Card not detected" if you try to access it.

From the command line type CATALOG (RET). This works at both the BASIC or monitor prompt. A list of files on the SD card is displayed. The listing will pause and wait for a key press after each set of 18 files.

Example file listing:

```
I  002  SLOTS.BAS
B  010  STARTREK.BIN
B  010  WUMPUS.BIN
I  003  CHECKERS.BAS
B  011  APPLE30.BIN
```

Etc.

CATALOG and LOAD are functions used to access the SD card and are not Apple 1 functions. The keywords are scanned by the Arduino and intercepted if the keywords CATALOG and LOAD are detected.

BASIC file should have a .BAS extension and binary data files should have a .BIN extension. The file listing also displays the size of the file (in kilobytes) and a "B" for binary data or "I" for Integer BASIC (similar to the Apple II catalog command.) This is determined by the extension of the file, not the contents.

You can load a file by entering LOAD SLOTS.BAS (RET) for example. The file will be read line-by-line from the SD card and transferred to the OpenAC1 memory just as if you are typing in the program

from the keyboard. It may seem slow, but it is about as fast as it was loading a program from cassette.

When the program is done loading, type RUN (RET) (if you are in BASIC.) For binary data loaded while in the system monitor, I suggest adding the start address followed by R as the last line of the text file on the SD card, so the program starts automatically when loaded (ex: 0300R).

When you load a .BAS file, the Arduino issues E000R before loading the contents of the file. This will ensure your system has loaded Apple BASIC before loading the file. If you are in the system monitor, that command will start BASIC, and if you're already in BASIC it will simply cause a harmless *** SYNTAX error.

Likewise, CALL -151 is issued by the Arduino when loading .BIN files. This will enter the system monitor if the system is in BASIC when loading the file.

The Rubout Character

You may have noticed by now that the backspace key does not work as expected. This is by design. When Steve Wozniak designed the Apple 1, he was a pioneer in using a video monitor. Steve has said "I had a TV set and a typewriter and that made me think a computer should be laid out like a typewriter with a video screen." But Wozniak did not implement a true backspace function. Before the "TV/Typewriter" paradigm, output was generally produced on a teletype (at that time, "Glass Terminals" did exist, but were extremely expensive and only suitable for business, education, and scientific use – certainly not for hobbyists). In those days, a "rubout" character was used instead of a backspace. The teletype would print an underscore ("_") which would indicate the previous character was ignored.

Using the rubout character, the following typos could be corrected and the computer would interpret this as a valid entry:

```
10 PRIM_NT "HELO_LO WORLD"
```

That's why your Apple 1 displays an underscore when you press the "Backspace" key.

Saving Programs

Sorry to say, there is currently no method to save programs you have entered. Since this is an open source project, I hope someone will try writing a save method and add it to the Arduino code.

Until then, you could connect the OpenAC1 to a terminal emulator on a laptop/desktop with a USB cable. When you plug the Arduino into a computer, it can be used as a serial port. Using this method you can save a listing by using the terminal emulator's text capture functionality.

Power Adapters

A 9v US power adapter is included in your kit. If you need to replace this adapter, or if you need an adapter outside of US/Canada/Mexico/Japan, you will need a 9v adapter with a 5.5mm x 2.1mm center-positive connector (a standard Arduino power adapter.)

Appendix A – Other Resources

Here are some resources if you would like to look into building or buying an exact Apple 1 replica.

The Apple-1 Enthusiasts group on Facebook is very active and is a good source for information:
https://www.facebook.com/groups/2340248572741025

Also take a look at the current offerings on eBay, just search for "Apple 1" in Vintage Computers.

There is lots of great info and an active forum available on Tom Owad's Applefritter website: https://www.applefritter.com/

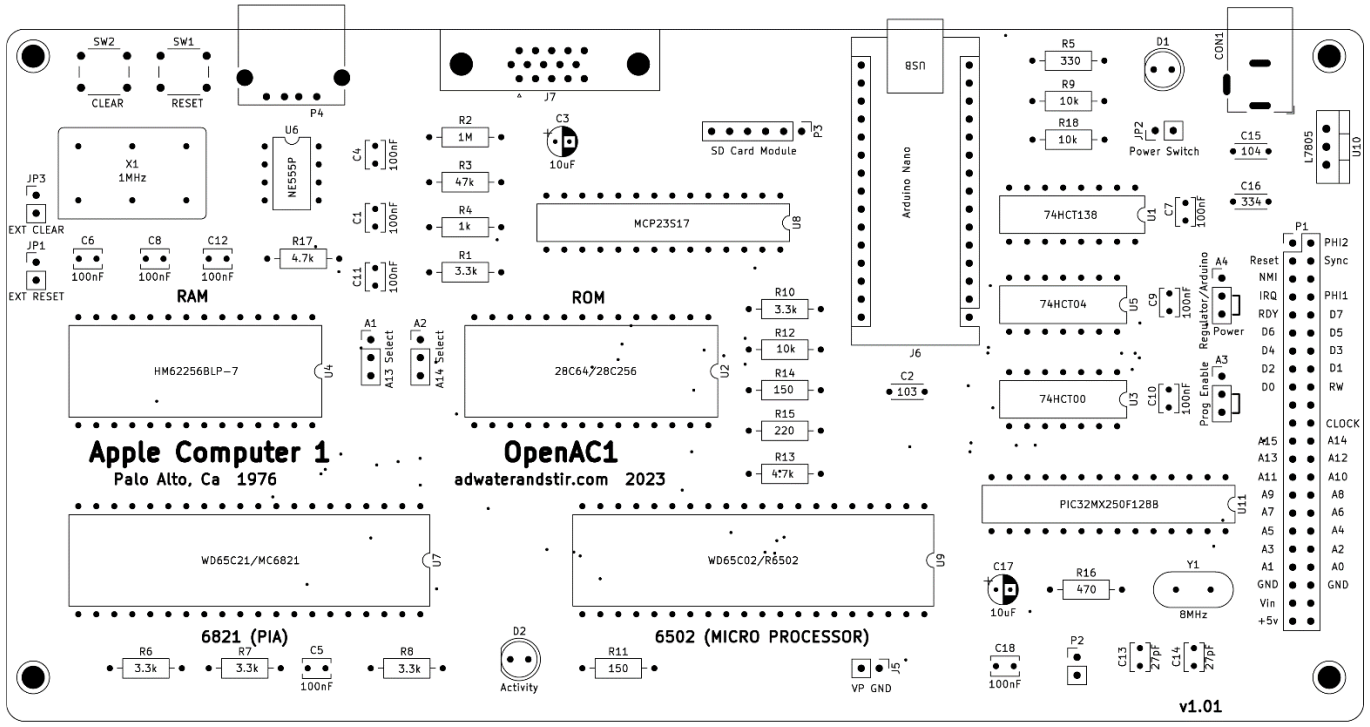
My Google Groups forum is the place to go to ask questions specifically about this kit: https://groups.google.com/g/openac1

Keyboards tested to work with OpenAC1:
https://adwaterandstir.com/keyboards/

If you'd like to download the Arduino source code and add your own enhancements, here is my GitHub page:
https://github.com/davischris

The original Apple 1 manual can be downloaded here:
https://adwaterandstir.com/files/apple1manual.pdf

The original Apple BASIC manual can be downloaded here:
https://adwaterandstir.com/files/basicmanual.pdf


Apple Computer 1
Palo Alto, Ca 1976
OpenAC1
adwaterandstir.com 2023
v1.01
RAM
ROM
6821 (PIA)
6502 (MICRO PROCESSOR)
HM62256BLP-7
28C64/28C256
WD65C21/MC6821
WD65C02/R6502
PIC32MX250F128B
74HCT138
74HCT04
74HCT00
MCP23S17
NE555P
Arduino Nano
SD Card Module
Power Switch
L7805
X1
1MHz
8MHz
CLEAR
RESET
EXT CLEAR
EXT RESET
A13 Select
A14 Select
Activity
VP GND
Regulator/Arduino Power
Prog Enable
Reset
NMI
IRQ
RDY
PHI1
PHI2
Sync
RW
CLOCK
D7
D6
D5
D4
D3
D2
D1
D0
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
GND
Vin
+5v