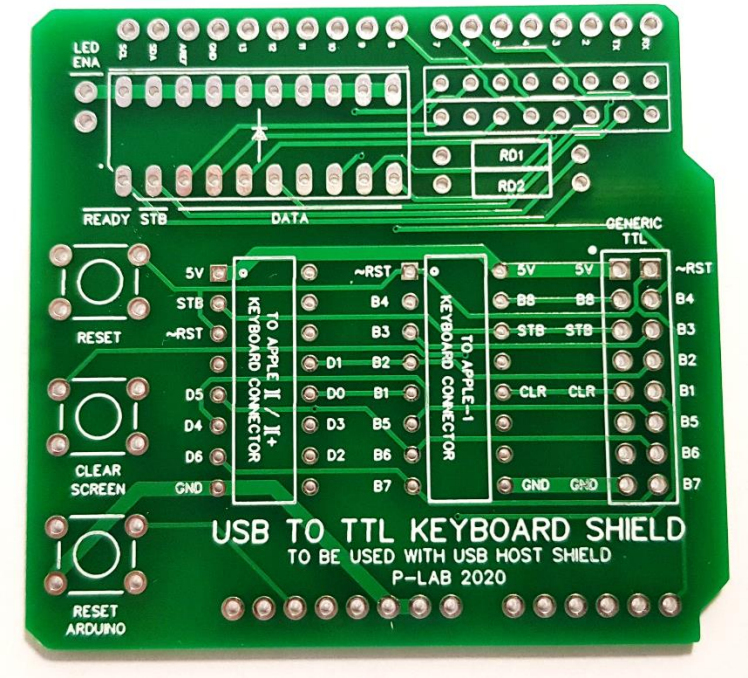
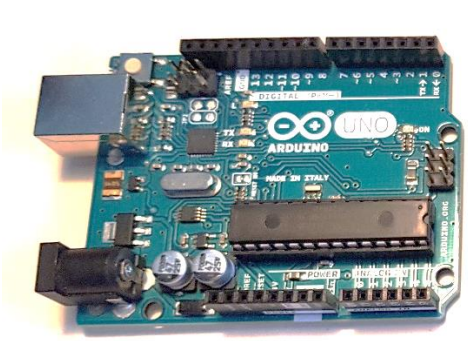
USB TO TTL KEYBOARD SHIELD

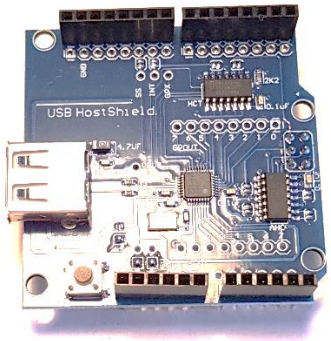
by P-LAB -- 2020

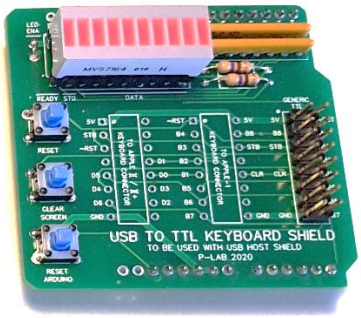


PROJECT DESCRIPTION

This Shield, designed and built for ARDUINO UNO, allows you to connect a common international USB keyboard to any vintage computer that needs a TTL keyboard.
It works together with a USB HOST Shield, which is mandatory.
Only the model "*ARCELI USB Host Shield for ARDUINO UNO*" has been tested, but other models might be compatible.







ELECTROSTATIC DISCHARGE DISCLAIMER

Electrostatic Discharges (ESD) can damage your computer and its peripherals. Before any operation, it is mandatory to connect yourself to ground potential. We do not take any responsibility and we are not liable for any damage caused by use of this product, even if indirect, special, incidental or consequential damages (including but not limited to damages for loss of business, loss of profits, interruption or the like).

PREREQUISITES

In order to complete the project you will need:

- Arduino UNO R3 or compatible,
- ARCELI USB Host Shield for ARDUINO UNO,
- *USB to TTL Keyboard Shield* (PCB version 1.1),
- A proper cable/ribbon cable suitable for your computer keyboard connection,
- Strip line connection to connect *USB to TTL Keyboard Shield* to USB Host Shield.

WHICH ADDITIONAL COMPONENTS SHOULD I MOUNT?

Actually none. It depends on you and your needs.
The *USB to TTL Keyboard Shield* does not do too much: it just adds connectors and some pushbutton.
The most interesting work is done by the firmware and it will be discussed later.
So, do you need RESET and CLEAR SCREEN pushbutton because you are planning to use the Shield with an Apple-1? Mount them!
Do you want to see how your keypresses are converted in bits and electrical signals? Mount the Resistor networks, a LED display and enjoy the *blinkenlights*!
Always check for short circuits between Shields due to long pins.
The mandatory components have been stated in the PREREQUISITES section.

CONNECTION TO A COMPUTER

That is the trickiest part: just follow the indications and everything will work fine.
Be extremely careful with every single connection or you will seriously damage your computer.
Often on vintage computers, there are voltages of +12V, -12V and -5V, in addition to the usual +5V: a wrong connection could take these voltages where they should not, **INSTANTLY** creating very serious damages.
Again, we are not responsible for any damage you may cause to your equipment.

The *USB to TTL Keyboard Shield* was created with Apple-1, Apple II and Apple II+ in mind: for this reason, you will find two DIP-16 connectors specifically dedicated to these computers.
The pinout and the nomenclature follows the original one of the computer, so it should not be too difficult to build the appropriate ribbon cable.

Double/triple check the voltage/ground of the cable before connecting it.
Make sure they end up on the correct Shield pins.
The unnamed pins are not connected to anything, so consider *safe* to drop any unwanted voltages there: they will not do any damage.
Please note that Data bits are called **B1..B8** for Apple-1/Generic TTL and **D0..D6** on the Apple II/II+ connector. Apple II/II+ use only 7 bits.

A third "Generic TTL" connector can be used for other computers.
All you need to do is to assemble the correct cable based on your computer's wiring diagram and the various signals that are printed on the *USB to TTL Keyboard Shield*.

NEVER disconnect your vintage computer from the *USB to TTL Keyboard Shield* while the power is on.

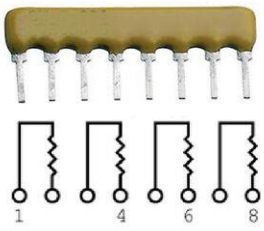
DESCRIPTION OF SIGNALS

LABEL	SIGNAL
5V	+5V: Power coming from the computer. It will power up the device
GND	GROUND
~RST	/RESET: Negated signal (Active LOW). Normally +5V. Goes to 0V when RESET pushbutton is pressed or when instructed by firmware
CLR	CLEAR SCREEN: Positive signal (Active HIGH). Normally 0V. Goes to +5V when CLEAR SCREEN pushbutton is pressed or when instructed by firmware,
STB	STROBE signal, managed by firmware. Normally 0V, it goes to +5V for 1 millisecond when a valid key is pressed
B1..B8	Data Bits, B1 it the least significant bit, B8 is the most significant bit. These signals are called D0..D7 on the Apple II/II+ connector

MORE NOTES ON THE HARDWARE

If you are not going to install pushbuttons and LEDs you can skip this section, otherwise keep in mind that:

- Pushbuttons are Normally OPEN; the contact must close on the short side.
- LED bar needs two Resistive Network and two single 470 Ohm resistors (RD1 and RD2).
- LED bar footprint is suitable for model MV57164. Others might be compatible.
- Every SIL Resistive Network is made by 4x470 Ohm, no common connection. Something like this:



Alternatively 8x470 Ohm 1/8W single resistors can be used, four for each required Resistive Network.

Resistors will be placed as shown in the picture.

- To enable the LEDs the "LED ENA" through-holes must be shorted together. If this is not done the LEDs will be isolated from GND and remain off.

MEANING OF THE LEDs

LED	MEANING
READY	If this LED is lit, the *USB to TTL Keyboard Shield* is ready and waiting your keypresses. At power-up, it will take some seconds to come up, due to the USB device recognition task. If the USB keyboard is disconnected it will go OFF. When the USB keyboard is reconnected, it will light up after a few seconds. The USB keyboard can be disconnected and reconnected without turning off the computer.
STB	Strobe signal, as described in the previous section. The light may appear weak due to the short pulse duration. This is not an indication of malfunction. Do not change the value of RD2 to make it brighter or overloads may occur.
DATA	These eight LEDs will light according to the ASCII code corresponding to the letter typed on the USB keyboard. The rightmost LED is the least significant bit (B1/D0), the leftmost is the most significant bit (B8/D7).

OPERATIONS

Normally, the firmware takes care of translating the keys you press into electrical signals that are understandable to the computer you have connected.
It takes care also of what is coming from the USB/serial port of the Arduino.

Yes: you can connect the Arduino USB port to a terminal and use it as a remote keyboard.
Simply connect to the Arduino with the USB cable to your computer and set the following parameters in the terminal program (PuTTY, minicom, etc):
9600 baud, **N**o parity, **8** data bit, **1** stop bit.

You will be able to type on your vintage computer via terminal **and** via local USB keyboard (not exactly at the same time, of course…). The two methods can coexist at che same time.
This function is useful for automatically typing programs (which you may have written elsewhere, perhaps on some text file) on the original computer, without fear of typing errors.
We will get to that later.
As mentioned above, this product was designed with Apple-1 in mind, which means that some features are designed specifically for this computer.
As you will see, the key/function mapping is slightly different from the local keyboard and the serial terminal, let us see more in detail:

SPECIAL KEY MAPPING

CLEAR SCREEN [Apple-1 only]

Clear screen is invoked with one of the following:

- **PRTSCN key** (Print Screen/SysRq) on USB keyboard
- **F11 key** on USB keyboard
- **Control+C** via terminal

RESET

Reset is invoked with one of the following:

- **Pause/Break** key on USB keyboard
- **F12** key on USB keyboard
- **Control+R** via terminal

SPECIAL FUNCTION MAPPING

HELP SCREEN

A brief help screen can be recalled by:

- **F1 key** on USB keyboard
- **Control+Q** via terminal

CHARACTERS TEST [Apple-1 only]

A small test program, which displays all characters cyclically, is automatically entered and executed by:

- **F2 key** on USB keyboard
- **Control+W** via terminal

To end the test, reset the computer.

RAM MEMORY TEST [Apple-1 only]

It is possible to load a program that tests the memory by pressing:

- **F3 key** on USB keyboard
- **Control+E** via terminal

At the end of the automatic typing, before running the actual test program it is necessary to enter the starting address and ending address+1 into locations 0-3 of memory (in Little-Endian order).
Example: to test from $E000 to $EFFF enter:

```
0: 00 E0 00 F0 {ENTER}
```

Example 2: to test from $0500 to $0700 enter:

```
0: 00 05 01 07 {ENTER}
```

To run the test:

```
280R {ENTER}
```

After some time (depending on how large the memory area to test is), you will get the results.
To end the test, reset the computer.
Please refer to the link in the bibliography for further details about the author or the test methodology.

CFFA1 MENU [Apple-1 only + CFFA1 board]

It is possible to invoke the CFFA1 menu by pressing:

- **F4 key** on USB keyboard
- **Control-T** via terminal

The address 9000R will be entered automatically and the familiar `CFFA1>` prompt will appear immediately.

EIGHTH BIT HANDLING / UPPERCASE / LOWERCASE …

Some vintage computers are able to understand, display and interpret only a subset of the entire ASCII table.
For example: Apple-1 only works with capital letters and can display only capital letters.
Any lowercase letters you enter, even if displayed correctly on the screen, will not be interpreted.
Even more: Apple-1 needs the eighth Data bit to be tied to logical level 1, otherwise it will not interpret any commands (even if they will be displayed on screen "correctly").
The situation described above is clearly out of any standard, and the fact that the submitted characters are still displayed correctly reflects the technical choices made at the time of the project (which are out of scope here).
The firmware can handle the eighth bit to create the desired mapping for Apple-1 (or not) with:

- **F7 key** on USB keyboard (default at boot) → eighth bit forced to 1, and any lowercase letter typed will be automatically capitalized. (LED corresponding to eighth bit will be ON)

- **F8 key** on USB keyboard → eighth bit will be freed. No capitalization will be made. In this condition, Apple-1 will not interpret anything, but keypresses will be still displayed on the screen. This setting might be needed for more standard computer.

EXAMPLES OF BIT MAPPING WITH DIFFERENT EIGHTH BIT SETTING:

F-KEY	KEY	OUTPUT	VALUE	APPLE-1 SHOWS	INTERPRETED CORRECTLY
F7	a	11000001	193	A	YES
F7	A	11000001	193	A	YES
F8	a	01100001	97	A	NO
F8	A	01000001	65	A	NO

If you are using an Apple II/II+, set F7/F8 according to your best experience. The Apple II/II+ does not require the eighth bit, but the self-capitalization offered by the F7 key may be handy.

WIRELESS KEYBOARDS

Some inexpensive small wireless keyboards were found to be fully functional.
Due to the lack, in some cases, of the PRINT SCREEN and PAUSE/BREAK keys, the F11 and F12 function keys must be used.

LOCALIZED KEYBOARDS

Keyboards with a localized layout (for example: Italian/Spanish etc.) will always be recognized as international and therefore some keys may not match. To overcome this inconvenience it may be necessary to modify the Arduino USB libraries.
This operation is Out-Of-Scope for this document but every support is welcome.

TELE-TYPING FROM TERMINAL AND FILE TRANSFER

When using a common terminal program, such as minicom or PuTTY, there is no special attention to be paid. Basically: what you write is what you get on the computer.
However, if you want to load a program or some ML routine that you have saved somewhere, some tricks are necessary so that no characters are lost during the transfer.
This happens because obviously no computer is designed to have keyboard input beyond a certain writing speed. If the speed becomes “super-human”, some/many characters will be lost, with the result that the program typed on the computer will be unusable.
Slowing down the Baud Rate of the serial port, besides not being enough, would have made remote writing too slow and unnatural.
It is also perfectly clear that this is not the right way to transfer programs to a computer, however, since there is the possibility... why not give it a try?
The small BASH script for Linux, below, sends to the appropriate DEVICE the file indicated at runtime, one character at a time. There are delays (sleep) obtained experimentally.
The delays below are calibrated on the Apple-1 INTEGER BASIC. The script can of course be improved, but it already works fairly well.

You may also need to finish the text files before sending them to the computer, especially regarding the line termination. The script below works well with LF ($0A, which is typical of Linux) for line termination.
If your text file comes from a Windows system, it could use CR+LF ($0D + $0A) for line termination.
This could cause, with the script below, the effect of a double ENTER.
Any contribution to improvement (especially for Windows users) is welcome.

So: if you called the BASH script with the name **'sendfile.sh**' and the file you want to send is in the same directory and it's called '**myprogram.txt**' the command will be:

```
sudo ./sendfile.sh myprogram.txt
```

Please note that the file transfer script must necessarily be launched from a new shell, while another shell is active with the terminal program running, otherwise Arduino will enter a cyclic reset loop.
Of course, you will not have to press any button in the window where the terminal program is running.

```
#!/bin/bash
DEVICE="/dev/ttyACM0"
stty -F $DEVICE 9600 cs8 -cstopb -parenb
INPUT=$1
while IFS= read -r -n1 char
do
        if [ "$char" = "" ];
                then
                echo -n -e '\x8d' > $DEVICE
                sleep 0.5
                echo
        fi
        echo -n  "$char" > $DEVICE
        echo -n  "$char"
        sleep 0.06
done < "$INPUT"
```

Be aware that the transfer is slow and should only be used for short files.
It has been calculated that with the best possible optimization, in terms of line length and internal timings, a 4 kBytes transfer would last 4-5 minutes.
The cassette interface transfers the same amount of data in about 30 seconds.

IT IS MANDATORY TO DISCONNECT THE USB ARDUINO CABLE BEFORE TURNING OFF THE COMPUTER.
If you do not do so, 5V from the USB port of your personal computer may continue to power the other computer, potentially damaging it.

BIBLIOGRAPHY / ACKNOWLEDGMENTS

Official Video of *USB to TTL Keyboard Shield*
https://youtu.be/ZlYEnyh8RC0

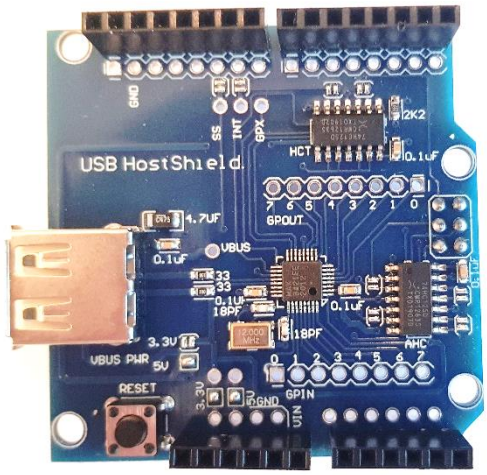
Apple-1 Operation Manual
https://www.applefritter.com/files/a1man.pdf

Mike Willegal 6502 Memory Test:
https://www.willegal.net/appleii/6502mem.htm

R&D Automation's CFFA1 page:
http://dreher.net/?s=projects/CFforApple1&c=projects/CFforApple1/main.php

ADDENDUM

In order to work properly, the "ARCELI USB Host Shield for ARDUINO UNO" pads must be configured (soldered manually, if necessary) as in the image below:



In particular:

Pad VBUS POWER 3.3V:	OPEN
Pad VBUS POVER 5V:	CLOSED
Pad OTHER 3.3V PAD:	CLOSED
Pad OTHER 5V PAD:	CLOSED