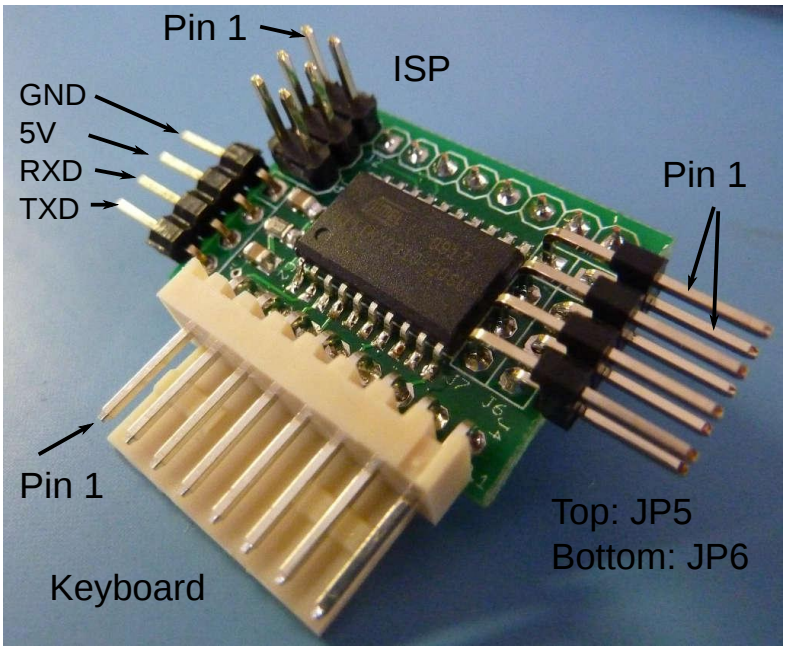
Keyboard tap to configure an Amiga 500

jmA500 ¡jm@metweb.de¿

February 16, 2018

1 Description

This little piece of hardware is used to provide some means for configuring an Amiga 500 with the keyboard. This is to replace at least some of the small switches that are commonly used to configure the extensions in an Amiga. The circuit listens to the keys pressed on the Amiga keyboard and then adjusts some of its output pins. These pins connect to jumper headers on the extension boards and hence provide some means to configure these extensions without opening the case or drilling holes into it.



Note however, that this project is not for beginners. If you have other extensions than I have, you will likely have to adapt the firmware to your needs. Also, you have to figure out whether your extensions are configurable by means of an external MCU, that is if

is suffices to drive a signal statically high or low. Switching a dynamic signal is not possible (without extra hardware)[1]

In any case you need a possibility to program the AtTiny4313 micro-controller with an in system programmer (ISP). These are cheaply available and connect to USB or parallel port. A lot of information regarding this micro-controller can be obtained on the excellent websites `http://www.avrfreaks.net/` or `http://www.mikrocontroller.net/` (German).

In addition, if you are not running a Linux machine similar to mine there might be some extra difficulties adapting the code to your build environment. In particular I am not using AVR studio and this distribution does not include any project file for it. However, pre-built firmware images are included, if you are happy with the firmware out of the box.

2 Disclaimer

I am not responsible for any damages done by this device, and I do not guarantee that it will work at all. Only connect this to your precious Amiga, if you are sure about what it does, how it works, and whether the firmware is what I claim it to be.

3 Acknowledgments

The idea to this project was born during a discussion in the `a1k.org/forum` and was suggested by Paradroid. However, if I did not have all the great hardware extensions created by the community I would not have had any reason to start the project. Thanks also to the members of the a1k.org forum encouraging me to continue this project.

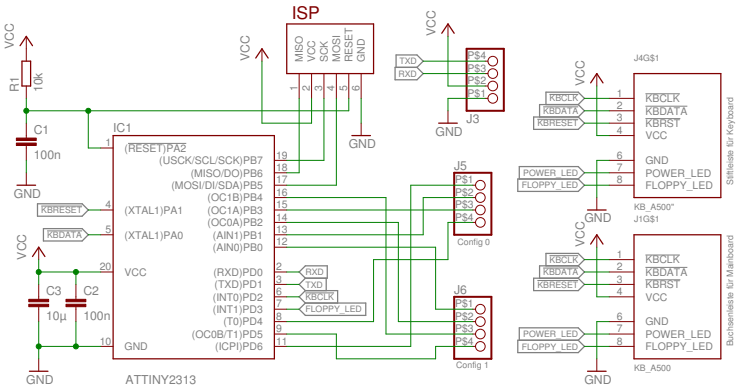
4 License

All parts of this project are free to use in non-commercial projects. The firmware is licensed under terms of the GPL 2. Please quote this document in any derived work. See the file COPYRIGHT in `src/attiny4313` for additional information, in particular the reference to Peter Fleury who provided a neat UART library for the Atmel MCUs.

5 Hardware

The hardware is plain and simple, an AtTiny4313 (or 2313) in minimal configuration with a few pin headers to connect the configuration jumpers and an ISP Programmer. In addition, the RxD and TxD pins are accessible on additional pins.

[1] I have seen some kickstart switches that use a switch to route a signal to either of the two ROMs.



There is a board layout done in such a way that the little PCB can be directly fitted on the keyboard connector on the mainboard of an Amiga 500(+).

However, the whole circuit is so simple that you might consider building it without a PCB, or you might use an Ardiuno or something similar.

5.1 Part list

IC1	AtTiny 4313 SO or AtTiny 2313 SO
C1,C2	100n SMD 0805
C3	10μ SMD 1206
R1	10k SMD 0805
J1	1x8 socket strip 180° 2.54
J2	2x3 pinheader 180° 2.54
J3	1x4 pinheader 180° 2.54
J4	1x8 pinheader 90° 2.54
J5,J6	2x4 pinheader 90° 2.54

For future extensions I suggest using a 4313 with 4k flash and 256 bytes of RAM if you can get one. The firmware distributed along with this file does not fit the 2313 if the UART functions are enabled. On the other hand, you may or may not find them useful. Note that I did not test the board with a 2313**A** although that might work without changing the code.

5.2 Pin Configuration

5.2.1 J1: Connects to A500 mainboard

1	$\overline{\text{KBCLK}}$
2	$\overline{\text{KBDATA}}$
3	$\overline{\text{KBRESET}}$
4	VCC
5	
6	GND
7	POWER_LED
8	FLOPPY_LED

5.2.2 J2: ISP

This is to connect an ISP programmer. The pin-out corresponds to Atmel 6-pin ISP connector. **Please make sure that the programmer does not power the MCU when connected to the A500 mainboard.**

1	MISO
2	VCC
3	SCK
4	MOSI
5	$\overline{\text{RESET}}$
6	GND

5.2.3 J3: Serial connector

This is a serial connection with 5V logic levels. Names are from the perspective of the MCU, so you have to connect the TxD of your computer to the RxD on the PCB.

1	GND
2	VCC
3	RxD
4	TxD

The firmware will output some status messages here, and listens to commands via a small parser. Have a look at the software how to use it (or send the string `help` to see a small help message).

5.2.4 J4: Connect your keyboard here

Same as J1.

5.3 J5, J6: Connectors for configuration jumpers

These connectors provide 8 pins to configure the hardware. Everything that happens on these pins depends on the firmware. The software provided with this documentation emulates open collector outputs, ie. it will either pull pins low or let them float (configure as inputs) but never pull them high.

This version of the firmware is used to configure generic devices. The PCB provides eight independent outputs that can be connected to various jumpers. You can configure my 1.8/2MB Ranger memory extension [**?**], a Kickflash [**?**, **?**] and the 68010@14MHz Turbo card by Matze [**?**] or any other device that has jumpers that can be activated by pulling them low. This includes any standard 512k memory extension. The Pins are labeled as follows:

J5	1	C1
	2	C2
	3	C2
	4	C4
J6	1	C5
	2	C6
	3	C7
	4	C8

Note 1: If you want to control a standard 512k expansion, connect one of the config pins instead of the switch. The switch connector on such a memory expansion has one pin which is connedted to ground, the other one is pulled high. Connect to the pin which is pulled high.

6 Firmware

The firmware for the Atmel is written in C and compiles with avr-gcc 4.8.2 but may or may not compile with different versions although I tried to be as compatible as possible. The directory `firmware` contains precompiled flash and eeprom images in Hex-format for the AtTiny 2313 and 4313. For both MCUs the fuses setting is

```
LFUSE = 0xD4, HFUSE = 0xDB, EFUSE = 0xFF.
```

Make sure to program both the flash memory with the corresponding `.hex` file and the eeprom memory with the `.eep` file. Never attempt to flash the MCU while plugged into a running Amiga 500, since it will switch configuration when finished programming. This could damage your Amiga.

The Makefile provided in the `src` directory should do the job by simply running `make`. If compiling for an AtTiny2313 you must adjust the `MCU` variable in the Makefile to read `MCU = attiny2313`. Also flash is too tight to use UART debugging. To disable it, open `main.c` and comment out the line `#define UART_DEBUG 1`. If you want to use

the UART features, us an AtTiny4313. If you happen to have a USBasp ISP programmer [**?**] you can program the MCU by simply running `make fuses` and `make program`. The Makefile supports programming with avrdude, the programmer is selected with the variable `AVRDUDE_PROG`.

The source code should be mostly self explanatory[2], although small code has been preferred to readable code in some places. The crucial global variable is `conf`. There are two bytes, `current` and `next`. Each bit in those variables corresponds to one config pin. **A one in these bits results in pulling the corresponding pin low.** The main loop checks whether one of the F-keys is pressed before the *Help*-key gets pressed three times and changed the next configuration accordingly. If it detects a keyboard reset the new configuration gets written to the pins and is stored in the eeprom.

7 Usage

This is the selections available if the configurator is connected as described in section 5.3 and programmed with the *generic* firmware. The keys F1-F8 are used to switch the configuration pins as follows:

F1	set C1	Shift+F1	clear C1
F2	set C2	Shift+F2	clear C2
F3	set C3	Shift+F3	clear C3
F4	set C4	Shift+F4	clear C4
F5	set C5	Shift+F5	clear C5
F6	set C6	Shift+F6	clear C6
F7	set C7	Shift+F7	clear C7
F8	set C8	Shift+F8	clear C8

To change the configuration proceed as follows:

1. Push *Fx* or *Shift-Fx* if you intend to enable (resp. disable) configuration pin *x*.
2. Hit the *Help*-key three times. This changes the next configuration that gets activated but does not yet switch the configuration pins.
3. You can repeat steps 1 and 2 to change further pins.
4. To activate the configuration perform a keyboard reset. This also stores the new configuration setting to eeprom so that it loads on the next power on.

If you accidentally select a configuration that does not run and is screwed in such a way that the keyboard is not initialized, you can do a keyboard reset for at least 6 seconds to load the hopefully save configuration where all pins are disabled. This fail-safe configuration is defined by the macro `CONFIG\_DEFAULT` in the file `ee\_helper.h`.

[2] An obviously biased judgment YMMV.

References

[1] Kickflash Manual. http://www.amigawiki.de/doku.php?id=de:projects:kickflash_a500.

[2] Kickflash Thread. http://www.a1k.org/forum/showthread.php?t=38329.

[3] Trapdoor extension Thread. http://www.a1k.org/forum/showthread.php?p=751285.

[4] Turbokarte von Matze. http://www.a1k.org/forum/showthread.php?t=33003.

[5] Usbasp. http://www.fischl.de/usbasp/.