
GND
VCC
B0
F0
CSA
28AWG

dasta400/ACPCPE

dasta400 ⋮

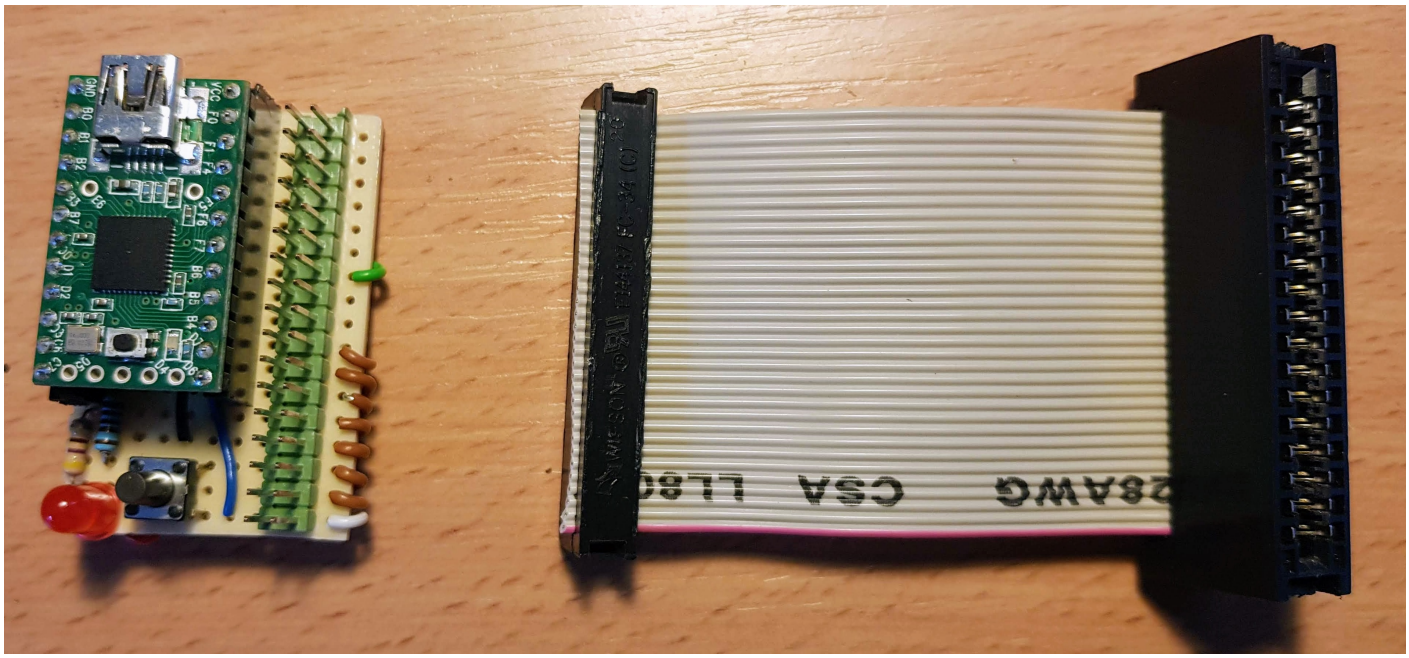
dasta400/**ACPCPE**

Amstrad CPC Printer Emulator

1 Contributor 0 Issues 7 Stars 1 Fork

README.md

ACPCPE v2.0 (A CPC Printer Emulator)



The (exchange the A for *Amstrad*, *Arduino*, *Awesome*, *Awful* or whatever you want) CPC Printer Emulator is a hardware and software project that allows to connect the Amstrad CPC's Printer port to a PC that

acts as Epson-compatible dot matrix printer and generates files instead of printing on paper.

CPC's Printer Port (Parallel) => Arduino pins => Arduino USB (Serial) => PC USB Serial port

With the ACPCPE it is possible to *print* from a real Amstrad CPC to a PC that acts as virtual printer and generates Text, HTML and Markdown files. **This README was written on a real Amstrad CPC6128 using Protext and then *printed* as Markdown file on a Linux machine**.

It can be used not only for word processor documents, but also for exporting BASIC programs to a PC and use them on an emulator. Simply *print* the program using Locomotive BASIC command *LIST #8* and then take the .txt file from your PC into the emulator (e.g. WinAPE's Auto Type option).

But why printing from an Amstrad CPC in 2019?

I know it sounds a bit crazy, but I still use sometimes my Amstrad CPC6128 for word processing. Yes, I'm a bit crazy. I'm not going into details to try to justify myself. It simply works for me and I enjoy it. Did I mention that this README file was written on a real Amstrad CPC6128 with Protext?

How it works?

The Amstrad CPC Printer Port is connected to an Arduino that reads the /Strobe and 7 Data signals. The Arduino software translates the 7 data bits into a single 8-bit byte and transmits it through the USB cable to a connected PC running a Python program that interprets the data and generates text files.

What's needed?

- Amstrad CPC (I only tested with CPC6128).
- Arduino-compatible board (I used Teensy 2.0).
- 34pin edge connector (I scavenged one from an old PC floppy cable).
- PC with USB port (so something modern-ish).
- Python 3.x
    - pySerial (https://pyserial.readthedocs.io)
    - USB cable to connect Arduino to PC.
- Push-button + 1K resistor for the Online/Offline.
- LED + 450 ohm resistor.

The Amstrad CPC

As mentioned above, the Amstrad CPC uses a 7-bit data bus for the printer instead of the more typical 8-bit, which forms a full byte. Recently I looked at the schematics and I think the reason behind this seems to be that the CPC is using a 74LS273 (octal flip-flop) for the Data and Strobe signals. Being of octal type means it can only carry eight signals. So my guess is that the designers at Amstrad had to sacrifice one Data signal to add the Strobe signal, because they already run out of output signals on the 8255 PPI which would have been my first choice, and thus ended up with 7 signals for the Data which it's enough for a character set of just 128.

Also, strangely Amstrad decided despite using a 34-pin edge connector to not use any of the typical Centronics signals and reduce the whole interface to 9 pins (/Strobe, Busy and 7 data). This means there

is no way for the Amstrad to; know when the printer is out of paper, send a reset, send a linefeed or even use the Ack signal.

They could have used instead a 9-pin edge connector or even a D-sub 9- pin connector.

For a detailed pinout of the Printer Port, check the Amstrad CPC's User Instructions manual or visit www.cpcwiki.eu/index.php/Connector:Printer_port

The Arduino

Connected to the Amstrad CPC's Printer Port and to a listening PC through a Serial port:

- uses Interrupt Service Routines to detect when the signal /Strobe goes low, and when the signal Online/Offline has changed.
- Once /Strobe goes low, reads all 7 pins for data, translates them into an 8-bit byte and send it through the serial port to a listening PC.
- When Online signal is detected, sends the *Select printer* ESC/P command and sets the *printer* as Ready.
- When Offline signal is detected, sends the *Deselect printer* ESC/P command and sets the *printer* as Busy.

I think it should work with any Arduino, but as we only need 9 pins anything above Arduino Micro or Nano is a bit of an overkill. In my case I only have MEGA2560, Uno and Leonardo, so instead I ended up using a Teensy 2.0 (http://pjrc.com) I had lying around.

The Python program

ACPCPE.py processes the bytes received from the Serial Port and generates different text files (.txt, .html, .md).

It is compatible with *Epson Standard Code for Printers* (ESC/P).

Word processors for the Amstrad CPC (e.g. Protext, Amsword, Tasword) send ESC/P codes to make a printer to print different types of fonts, set the size of margins, jump page, etc.

The received 8-bit bytes are processed as follows:

- If the byte corresponds to a *Select printer* command, it adds all following bytes to a buffer in memory.
- If the byte corresponds to a *Deselect printer*, it stops listening, starts interpreting each byte from the buffer and finally generates the output file(s).
- For each received byte, if it represents a printable ASCII character, it's simply written to the output file.
- If the byte represents an ESC/P command, it's interpreted and output is flagged for generating HTML (and/or Markdown) file as result.

Python usage

python ACPCPE.py -h

positional arguments:

```
port             serial port to listen from.
```

optional arguments:

```
-h, --help       show this help message and exit
-e, --echo       output data to screen too
-q, --quiet      suppress non-error messages
-nf, --nofile    do not output to file
-nl, --nolog     do not create a log file
-r, --raw        generate a file  (.raw)  with the received hex values
-md, --markdown  output file will be Markdown instead of HTML
-v, --version    show program's version number and exit
```

Supported ESC/P codes so far

```
10              Line Feed
17              Select printer
19              Deselect printer
13              Carriage Return
27 64           Initialise printer
27 45 48        Cancel underlining
27 45 49        Select underlining
27 52           Select italic mode
27 53           Cancel italic mode
27 69           Select emphasised (bold) mode
27 70           Cancel emphasised (bold) mode
27 71           Select double strike mode
27 72           Cancel double strike mode
27 83 48        Select superscript
27 83 49        Select subscript
27 87           Cancel superscript/subscript
```

What's next?

- To add more supported ESC/P commands.
- Support for graphics?

Changelog

- v1.0 - debounce button using an NE555 in bistable mode.
- v2.0 - sofware debounce button. NE555 removed.