CapriceESP32

(A version for the Odroid GO Hardware)

Introduction

At the time the Amstrad CPC came out in the 80th, I was fascinated on what lies within the creativity of computer games. It was my first computer where me and my brothers spent days and nights playing games. What a perfect time looking back from now. It was also the time I got fascinated by how a computer works, I have started to learn digital science and computer programming, finding hacks and game cheats in first place. Well, there is always a way to start with :-)

Now, sever centuries later, I came in touch with the Odroid GO by searching for ESP32 programming and evaluation board references. I liked already the ESP8266 for doing some nice IOT projects on my own for home automation purpose. Then, after purchasing an Odroid GO, I have started to upload some nice EMUs checking some games.

Well, then I was looking for a Amstrad CPC EMU for the Odroid GO. As far as I know, there is no public version available. I was caught by the idea playing my favorite games I was addicted to in my youth 30 years later on this tiny ESP32. How could such a big machine fit to half a square inch of space?

Then I started with the end of mind. Searching for a simulator platform saving me some development time capable to develop the wrapper on a decent PC. On the Amstrad CPC, I searched for a good starting point to develope from. I decided to go for the Caprice Palm OS version. It was the most flexible and most decent version of the original Caprice32 version. Still a mystery to me how to develop an EMU. Well done Ulrich Doewich and Frederic Coste! Every thing else is history.

Here we go, a first pre-release of the ESP32, Odroid GO version of the Caprice CPC engine is made available here. What needs to be done to get going is described below.

Preparing the SD Card

```
>|whatever_sd_card_name_you_take/
>|→ /cpc/dsk  :: Search path for DISK images
>|→ /cpc/cps  :: Default snapshot path (not used yet)
>|→ /cpc/scr  :: Default screenshot path (not used yet)
>|→ /cpc/cmf  :: Default cheat file path
>|→ /cpc/kmf  :: Default keymap file path
```

It is very important to setup the SD card accordingly. Without that it won't work. This configuration is uses to be standard compliant to Odroid GO Emulator setup. In general used to keep multiple EMUs organized on one single SD card. The **/cpc/dsk** path may contain sub-directories. Please not that sub-directories cannot be stepped into at the moment. So please put all your *.dsk disk images into the root directory of: **/cpc/dsk.**

Compile from source

Normally, there is no need to compile from source. The repository contains two executable versions. One to use with *:>make flash (monitor)*, the other as a FW image which can be loaded through the Odroid GO Firmware process from SD card. As platform, the esp_idf version 3.2 is uses. It is basically the standard Odroid GO development platform. I have added also Cmake support for the actual esp_idf version 4.x for testing purpose. It should compile, however, I have stopped development on 4.x because no performance improvements determined.

Running the Emulator

After compile and flash, or FW upload, you should see the following screen:

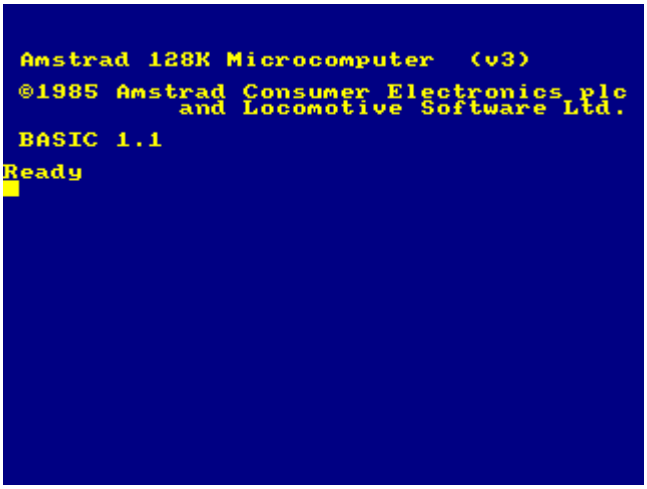


Illustration 1: Start screen of the Amstrad CPC Emulator on the Odroid Go

By connecting the serial monitor, you will get additional information about frames-per-second, audio stream, battery status, keypad status, and CPU load. In general, you have the following two options to navigate:

The virtual keyboard:

The virtual keyboard can be launched by pressing **>M< (Menu) + >A< (Button)**. When inside the virtual keyboard, the navigation keypad is attached to it. You can move, left, right, up, down to select a character. The selected character is then highlighted red. When pressing **>A<**, the current character will be emulated. You can leave the virtual keyboard by pressing **>M< (Menu) + >A< (Button)** again. Then, the navigation attachment falls back to joystick support.

Illustration 2: Virtual Keyboard of the CapriceESP32 Emulator by pressing >M< (Menu) + >A< (Button)

The virtual keyboard size was kept as low as possible in order to overlay as minimum as possible with the emulator screen. The size and position are fixed. You can't move the virtual keyboard on the screen at the moment.

Loading and Running games.

On a standard Amstrad CPC, the Basic OS is always booted at startup prompting with Ready. What one normally does is to put a disc into the drive and type >CAT< (catalog). Then the system lists all available files. By typing >run" + <filename><, the system executes the file selected. This procedure is normally sufficient to load and run games.

This is a bit unhandy on a device without hardware keyboard. So the procedure was simplified on the Odroid GO. In order to load a game, you need to press **>M< (Menu) + >B< (Button)** to get into the context menu:

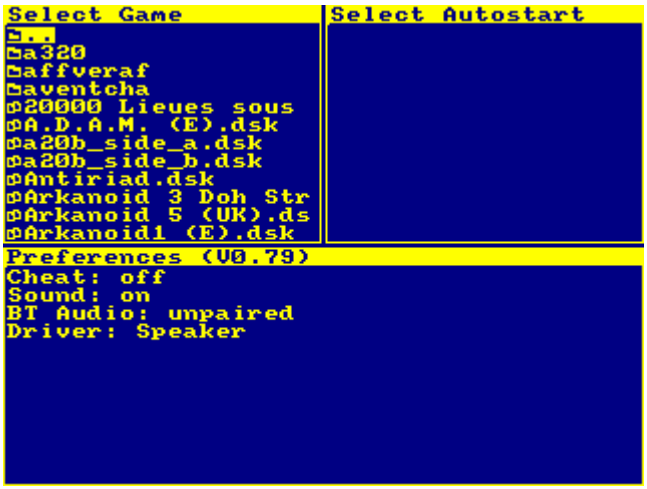


Illustration 3: Context menu by pressing >M< (Menu) + >B< (Button)

You will see three sub-panels. The active panel (highlighted yellow) is the game select panel by default. With the cursors you can move up and down inside this window. The actual one is always highlighted yellow. As a limitation so far, folders cannot be entered. I might implement this in future. Then, by pressing **>A<**, you will see the available files on this disk selected. In this

particular case “Ghosts ‘n Goblins” has been selected. In order to get now in the Select Autostart panel, you need to press >**M**< + >**RIGHT**<.

Illustration 4: Autostart file selection process

Panel navigation is implemented when >**M**< + >**LEFT, RIGHT, UP, or DOWN**< is pressed. So you can switch between **Select Game** and **Select Autostart** by pressing >**M**<+>**RIGHT**<, or >**M**<+>**LEFT**<, respectively.

Please note: In case the catalog is already seen in the **Select Autostart** panel, the disk is loaded already into the emulator. One, can also switch back to the emulator by pressing >**M**<+>**B**< again. This is useful when another disk needs to be loaded, for instance games provided on multiple disks. Don’t press >**A**< in such a situation. Then, the emulator will be reset by moving out of the context menu. It will enter the Autostart feature.

Please also note: To start the autostart process, one need to press >**A**< on the particular file selected in the catalog. In this example it is “ELITE.BAS”. There is no visual feedback to the user that Autostart has been initiated. Please do not wait for an on-screen-display (OSD) window giving feedback on this. If you now go back to the emulator window by pressing >**M**<+>**B**< , the emulator will reset and start typing the file name to run.

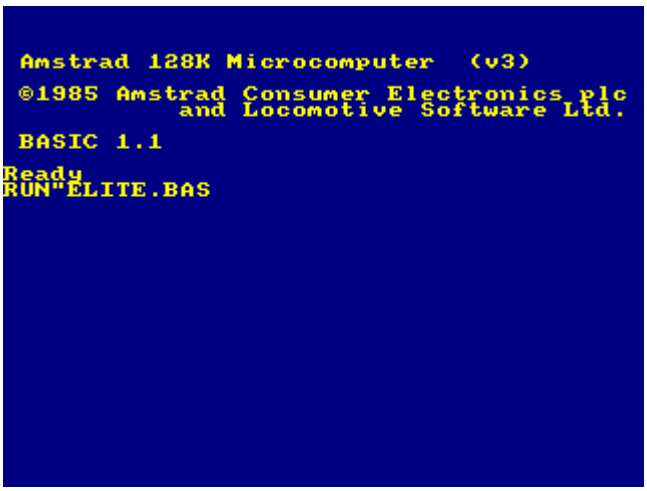


Illustration 5: Autostart execution

After that, the game should load and you are ready to play!

Illustration 6: Game screenshot

During the game you can press **>M<+>A<** to get to the virtual keyboard. By pressing **>M<+>B<** you can hold on the emulator, or you can select another game to play.

Cheating with Peeks and Pokes

Well, my gaming skills did not improve since then. So, I decided to implement a cheat mode feature to overcome some boring sequences of particular games. This idea is not new, it is already well established and there exists a huge cheat database evolved throughout the years.

This was possible because the Amstrad CPC architecture comes with a very basic memory interface. It is accessible through BASIC with *peek()* and *poke()* commands. When you have the program loaded into memory, you can start hacking. These were also the good times, when computer programming was much simpler (without any security measures preventing that).

How does it work now for the CapriceESP32 emulator? First you need to go and search for the "*Amstrad CPC Poke database*", or you have already your own to look up. An example how to generate a Cheat mode file for the emulator is given in the appendix. It is an Octave script, but shows how the data is prepared. It might be easy to port it to other languages.

The format can handle multiple Cheats with multiple Pokes as such. In such a case, all the Cheats have to be listed first, followed by the list of all Pokes. For an individual Cheat, the list of Pokes need to be in ascending order, where the *cheat.index* value corresponds to the starting index of the corresponding Poke-list. The *cheat.length* value holds the length of the corresponding Poke-list.

Defining your personal Keymap

In some cases it is handy to have a personal keymap file available. This avoids the use of the virtual keyboard to make configurations or of starting games frequently. Here, I followed the concept of the C64 emulator by Schumi and have taken over identical naming conversion for convenience. So a file might look like:

[KEYMAPPING]

UP = JST_UP
RIGHT = JST_RIGHT

DOWN = JST_DOWN
LEFT = JST_LEFT
SELECT = 1
START = 2
A = JST_FIREA
B = JST_FIREB

Since the keyboard is a bit different, here are the supported keys implemented so for. In case you need further keys, please let me know.

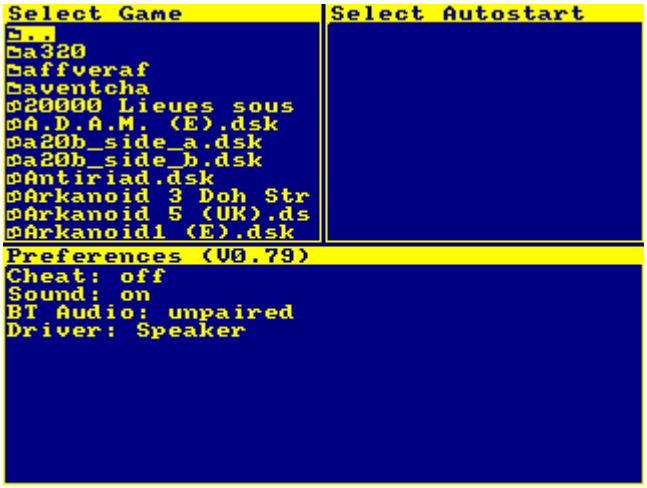
```
    if (!strcmp(mappingString, "JST_UP")) return(CPC_KEY_J0_UP);
    if (!strcmp(mappingString, "JST_RIGHT")) return(CPC_KEY_J0_RIGHT);
    if (!strcmp(mappingString, "JST_DOWN")) return(CPC_KEY_J0_DOWN);
    if (!strcmp(mappingString, "JST_LEFT")) return(CPC_KEY_J0_LEFT);
    if (!strcmp(mappingString, "JST_FIREA")) return(CPC_KEY_J0_FIRE1);
    if (!strcmp(mappingString, "JST_FIREB")) return(CPC_KEY_J0_FIRE2);

    if (!strcmp(mappingString, "JST2_UP")) return(CPC_KEY_J1_UP);
    if (!strcmp(mappingString, "JST2_RIGHT")) return(CPC_KEY_J1_RIGHT);
    if (!strcmp(mappingString, "JST2_DOWN")) return(CPC_KEY_J1_DOWN);
    if (!strcmp(mappingString, "JST2_LEFT")) return(CPC_KEY_J1_LEFT);
    if (!strcmp(mappingString, "JST2_FIREA")) return(CPC_KEY_J1_FIRE1);
    if (!strcmp(mappingString, "JST2_FIREB")) return(CPC_KEY_J1_FIRE2);

    if (!strcmp(mappingString, "KBD_SPACE")) return(CPC_KEY_SPACE);

    if (!strcmp(mappingString, "KBD_F0")) return(CPC_KEY_F0);
    if (!strcmp(mappingString, "KBD_F1")) return(CPC_KEY_F1);
    if (!strcmp(mappingString, "KBD_F2")) return(CPC_KEY_F2);
    if (!strcmp(mappingString, "KBD_F3")) return(CPC_KEY_F3);
    if (!strcmp(mappingString, "KBD_F4")) return(CPC_KEY_F4);
    if (!strcmp(mappingString, "KBD_F5")) return(CPC_KEY_F5);
    if (!strcmp(mappingString, "KBD_F6")) return(CPC_KEY_F6);
    if (!strcmp(mappingString, "KBD_F7")) return(CPC_KEY_F7);
    if (!strcmp(mappingString, "KBD_F8")) return(CPC_KEY_F8);
    if (!strcmp(mappingString, "KBD_F9")) return(CPC_KEY_F9);

    if (!strcmp(mappingString, "KBD_COPY")) return(CPC_KEY_COPY);
    if (!strcmp(mappingString, "KBD_LEFT")) return(CPC_KEY_CUR_LEFT);
    if (!strcmp(mappingString, "KBD_UP")) return(CPC_KEY_CUR_UP);
    if (!strcmp(mappingString, "KBD_DOWN")) return(CPC_KEY_CUR_DOWN);
    if (!strcmp(mappingString, "KBD_RIGHT")) return(CPC_KEY_CUR_RIGHT);
    if (!strcmp(mappingString, "KBD_LSHIFT")) return(CPC_KEY_LSHIFT);
    if (!strcmp(mappingString, "KBD_RSHIFT")) return(CPC_KEY_RSHIFT);
    if (!strcmp(mappingString, "KBD_CONTROL")) return(CPC_KEY_CONTROL);
    if (!strcmp(mappingString, "KBD_BS")) return(CPC_KEY_BACKSLASH);
    if (!strcmp(mappingString, "KBD_TAB")) return(CPC_KEY_TAB);
    if (!strcmp(mappingString, "KBD_CAPSLOCK")) return(CPC_KEY_CAPSLOCK);
    if (!strcmp(mappingString, "KBD_HOME")) return(CPC_KEY_CUR_HOMELN);
    if (!strcmp(mappingString, "KBD_ENTER")) return(CPC_KEY_ENTER);
    if (!strcmp(mappingString, "KBD_RETURN")) return(CPC_KEY_RETURN);

    if (!strcmp(mappingString, "KBD_ESCAPE")) return(CPC_KEY_ESC);

    // handle single Chars
    return(palm_kbd[(mappingString[0] & 0x7F)]);
```

Note: A key mapping file must be placed in the /kmf sub-folder with identical spelling to the .dsk file name it shall be applied to, but with .kmf extension. Volume button and Menu button are not available to modify. They are needed in their default configurations.

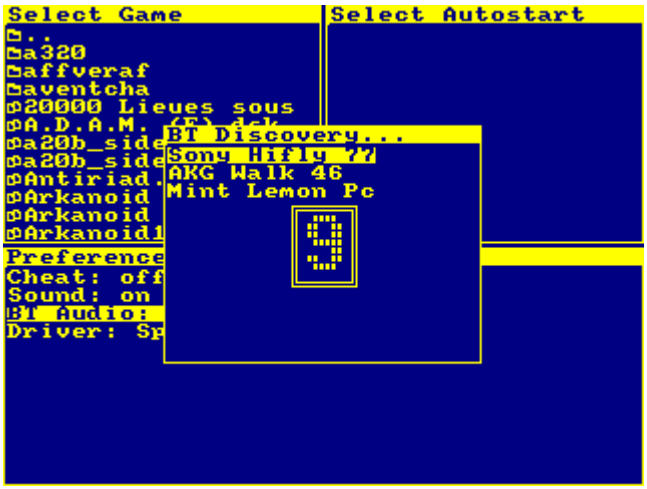
Bluetooth audio streaming support

Since the ESP32 is capable to support Bluetooth audio A2DP source profile out of the box, I have added it to the CapriceESP32 emulator. It needs a lot of event handling. I might have missed one or the other, but it should work. How to get connected..

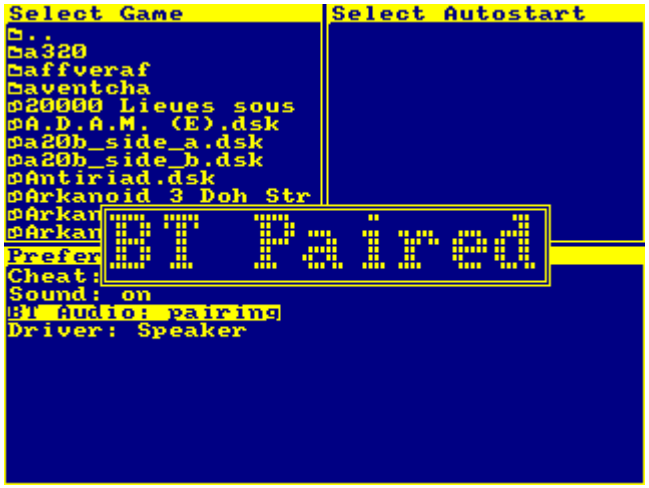
First, you need to go into the context meny by pressing >M< + <B>



With >M< + >DOWN< you can move the focus to the Preferences window. Select the BT Audio: item and press >A< to start the Bluetooth discovery mode. In case you have a A2DP capable rendering device ready (Like BT speaker, BT Earpods, BT Headset, BT PC), you will find a list of possible devices, the Odroid Go can connect and stream audio. Still, in some cases, it cannot connect, I have no idea why.



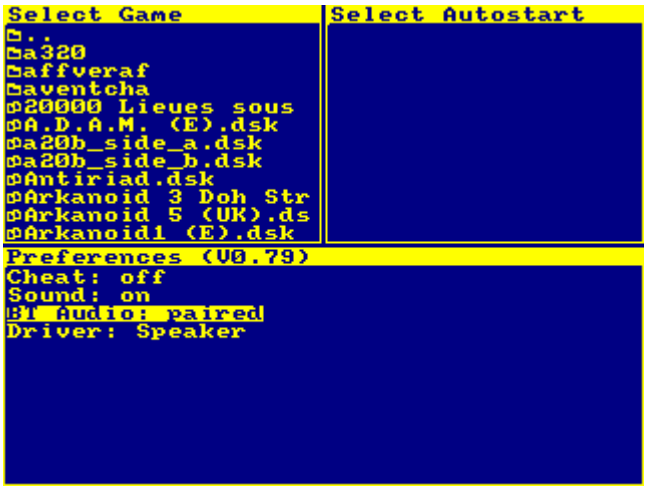
You need to wait until the timer expires. I have set it to 10 seconds, should be OK to discover all audio rendering devices close by. After timer expiry, you can select one of your list, to connect by pressing >A<. After that, you should get first **BT Paired** message



then **BT Ready** message

If this is not the case, there is an issue with the connection, and unfortunately your audio device cannot be used.

If everything goes well, you should see now **BT Audio: paired** highlighted



Now a BT connection is established and you can switch the Driver from internal speaker to BT audio. You may go down to the Driver option then press >A< and you will enable BT sound streaming to your BT device. You can switch back to the internal speaker on the fly, by pressing the Driver option again. In case, there is no active BT connection, you will get an error feedback. You might shut down the Odroid GO, and start over again.

Some BT devices support remote wakeup and can connect automatically on startup, if one time paired, like the Jabra Elite 65t. So when the EarPods are active and the Odroid GO is switched on, it will automatically connect and the Odriod GO will switch to the BT streaming automatically. I find this convenient and have enabled this feature. There is currently no feature to disable this functionality by default. You can still disconncet by pressing on the BT audio option, when **BT Audio: paired** is visible.

Known Limitations

Some games that use overscan are not displayed fully. Overscan means that it uses the full border area for the game. The current bit-blit algorithm implemented does not use scaling and blending. I might implement this feature in future.

Some remakes or ports like *Pinball Dreams* do not work correctly. I have checked that it is also not working correctly in Caprice32 and Caprice Palm OS version. It works on Caprice Forever. So I need to approach Frederic Coste how to solve it. It is something deeper in the emulator.

Settings are not implemented so far. Like:
- green monitor feature,
- switching from Joystick to Cursor keys, or other per-defined hardware key arrangements,
- night mode
- screenshot
- game snapshot (to continue playing on a later stage)
- of course there are memory leaks and other bugs need to be fixed

I might work on this in future.

One final remark

Main idea of this project was to get it working real time on an ESP32. During this project I have learned a lot about the esp_idf (I just used it with Arduino previously). I have learned how an real-time OS works in reality and how a dual core CPU can be used to increase performance. It was also a topic related to two things:

- There are always two creations, one in mind and one in reality!

- Always start with the end of mind!

Have Fun! But maybe two years to late for the Odroid GO!

In case of mistakes and spelling, please reach me!

Cheat mode format (.cmf)

```
% (C)heat (M)ode data (F)ormat
%
% First! File name has to be identical with disk image name with the extension ".cmf".
%
% Structure definition
% Header:
%   int  magic;                     // magic is .cmf 2E636D66
%   char title[64];                 // normally the game title the cheat corresponds to
%   int  index;                     // which cheat shall be used in the game
%   int  length;                    // number if individual cheats the file     contains
%
% Cheat:                            // a single cheat contains
%   char description[64];           // description of the Cheat (lives, power , invulnerable,..)
%   int index;                      // the start index of the poke list (base = 0)
%   int length;                     // the length of the poke list
```

```
%
% Poke:                             // an atom Cheat consists of the trupple: address, the poke code and the original code
%   short address;                  // memory address
%   byte cheat_code;                // poke value
%   byte original_code;             // original value, please note: in case the original_code does not match,
                                    // the cheat will not be applied

header_game_title = 'Arkanoid1';

text_max_length = 64;
header_magic = hex2dec('2E636D66');
header_index = 0;
header_length = 1;

cheat_description = 'Infinite Lives P1';
cheat_index = 0;
cheat_length = 1;

poke_address = '0x02f3';
cheat_code = '0x00';
original_code = '0x35';

file_name=[header_game_title,'.cmf'];

%%  open the file
fid = fopen(file_name,'wb','ieee-le');

%% write the header
fwrite(fid,uint32(header_magic),'uint32');

for n=1:text_max_length
    if n<=length(header_game_title)
        c_char = double(header_game_title(n));
    else
        c_char = 0;
    end
    fwrite(fid,c_char,'uint8');
end

fwrite(fid,uint32(header_index),'uint32');
fwrite(fid,uint32(header_length),'uint32');


%% write the cheat
for n=1:text_max_length
    if n<=length(cheat_description)
        c_char = double(cheat_description(n));
    else
        c_char = 0;
    end
    fwrite(fid,c_char,'uint8');
end

fwrite(fid,uint32(cheat_index),'uint32');
fwrite(fid,uint32(cheat_length),'uint32');

%% write the poke
fwrite(fid,hex2dec(poke_address(3:end)),'uint16');
fwrite(fid,hex2dec(cheat_code(3:end)),'uint8');
fwrite(fid,hex2dec(original_code(3:end)),'uint8');

fclose(fid);
```