# VIDEOPAC FPGA

FPGA Core Odyssey2 / Videopac G7000

## Charset editing

The new Core that implements the operation of the **Magnavox Odyssey2** console, known in Europe as the **Philips Videopac G7000** (hereafter Videopac), allows customization of the character set that resides in the original machines on its graphics and sound chip, the I*ntel 8244/8245*. *https://github.com/RW-FPGA-devel-Team/Videopac-G7000*

To do this, from its OSD menu, in the option "**Change VDC font *.CHR**" you can load the character set you like the most.


You can find some of them already created in the same folder of this manual, as for example ‘charset_videopac.chr’ or ‘charset_quads.chr’.

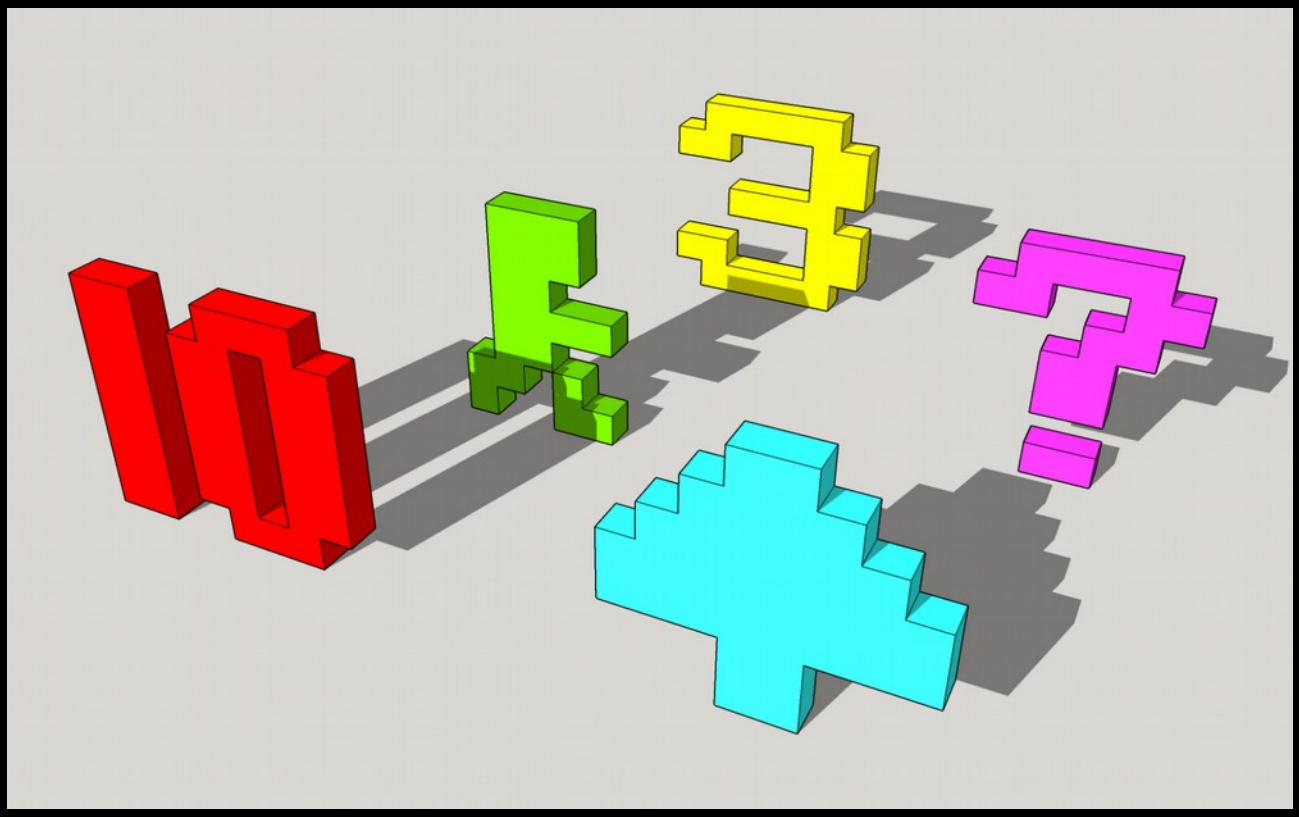
It is recommended that you have a folder called 'CHR' inside your Videopac game folder.

### How can I modify them?

Its editing could not be simpler and to learn there is nothing better than an example. Let's work on a practical case:

The Videopac is capable of handling hardware sprites. But sometimes it can also be fun (whether or not you are a game programmer) to manipulate the 64 characters found in the ROM, specifically inside its core, the aforementioned Intel custom VDC chip, which generates the video signal and audio pulses, among other things.

Each of these 64 characters is formed by a cell of 8x8 pixels.

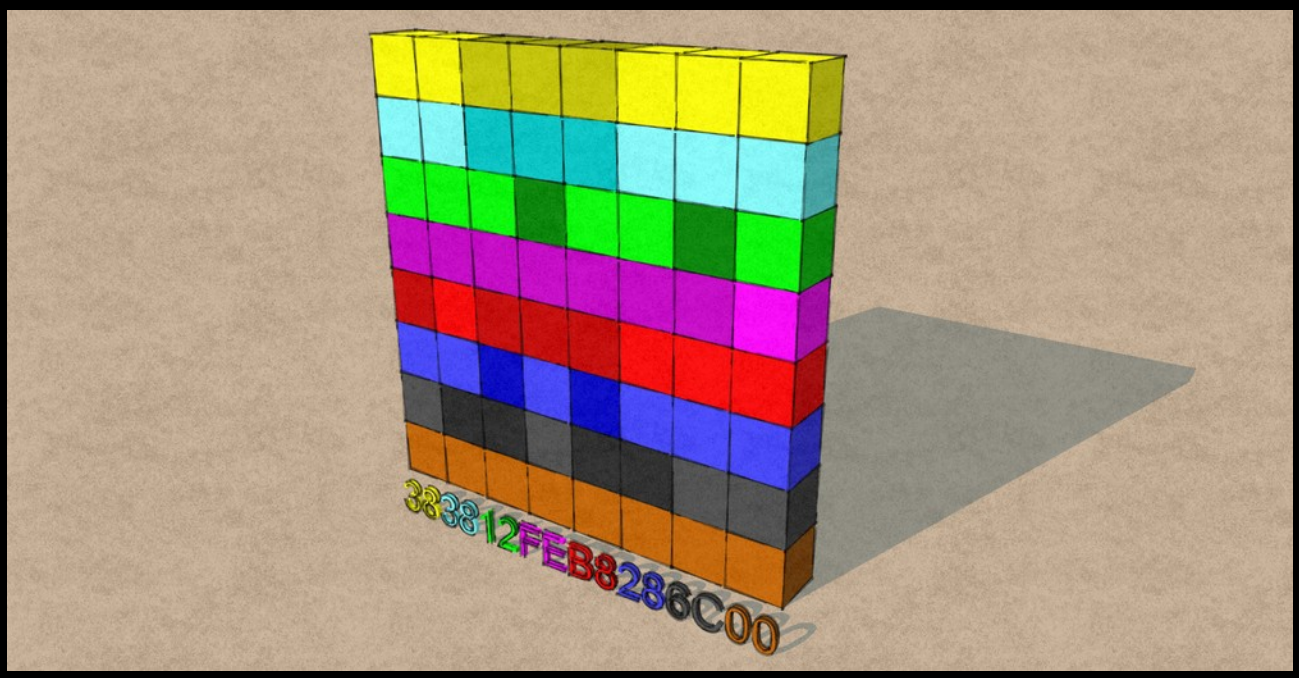


For the manipulation of these image points we have our own Charset Editor. A small html code that allows you to conveniently turn off or on each pixel (left and right mouse buttons).

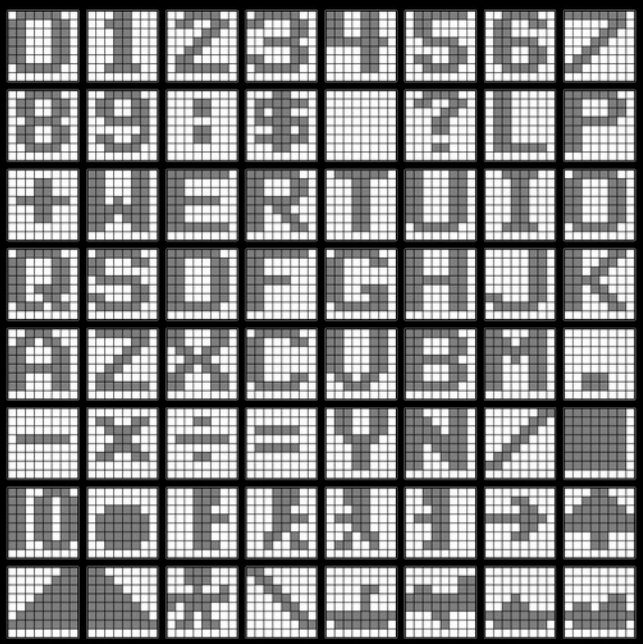
You will notice that a string of numbers and letters is displayed at the bottom. These are the hexadecimal codes that represent the image you are creating.

Each 1x8 pixel line is stored occupying one byte of ROM memory on the chip. Therefore each 8x8 pixel character consumes 8 bytes of space.

You can see in this image the zeros and ones represented by light and dark tones, in 8 colors for the 8 bytes...



In the Videopac the memory space reserved for these characters (64 in total) is 512 bytes. These are the original characters of the machine. We are going to work with them...



## Some considerations

As you can see, the number '0' and the letter 'O' have the same pixel configuration, so we could do without the letter 'O' and when you need it, use the zero instead. But we can also encourage you with other changes.

We have other characters that invite manipulation:

- In the character 'O' we could put the letter 'ñ', so needed in Spanish.

- The character "$" could be replaced by "!". I think that in texts for video games, the exclamation is more valued.

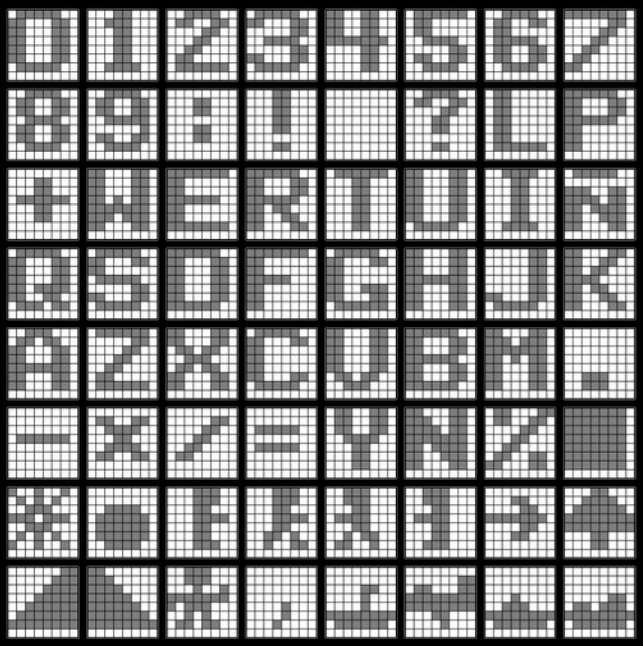
- The division sign can be enhanced with slash.

- The original slash character for the percent sign.

- The character "10" (which is of little practical use) for an asterisk.

- The character "," that we would put in the place of the backslash and would give us a double service: The tilde for accented vowels and the comma itself.

The set would look like this...

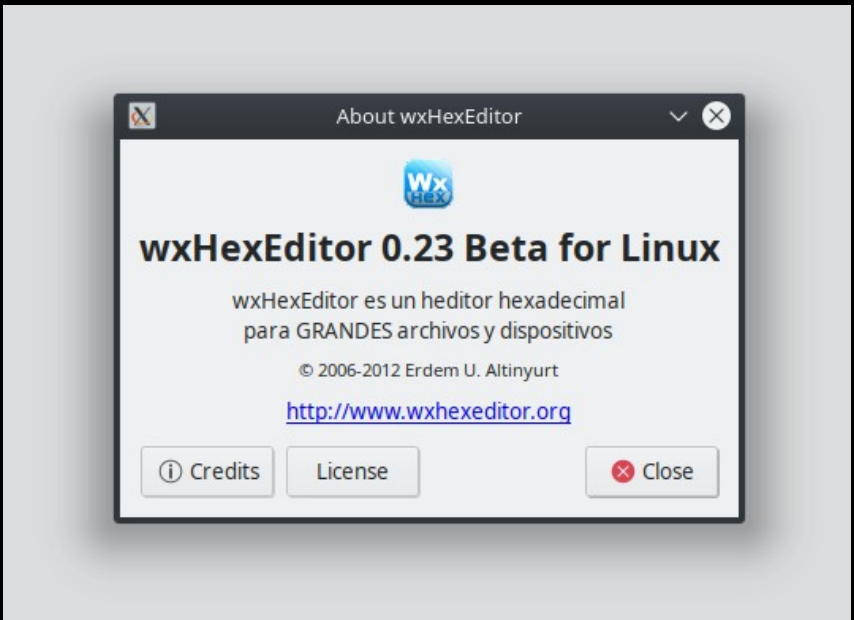


We will need 3 things:

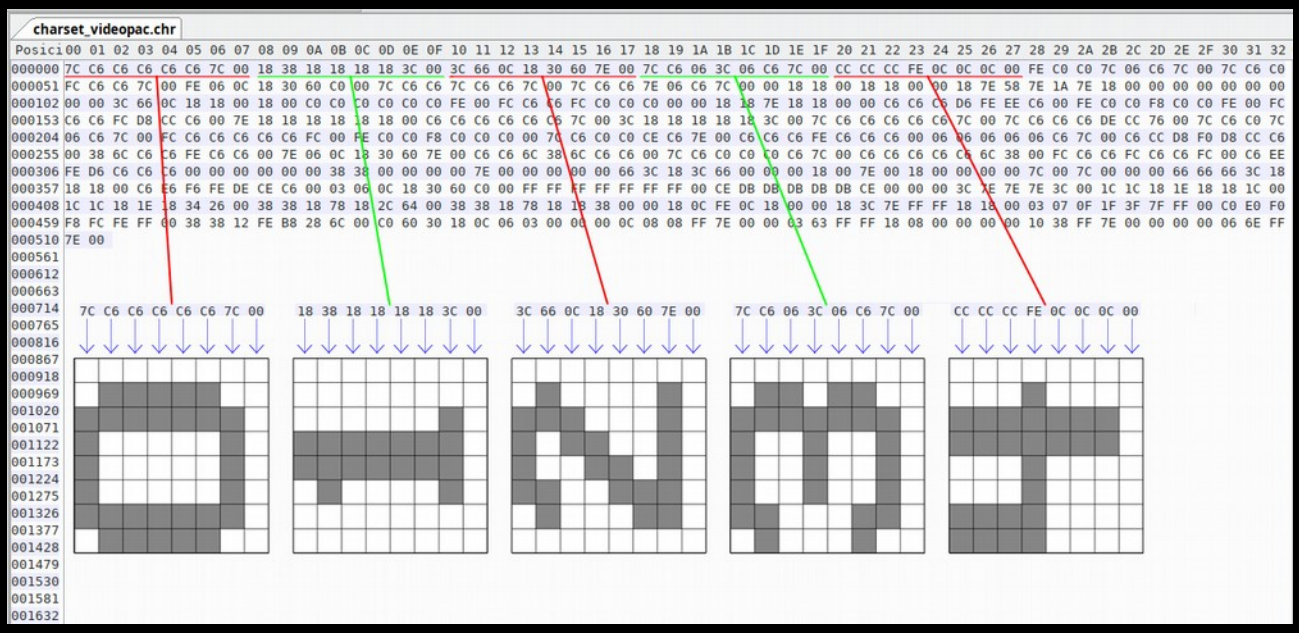
- The file "*charset_videopac.chr*" with a length of 512 bytes.

- The graphic editor "*Charset_Editor.html*".

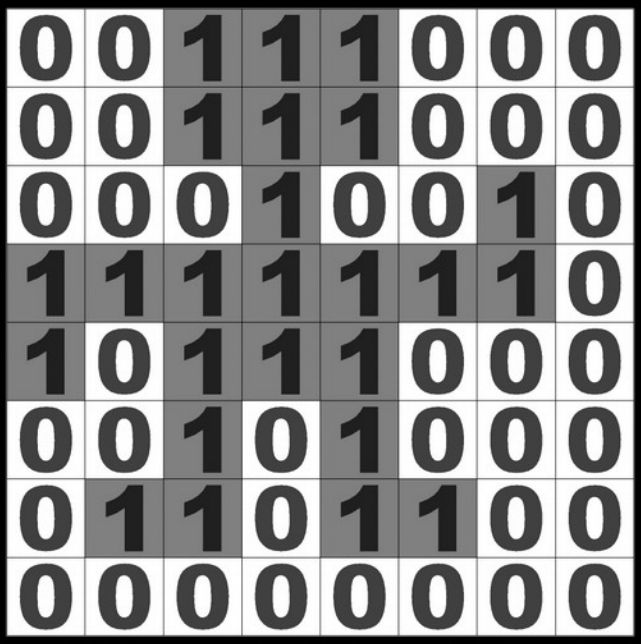
- Any hexadecimal editor. In our case we will use "wxHexEditor", but you can use any other for Linux, Windows or Mac. It does not matter.




The structure of the data is as follows: The first byte (2 digits in hexadecimal) corresponds to the bit information, zeros and ones, of the first dotted line of the chart, the second byte to the second, the third byte to the third, and so on. After the eighth byte will begin the first byte of the second character of the table, then the second byte, third and following bytes, until completing the 64 characters of 8 bytes each of the entire table.

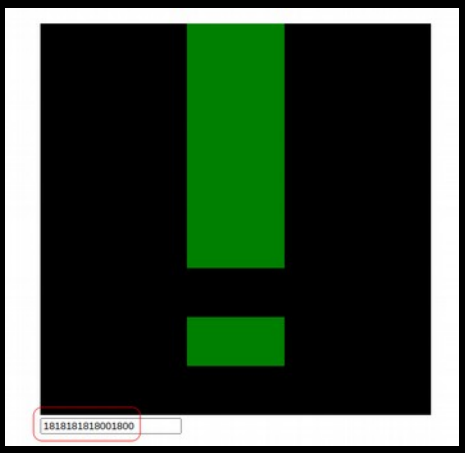



Bit a 0 indicates that this pixel on the screen should take the background color of the game and bit a 1 indicates that this pixel should be presented on the screen with the color assigned by the programmer for the whole character to which it belongs, i.e. all the 'ones' of its 8 bytes, as shown in the following image...




Let's start with the character "$" which we will convert into the character "!". To do this, we open the file "*Charset_Editor.html*".

Using the mouse we can 'turn on' and 'turn off' the pixels we need to form the exclamation image. Notice that the bytes shown in hexadecimal, at the bottom, are changing in real time. All the bytes, when you have the finished image will have the value '00' or '18', forming a hip of 16 digits.

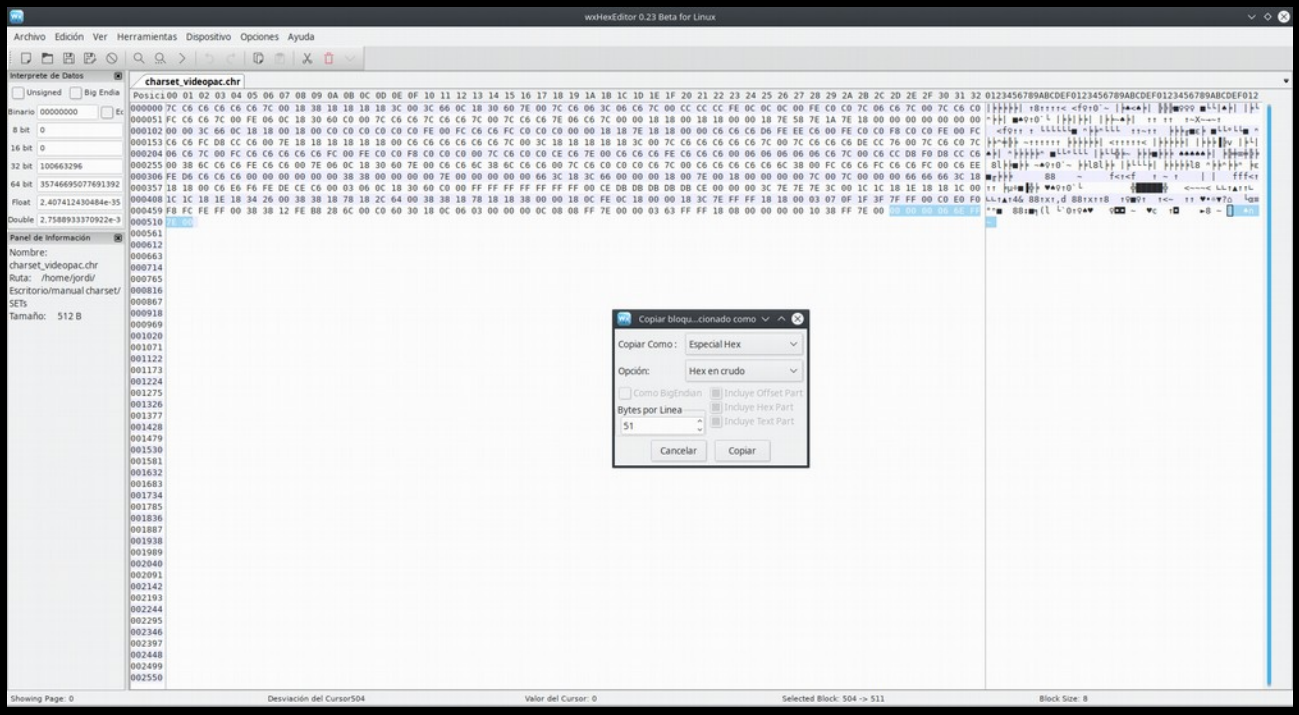



The editor also works the other way around. That is, you can type those hexadecimal values in that text field and pressing 'enter' will update the dot matrix. This is especially useful if you want to search for a particular character, from the wxHexEditor, so that you can edit its 8 bytes precisely.

To do so, simply follow these steps:

Select with the mouse any group of 8 bytes, for example the last 8 bytes of the file.

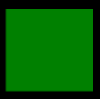
You do not need to count the bytes you have selected. In the Status Bar, the "*Block Size:*" section will always tell you how many bytes you have shaded. Now from the context menu choose copy as... and in the dialog choose "*Special Hex*" and "*Raw Hex*".




Now paste the contents of the clipboard into the hexadecimal text box of the html editor and when you press 'enter' this image will appear...

As you may have guessed, to place your new exclamation point graphic in the correct place in the table, which corresponds to the "$" sign, you have to count memory locations in the file. Knowing that each character is 8 bytes and that you have the dollar in position 12 of the table, we have to paste the contents of the clipboard (which you must have previously 'copy' the selection of the bytes in the html editor. See the image below) by positioning the text cursor, with the mouse, on the first character of byte number 88. The wxHexEditor program will indicate, at the bottom, at all times, where you are positioned, in the text 'Cursor offset'.

Once the content has been pasted the appearance is as you can see in the following page...



charset_videopac.chr

```
Posici 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32
000000 7C C6 C6 C6 C6 7C 00 18 38 18 18 18 18 3C 00 3C 66 0C 18 30 60 7E 00 7C C6 06 3C 06 C6 7C 00 CC CC CC FE 0C 0C 0C 00 FE C0 C0 7C 06 C6 7C 00 7C C6 C0
000051 FC C6 C6 7C 00 FE 06 0C 18 30 60 C0 00 7C C6 C6 7C C6 C6 7C 00 7C C6 C6 7E 06 C6 7C 00 00 18 18 00 18 18 00 00 18 18 18 18 18 00 18 00 00 00 00 00 00 00
000102 00 00 3C 66 0C 18 18 00 18 00 C0 C0 C0 C0 C0 C0 FE 00 FC C6 C6 FC C0 C0 C0 00 00 18 18 7E 18 18 00 00 C6 C6 C6 D6 FE EE C6 00 FE C0 C0 F8 C0 C0 FE 00 FC
000153 C6 C6 FC D8 CC C6 00 7E 18 18 18 18 18 00 C6 C6 C6 C6 C6 7C 00 3C 18 18 18 18 3C 00 7C C6 C6 C6 C6 7C 00 7C C6 C6 C6 DE CC 76 00 7C C6 C0 7C
000204 06 C6 7C 00 FC C6 C6 C6 C6 C6 FC 00 FE C0 C0 F8 C0 C0 C0 00 7C C6 C0 C0 CE C6 7E 00 C6 C6 C6 FE C6 C6 C6 00 06 06 06 06 06 C6 7C 00 C6 CC D8 F0 D8 CC C6
000255 00 38 6C C6 C6 FE C6 C6 00 7E 06 0C 18 30 60 7E 00 C6 C6 6C 38 6C C6 C6 00 7C C6 C0 C0 C0 C6 7C 00 C6 C6 C6 C6 C6 6C 38 00 FC C6 C6 FC C6 C6 FC 00 C6 EE
000306 FE D6 C6 C6 C6 00 00 00 00 00 00 38 38 00 00 00 00 7E 00 00 00 00 00 00 66 3C 18 3C 66 00 00 00 06 0C 18 30 60 00 00 00 00 00 7C 00 7C 00 00 00 66 66 66 3C 18
000357 18 18 00 C6 E6 F6 FE DE CE C6 00 03 06 0C 18 30 60 C0 00 FF FF FF FF FF FF FF 00 CE DB DB DB DB DB CE 00 00 00 3C 7E 7E 7E 3C 00 1C 1C 18 1E 18 18 1C 00
000408 1C 1C 18 1E 18 34 26 00 38 38 18 78 18 2C 64 00 38 38 18 78 18 18 38 00 00 18 0C FE 0C 18 00 00 18 3C 7E FF FF 18 18 00 03 07 0F 1F 3F 7F FF 00 C0 E0 F0
000459 F8 FC FE FF 00 38 38 12 FE B8 28 6C 00 C0 60 30 18 0C 06 03 00 00 00 0C 08 08 FF 7E 00 00 03 63 FF FF 18 08 00 00 00 00 10 38 FF 7E 00 00 00 00 06 6E FF
000510 7E 00
000561
```

You must repeat the same operation with each character you want to change, until you are left with a data set like this...

charset_videopac.chr

```
Posici 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32
000000 7C C6 C6 C6 C6 7C 00 18 38 18 18 18 18 3C 00 3C 66 0C 18 30 60 7E 00 7C C6 06 3C 06 C6 7C 00 CC CC CC FE 0C 0C 0C 00 FE C0 C0 7C 06 C6 7C 00 7C C6 C0
000051 FC C6 C6 7C 00 FE 06 0C 18 30 60 C0 00 7C C6 C6 7C C6 C6 7C 00 7C C6 C6 7E 06 C6 7C 00 00 18 18 00 18 18 00 00 18 18 18 18 18 00 18 00 00 00 00 00 00 00
000102 00 00 3C 66 0C 18 18 00 18 00 C0 C0 C0 C0 C0 C0 FE 00 FC C6 C6 FC C0 C0 C0 00 00 18 18 7E 18 18 00 00 C6 C6 C6 D6 FE EE C6 00 FE C0 C0 F8 C0 C0 FE 00 FC
000153 C6 C6 FC D8 CC C6 00 7E 18 18 18 18 18 00 C6 C6 C6 C6 C6 7C 00 3C 18 18 18 18 3C 00 7C 00 E6 F6 DE CE C6 00 7C C6 C6 C6 DE CC 76 00 7C C6 C0 7C
000204 06 C6 7C 00 FC C6 C6 C6 C6 C6 FC 00 FE C0 C0 F8 C0 C0 C0 00 7C C6 C0 C0 CE C6 7E 00 C6 C6 C6 FE C6 C6 C6 00 06 06 06 06 06 C6 7C 00 C6 CC D8 F0 D8 CC C6
000255 00 38 6C C6 C6 FE C6 C6 00 7E 06 0C 18 30 60 7E 00 C6 C6 6C 38 6C C6 C6 00 7C C6 C0 C0 C0 C6 7C 00 C6 C6 C6 C6 C6 6C 38 00 FC C6 C6 FC C6 C6 FC 00 C6 EE
000306 FE D6 C6 C6 C6 00 00 00 00 00 00 38 38 00 00 00 00 7E 00 00 00 00 00 00 66 3C 18 3C 66 00 00 00 06 0C 18 30 60 00 00 00 00 00 7C 00 7C 00 00 00 66 66 66 3C 18
000357 18 18 00 C6 E6 F6 FE DE CE C6 00 63 66 0C 18 30 66 C6 00 FF FF FF FF FF FF FF 00 92 54 38 EE 38 54 92 00 00 00 3C 7E 7E 7E 3C 00 1C 1C 18 1E 18 18 1C 00
000408 1C 1C 18 1E 18 34 26 00 38 38 18 78 18 2C 64 00 38 38 18 78 18 18 38 00 00 18 0C FE 0C 18 00 00 18 3C 7E FF FF 18 18 00 03 07 0F 1F 3F 7F FF 00 C0 E0 F0
000459 F8 FC FE FF 00 38 38 12 FE B8 28 6C 00 00 00 00 00 00 08 08 10 00 00 00 0C 08 08 FF 7E 00 00 03 63 FF FF 18 08 00 00 00 00 10 38 FF 7E 00 00 00 00 06 6E FF
000510 7E 00
000561
```

Now just save it with the name "my_charset.chr" on the microSD card of your FPGA and it will be ready to use!

Let's see the result with this case study, the classic game Videopac Nº22, -Space Monster- which represents a good example.

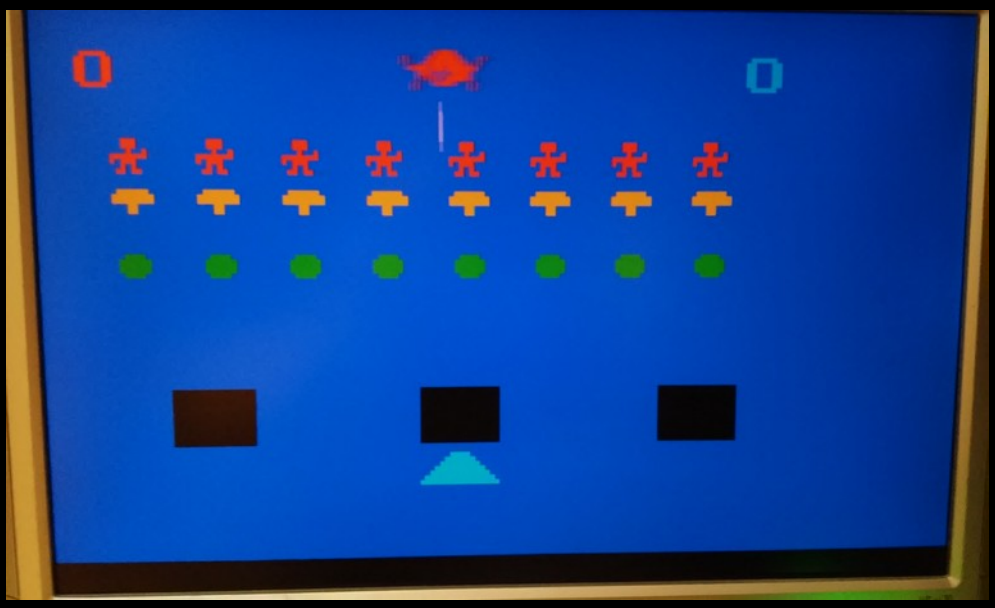
You need 2 things:

- The ROM of the game. Maybe you can find it as "vp_22.bin" (2 KB)

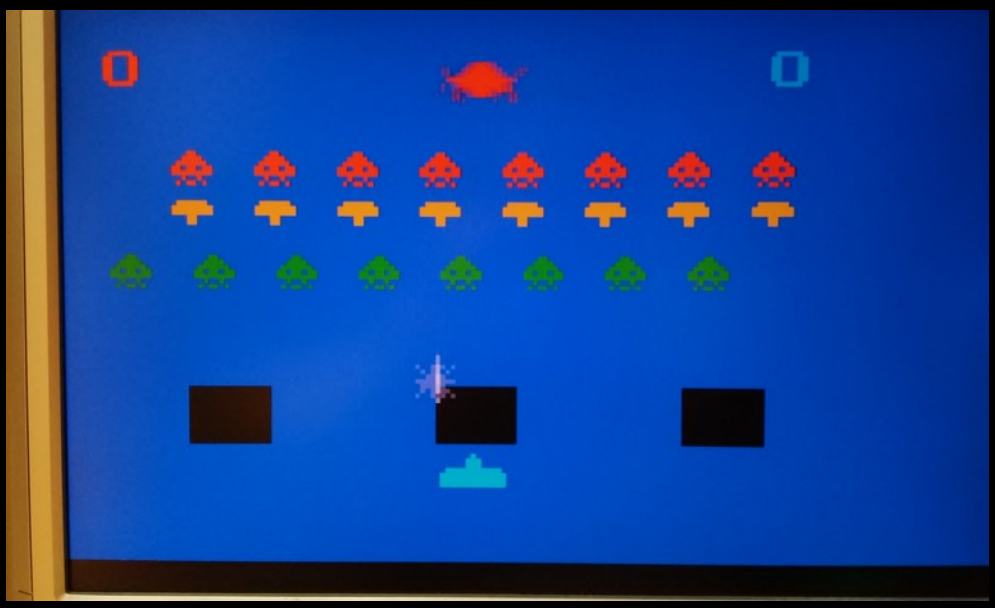
- The file "me_marcianitos.chr" in (for example) the CHR folder, which you will have created inside the Videopac game folder.

You can load the 2 files in the order you prefer. But if you load the game first to get to know it and then load the .chr file you will be able to see the changes that occur.

First you will see it like this...



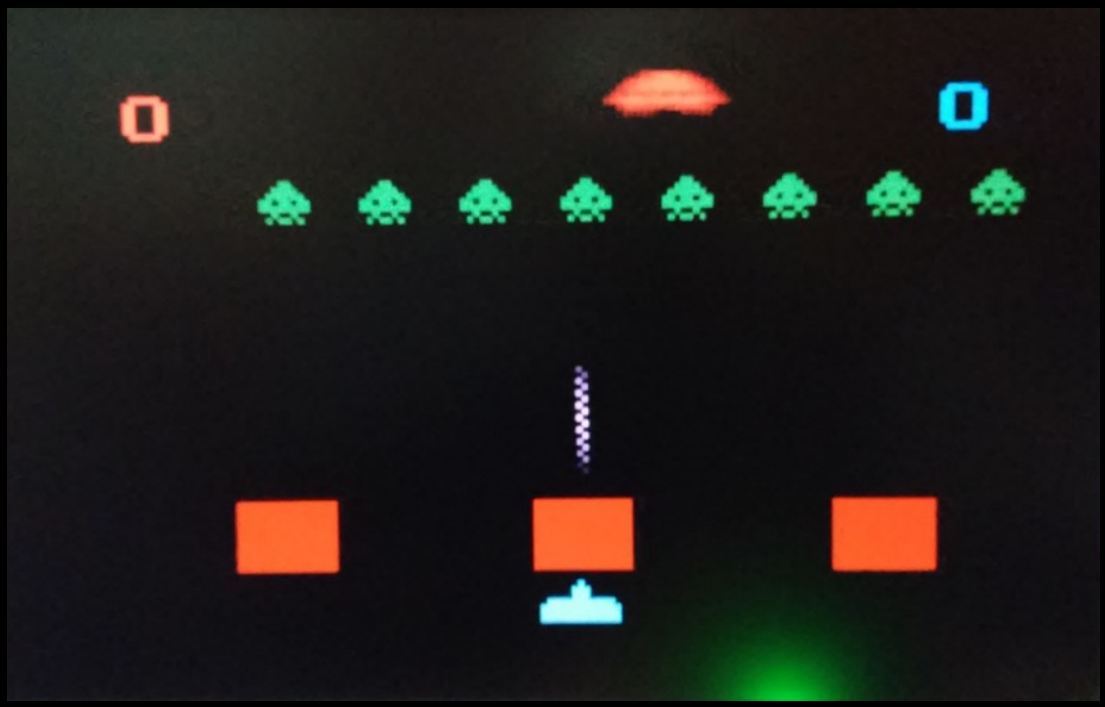
And then like this...



You may notice that the enemies have changed to terrifying Martians and that the pyramid you had as a missile launcher is now a missile launcher, like the real ones ;)

## We go one step further:

If you now load the binary 'me_marcianitos.bin' you will see that something else has changed.



Now the enemy missile launchers and their ball-barriers have disappeared. You can also see that you no longer have the space monster (instead it's a UFO ship) and that the missiles also look different. That happens because the charset does not include (logically) the sprites that each game may or may not have. That must be changed directly by editing the binary of the game itself from the wxHexEditor. It is not a simple task to find the position in the memory for each of the graphics.

The mechanics of the game is simple: The first shot that hits the Martian breaks his protective barrier and at the same time he can no longer use his laser weapon. At the second shot, the Martian, who was defenseless, is eliminated.

There is, in this first version, a known glitch: The UFO spacecraft deforms when shooting at us.

I hope you find this manual useful and that you will be encouraged to create your own charsets for Videopac!