Laser 200 clone in a FPGA

Jesús Arias (2025)

1 Introduction

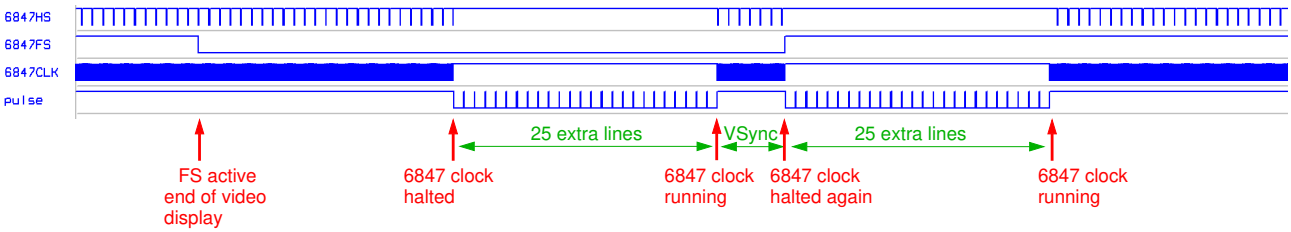
The Laser-200 computer, from Video Technology, also probably better known as the VZ200 outside Europe, was an early 8-bit computer where, incidentally, I typed my first “20 GOTO 10” line during a visit to a computer showroom where the big star was a talking ZX-Spectrum with an speech synthesis card attached (Currah Microspeech). That little computer lacked the glamour of its competitors and I haven’t seen any other of those since. I rediscovered it recently in a Youtube video and was very pleased to be able to dig into its internal details. In particular, its small amount of RAM made me think instantly about an FPGA recreation. And quoting Adrian Black “...without further ado, lets go right into it.” :)

2 Laser 200 details

That computer was a simple one, even simpler than the Spectrum, and its performance in terms of memory and video resolution placed it in between the ZX81 and the Spectrum. And, while Sinclair’s resorted to ULAs for video display, VTech chose the Motorola’s 6847 as the video controller, thus simplifying a lot the electronics inside the case.

Well... Not really so much due to an stupid detail: The 6847 only allowed the generation of NTSC video, but the VTech market demanded a PAL version. Unfortunately Motorola seems to have ignored the rest of the World outside USA as usual at that time, before, or since. So, what they could do? A clever hardware patch with a few TTL counters and gates: When the 6847 ends its video frame its clock is halted and the counters keep on generating 50 extra lines of video before releasing the 6847 clock again, so, after two interlaced frames the total line count is 625 while for the 6847 only 525 lines had passed. Well, the PAL patching logic is a bit

more complex, with 25 lines inserted before the vertical retrace and another 25 lines after the retrace, as the simulation of its schematic shows:



Here, the 6847 sets its frame-sync (FS) signal low just after the last visible pixel of the frame. The patching logic allows the video clock to run for a few more lines before halting it, and then the horizontal sync pulse is generated by this logic instead of the 6847. The video clock is released after 25 lines and the 6847 resumes its counting and generates the vertical sync waveform, raising FS after that. This halts the video clock again and another 25 lines are generated by the patching logic before the 6847 can continue displaying the new frame.

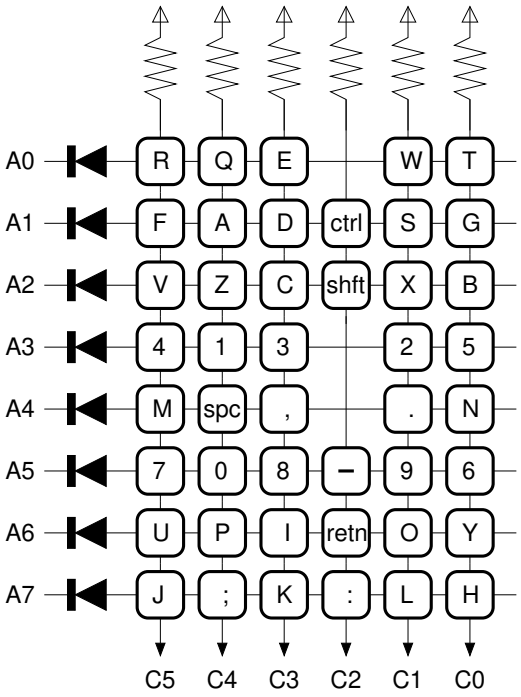
Apart from this video trick, the rest of the computer schematic is pretty straightforward. It includes:

- A Z80 running at 3.57MHz. Again, this is the frequency of the NTSC color subcarrier, that is also required as the time base for the 6847 but has nothing to do with PAL color in that computer (notice the 6847 shifts pixels out on both edges of the clock, so, its effective clock frequency is doubled).
- 16KB of ROM with BASIC mapped at address 0x0000.
- 2KB of static RAM, that can be further expanded by adding another 4KB of static RAM on some model variants, or with 16KB of RAM in an external expansion module. This RAM starts at address 0x7800.
- 2KB of video RAM. This memory is shared between the CPU and the video controller, with the CPU having more priority. That means we should only access the video memory during retraces or we will end up with some “interference” on the screen. The video RAM is mapped at address 0x7000.
- A 6-bit output register mapped at memory addresses 0x6800 to 0x6FFF. Its bits are:

bit #	Function
0	Speaker output (+)
1	Cassette output (LSB)
2	Cassette output (MSB)
3	6847’s A/G (0: Text mode, 1: Graphics mode)
4	6847’s CSS: alternate color palette if 1
5	Speaker output (-)

- An 8-bit input register, also mapped to memory address 0x6800 to 0x6FFF. Its bits are:

bit #	Function
0 to 5	Keyboard column (C0 to C5, see matrix)
6	Cassette input
7	Frame sync. (vertical retrace)



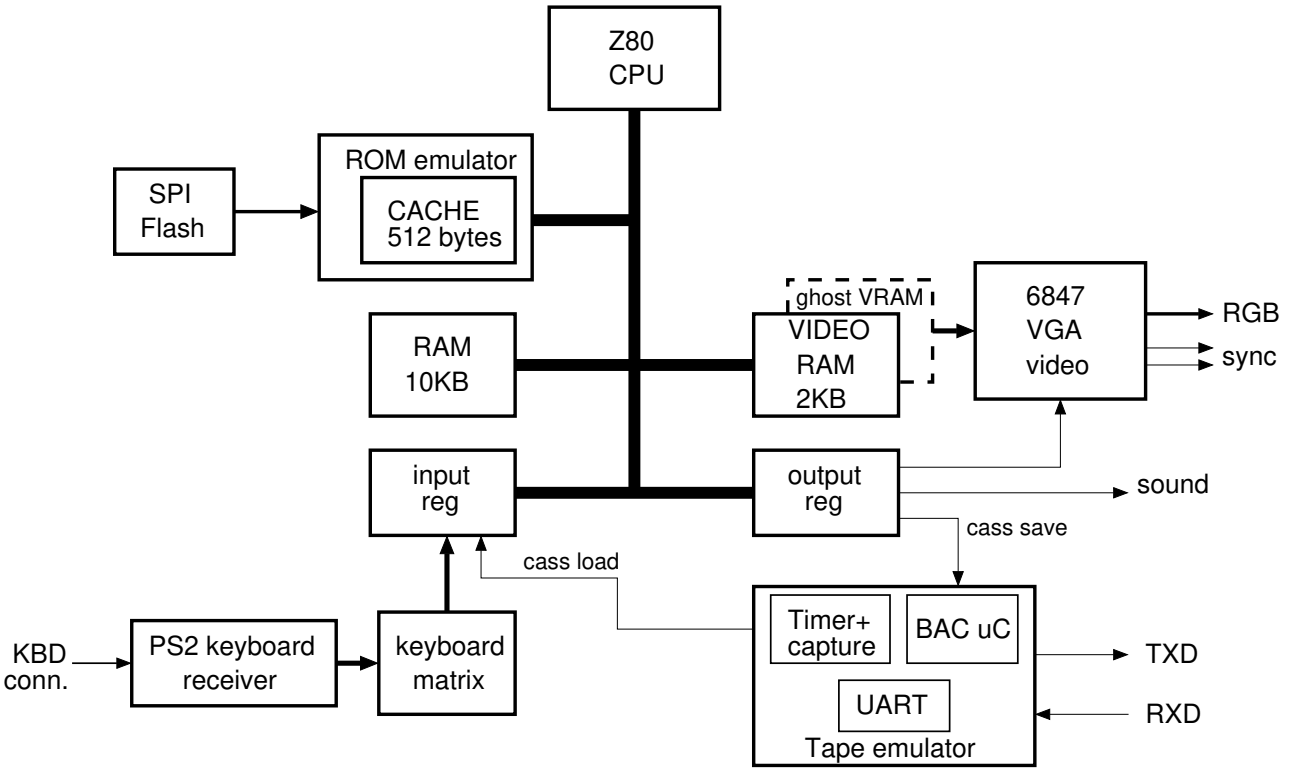
- The Frame sync. signal of the 6847 is also tied to the interrupt input of the Z80.

And that's all folks! But I think these control signals deserve some further comments. The speaker is a piezoelectric one and is driven from two bits of the output register. These bits (#0, and #5) must have opposite values and must change state simultaneously in order to create some sound. The speaker can actually have three different voltages at its terminals: +5V, -5V, or 0V (if the two sound bits are 00 or 11).

The cassette output has a 3-level DAC attached to bits #1 and #2, but in fact only the combinations 00 and 11 are used by the firmware, so, only bit #2 is really needed for tape recording (a newer model, the Laser-300, only has one bit (#2) for cassette output).

And about the keyboard, it is an 8x6 matrix with 45 keys actually implemented. The columns have pull-ups and are connected to the bits #0 to #5 of the input register, while the rows are driven by the lower 8 bits of the Z80 address bus through series diodes, the same Sinclair did in its Spectrum (but with the higher address bits). So, in order to select a particular row, its corresponding address bit has to be zero while the 7 remaining bits must be one (addresses: 0x68FE: row #0 to 0x687F: row #7)

3 FPGA replica



The block diagram of the Laser 200 replica is presented in the above figure. Of course, It differs from the original in several aspects:

- The screen here is a VGA monitor with 640x480 resolution and 60Hz refresh. This means a completely different timing and an RGB color space instead of YPbPr.
- The keyboard is of the PS2 type, not a plain key matrix.
- The tape interface includes another computer that translates the tape waveforms into ".CAS" files dumped over a serial port.

Lets describe these blocks in more detail:

3.1 Clocks

The main clock input is obtained from a PLL and runs at 25MHz, or to a close frequency (25.125MHz in the Alhambra board). This is the clock for the cached SPI ROM controller, but for the rest of the system this frequency is further divided by the following prescalers:

- 1/2, resulting in 12.5MHz. This is the pixel clock for the video controller, and results in 320 visible pixels for each video line instead of the 640 pixels we would get with the usual 25MHz clock.
- 1/7, resulting in 3.57MHz, almost the exact NTSC color subcarrier frequency (315/88 MHz) used as the main clock in the original computer. This is the clock for the Z80, the BAC microcontroller, and most of the system logic, but we should remark that the Z80 clock, "cclk", is a "gated" copy of this frecuency and it can be halted for some long time intervals.

3.2 Memory

The intended FPGA board for this design is the Alhambra-II, that includes an ICE40HX4K FPGA and little more. This FPGA has 32 BRAMs totaling 16KB. This is enough for the Laser 200 RAM, but we also have to

store the ROM of the computer somewhere, and the only other storage device in the board is the same SPI flash used for the FPGA configuration. The problem with this flash is its serial interface that results in long access times. This problem was alleviated by including an small cache memory for the ROM data with the following characteristics:

- 512 bytes of RAM (1 BRAM), divided into 8 cache lines of 64 bytes each.
- The contents of the cache lines are read without delays if the MSBs of the address bus matches the value stored in a related "tag" register. If neither of the 8 tag registers matches the address the least recently used line is invalidated and its contents are read from the SPI flash. This takes 544 cycles at 25MHz or 22.7μs. During this time the Z80 clock, "cclk", is forced high and code execution is stopped.

Leaving the ROM outside the FPGA the internal BRAM blocks have the following usage:

Size	number BRAMs	Use
2KB	4	Video RAM
2KB	4	Ghost Video RAM
512 bytes	1	ROM cache
1KB	2	BAC microcontroller
10KB	20	RAM
Total	31	

Here 11 BRAMs are used in the computer subsystems. The Laser 200 firmware can detect the actual size of the available RAM and this allowed us to assign almost all the remaining BRAMs to the main RAM of the recreated computer instead of the 2KB or 6KB of the real counterparts. It is a little tricky to check the free RAM from BASIC, but it can be done in the following way:

1. Type "10 PRINTPRINT(0)". Yes, that's correct
2. Type "POKE 31470,218"
3. Type "RUN". The number of free bytes for BASIC are finally displayed:

```
10 PRINTPRINT(0)
POKE 31470,218
RUN
 9418
READY
```

3.3 Video controller

The designed video controller mimics the two 6847 modes of the Laser computer, that are:

- Text mode with 32×16 characters that can display up to 64 different characters including uppercase letters, digits and symbols. Characters are displayed with inverse video if their bit #6 is set, and, if bit #7 is set the character space is filled with up to 16 different "semigraphics" characters that can also display 8 different colors. Each character is displayed into a 8×12 pixel cell, but in the internal character generator ROM it has 5×7 pixels. In the design this ROM is asynchronous and requires about 240 logic cells but no BRAMs. For characters the text color is green and the background dark green, or orange and dark orange depending on the CSS signal from the output register, while for semigraphics the background color is always black (this is the only case when black is displayed by the real 6847)

Character text:

bit #7	bit #6	bits #5 to #0
0	Inverted video if 1	Character code

Character Set (codes 0 to 63):

Semigraphics:

bit #7	bits #6 to #4	bit #3	bit #2	bit #1	bit #0
1	ON color	B3	B2	B1	B0

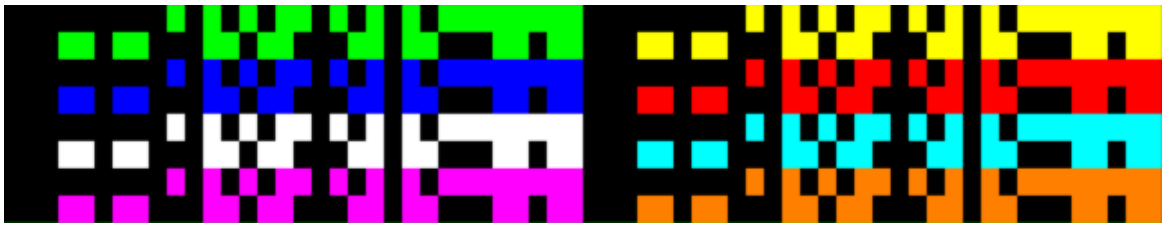
Block positions:
(inside character cell)

B3	B2
B1	B0

ON colors:
(OFF is black)

bits #6 to #4	Color
0	Green
1	Yellow
2	Blue
3	Red
4	Buff (white)
5	Cyan
6	Magenta
7	Orange

The 128 semigraphics codes:



- Graphics mode with 128×64 pixels and 2 bits per pixel. Here each graphic pixel is actually a 2×3 pixel on the screen, so, the screen resolution is the same as in text mode, or 256×192 pixels. 4 colors can be displayed simultaneously on the screen, but these 4 colors also depends on the level of the CSS signal in the output register (colors 0 to 3 if CSS is low, or 4 to 7 if CSS is high).

bits:	7,6	5,4	3,2	1,0
Pixel (3: left ... 0: right)	P3	P2	P1	P0

The VGA resolution is much more than really needed, and therefore, it was halved, both horizontally and vertically. That was accomplished by means of a half frequency pixel clock (12.5MHz) and by repeating the lines two times, resulting in a 320×240 resolution, more close to the desired 256×192 screen. The excess horizontal pixels and vertical lines were placed on the screen borders.

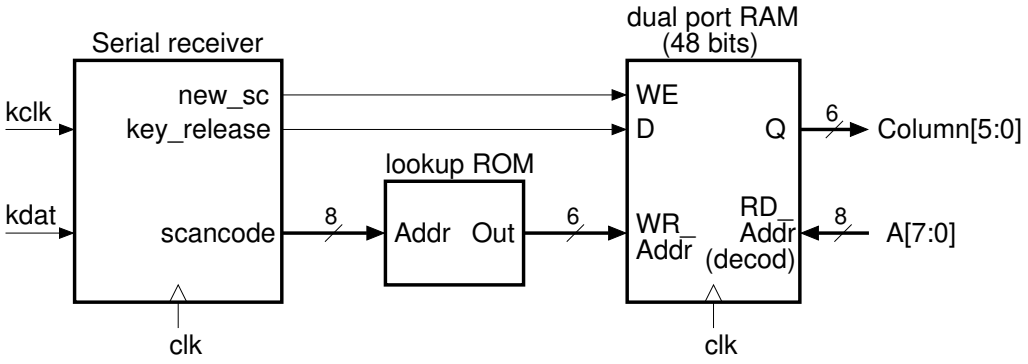
The video generator reads its data, either characters or pixels, from a “ghost” RAM memory. This RAM is written by the Z80 in parallel with another regular video RAM, but it is read only by the video controller.

This is easy to do thanks to the dual-port feature of the BRAMs. In this way there are no contentions between the Z80 and the video controller when accessing the video RAM and no "interference" is ever displayed on the screen. When the Z80 reads the video RAM the data comes from the regular RAM instead of the "ghost" one. Both memories could have a different content on startup, but after a "CLS" command their data is going to be exactly the same.

And finally, the designed controller outputs a digital IRGB video signal instead of an analog YPbPr. This resulted in some color differences with respect to the original 6847:

- Dark green and dark orange are simply black. This is simpler, but IMO, also very desirable because it results in a better contrast for text.
- The orange color (100% red, 50% green) can't be located in the IRGB palette, so, a pink color (IRGB=1100) is used instead.

3.4 Keyboard interface



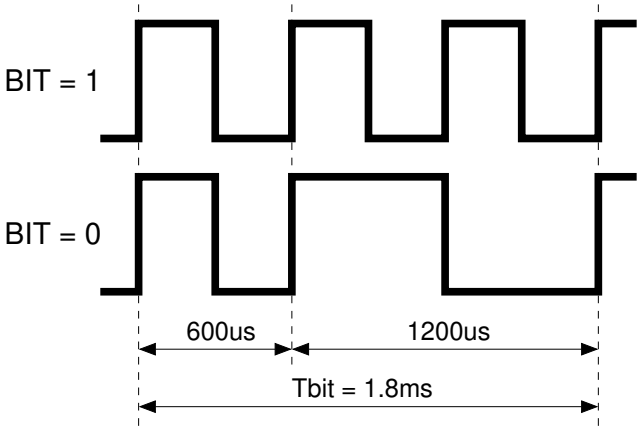
The keyboard has a PS2 interface that is just a synchronous serial port where the scancodes of the pressed keys are received. Released keys sends the same scancode but preceded with an 0xF0 byte. This information is used to generate a key release signal while the key position in the matrix is obtained by means of a lookup table (asynchronous ROM). Then follows a 48-bit RAM where a single bit is written with the key release value when an scancode is received. The contents of this RAM are presented to the Laser's input register as 6-bit words depending on the 8 lower bits of the Z80 address bus. Well, in fact the data presented is the logical AND of the rows actually selected, as it should be in the real keyboard matrix.

So, at the end each key in the original computer has an equivalent one in the PS2 keyboard, but not all keys had a clear substitute and, in particular, the keys ";" and ":" are actually mapped to "Alt-Gr" and "Windows" on my Spanish keyboard.

This PS2 peripheral is almost the same as that of the ZX-Spectrum clone, but it incorporates a further refinement: a timeout counter that monitorizes the keyboard clock and resets the receiver shift register if there is no activity for some time. This avoids being locked into receiving wrong scancodes if a partial data is received on startup, a problem that shows up sometimes.

3.5 Cassette emulation

First, the format of the cassette data had to be studied, and the technical reference manual gives very little clues about it appart from an useless peak-to-peak voltage. But, after looking at some captured waveforms, dumps of .VZ and .CAS files, and to the source code of the z88dk-appmake utility, I was able to figure it out. Lets present first the way single bits are modulated into the cassette lines, and remember, the cassette output has a 3-level DAC but only the maximum and minimum levels are actually used:



As we can see in the previous figure, a one is coded as a train of 3 square-wave cycles with a 1660Hz frequency, while a zero comprises a single 1660Hz cycle followed by another 830Hz cycle. The time for both bits is the same, around 1.8ms, resulting in a bitrate of 550 bits/s.

Then follows the structure of a whole tape dump, where the data is shifted serially, MSB first, and the contents of the tape are:

Field	Size (bytes)	value	Comments
Sync	128	0x80	
Preamble	5	0xFE	
Type	1	0xF0: BASIC, 0xF1: binary	
Filename	<17	String	variable length
Filename terminator	1	0x00	
Gap	0.5	nothing (silence)	Just a delay
Start Address	2	xxxx	Little endian, 16-bit
End Address	2	xxxx	Little endian, 16-bit
Data	n	xx ... xx	variable length
Checksum	2	xxxx	Little endian, 16-bit
Trailing	19	0x00	Ignored

Here I want to remark the importance of the gap between the filename and the data. Without it the computer fails to read the tape. And the checksum value that is just the addition of all the 8-bit data after the gap, addresses included, truncated to 16 bit, and added at the end of the data block without further inversions. After the checksum comes a trailing block of plain zeroes, I don't known why.

This is precisely the contents of a ".CAS" file (except for the gap, that isn't included), but the most common tape format for the emulators of these computers is the ".VZ" file, that lacks the Sync and Preamble fields, but more importantly, also the End Address and Checksum fields. This is the reason why the .VZ format isn't directly supported in the built-in tape emulator: We must emit the End Address value early, but we don't know it until the whole tape file is uploaded and its bytes counted, and by then it is too late. Thus, only the .CAS format is supported, so, tape images will have to be converted from .VZ to .CAS format in the PC before sending them to the tape emulator through the serial port of the FPGA board.

The tape emulator is built around a BAC-3 microcontroller with 256 words of program memory and 512 bytes of data memory, along with a few peripherals:

- Simple output ports for cassette output and state debug via LEDs.
- An UART with a fixed 115200 baud rate for the PC interface.

- A timer counter that runs with the Z80 clock signal, and therefore is halted along the CPU every time the ROM cache have to read a line from the SPI Flash. This is important because these idle times during flash reads can distort the cassette waveforms enough to make the images unreadable. But, if time is measured in CPU cycles instead of absolute time, the waveforms remain perfect.

- A capture register where the value of the timer counter is stored every time a rissing edge is detected in the cassette waveform generated by the Z80. This event also sets a flag that signals it, while a read of the capture register resets this flag.

The BAC's firmware deals with both the TX (Load) and RX (Save) transactions between the Laser's tape and the serial port. It also switches between TX and RX without any user input. Lets describe these tasks with a little more detail:

- TX mode is entered as soon as a byte from a .CAS file arrives at the UART receiver. A Z80delay routine waits the desired number of Z80 cycles before returning, but it also keeps reading the incoming serial data, storing it into a 256 byte FIFO, and signaling the PC to stop sending data via an XOFF character if the FIFO is more than half full. The main TX code calls the Z80delay routine until the FIFO is half-full and then starts reading the FIFO and generating the cassette waveform. When the FIFO is under 1/4 full an XON character is sent to the PC, resuming the sending of more data, and the loop keeps running until the FIFO is completely empty. Also, the first 0x00 byte is followed by a gap, but this is done only once. The remaining 0x00 bytes will not generate any gap.

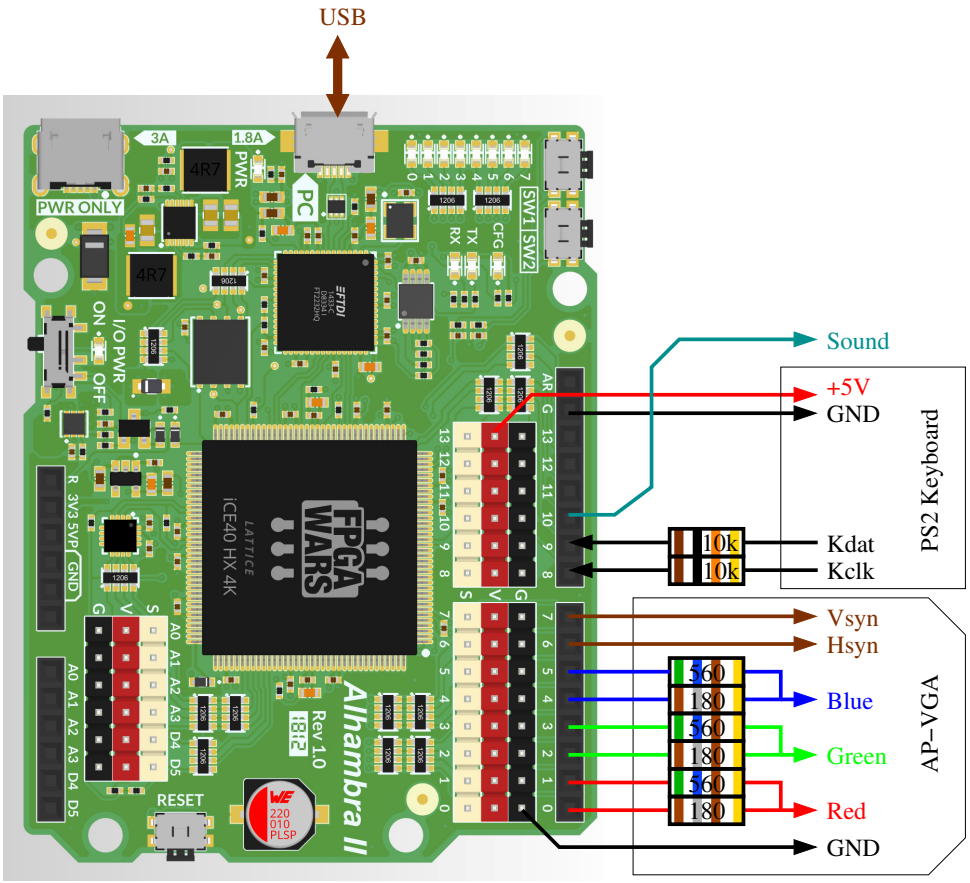
- RX mode is entered if a rising edge in the cassette line is detected thanks to the capture register. The captured time stamp is stored into a variable and subtracted from the time of the next capture event. A long time difference means a 0-bit waveform is being received, while three consecutive short times is an one. These bits are feed into a software 8-bit shift register until a value of 0x80 is detected, and then the bytes are shifted just 8 times and dumped over the serial port. In this case the data rate is very slow when compared to the PC and no flow control is required at all. The received bytes, when stored into a file in the PC, constitute a .CAS file that requires no further processing.

3.6 Sound

The sound quality of the original computer is surely very poor due to the crappy piezoelectric speaker included, but anyway, its three level output could be exploited by an hypothetical software in order to play two notes simultaneously. This has been considered in the replica, but using a single PWM output instead. Here, the output is low (sound bits 10), high (sound bits 01), or a 111kHz square wave (sound bits 00 or 11).

4 Hardware connections

Let me present how to connect a VGA monitor, and a PS2 keyboard, to an Alhambra-II board:



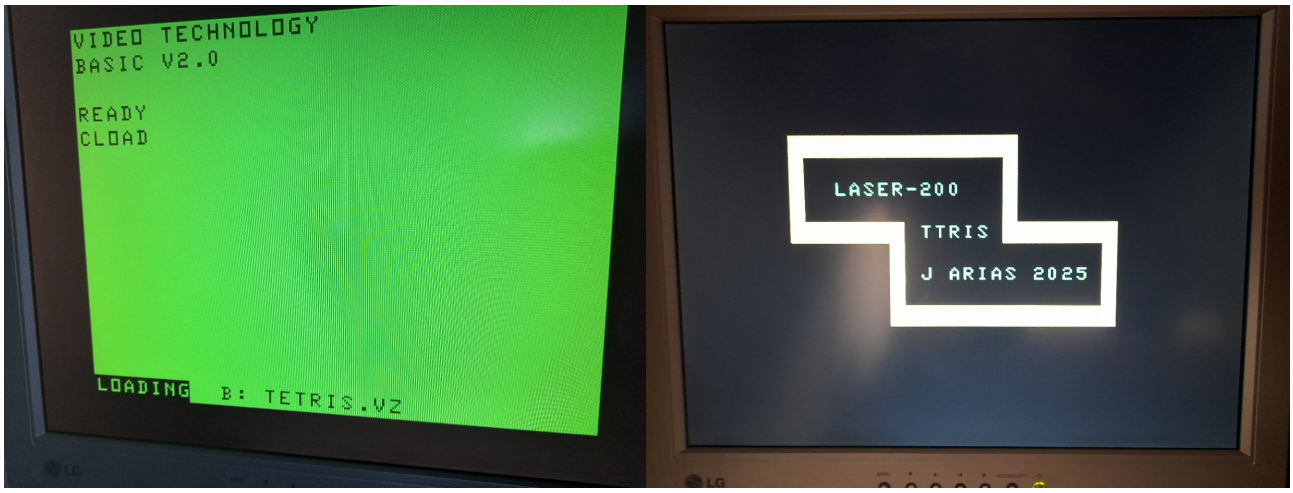
The VGA interface uses the same resistor DAC as the Eladio's AP-VGA adaptor, but with lower resistor values. The AP-VGA seems to have been designed for Arduino with 5 volt logic levels and when connected to the Alhambra board the resulting image is a bit dim.

And another problem is the PS2 keyboard, that also generates 5 volt signals. But in this case a simple series resistor can limit the current flowing to the FPGA pin into a harmless value, and the use of more sophisticated level shifters is simply overkill.

And, finally, a single bit PWM output is provided for sound. This is a digital signal and some attenuation and DC blocking could be needed if connected to an audio amplifier. I Think a 100μF capacitor in series with a 100Ω resistor and a speaker can still generate sounds loud enough for its use without any other amplification.

5 Screenshots

An Alhambra-II board attached to an VGA monitor and a PS2 keyboard (only the connector is visible in the snapshot) via breadboard (not really an example for mass production ;) and a BASIC program with graphics.



And now the loading of a tape file into the computer along with the terminal application in the PC side transferring the file with XON/XOFF flow control.

6 Conclusions

I was a bit worried about the impact of cache misses on the Laser's software but results were good, with no noticeable slowdowns, and even the crude music that computer is able to generate seems to be still in tune. The only time-critical routines of the ROM seems to be the cassette related ones, and this was already solved in the tape emulator computer by means of counting the actual clock cycles of the Z80 instead of absolute time. Well, that was a relief, but there are still some potential issues regarding other software, mainly games, like:

- The frame-sync signal in the 6847 starts just after the last visible pixel and lasts until the vertical retrace, while in the recreated computer it starts after the last visible line and lasts until the first visible line in the next frame, this is about 4.5ms low versus the original 4.1ms low. This is an small difference and the interrupt routine does its processing without any problem.

- The frequency of the video interrupt is 60Hz instead of 50Hz. This means the cursor is blinking a 20% faster than it should be. Maybe, if some of these computers were sold in the US this frequency is still correct for them ;)

- The tape emulator is unable to manage different encodings, so, no "turbo loaders" are going to work.

But, honestly, the games I found for this computer were very crude and far from the level of sophistication found in other machines like the Spectrum, so, it is quite improbable for them to get affected by these subtle details. Yet, we must keep these differences in mind.

Another interesting thing is the way the bits are coded in the cassette signal. I found no technical reason for these weird waveforms. A simple Manchester encoding would get 1660 bps instead of 550 bps while using the same bandwidth in its signal, and I don't think this encoding makes the tape more reliable. Also, I can't figure out why the cassette output was designed to have a 3-level DAC, a hardware feature unused by the ROM code. Maybe someone though about using the intermediate level for gaps and silence, thus avoiding some transients at the start of audio bursts, a feature that could be exploited in order to have shorter preambles. Well, at the end only two levels are actually used, preambles are long, and tapes are very slow. But, because the memory is small, the loading times aren't very long.