Designing the Synchro-Generator of an FPGA or CPLD Video Interface for Micro-Systems

Aleodor Daniel IOAN
"Gh. Asachi" Technical University of Iasi
Bld. Dimitrie Mangeron nr. 53 A, RO-700050 Iasi
aioan@ac.tuiasi.ro

***Abstract*—This paper is focused on the idea of designing and implementing an original video interface inside modern programmable digital circuits, like CPLDs and FPGAs. Such an interface provides direct connection to a TV or VGA display for microsystems with microprocessors or microcontrollers, which can be useful in various application fields. The first step, presented here from idea to optimal implementation, is the design of a TV compatible synchro-generator that is the heart of the video interface. Future research and development will be the RGB video memory scanner for this TV mode and the design of a new VGA compatible mode of the interface, to be included in the same FPGA or CPLD circuit. This work can also be considered a usage example of the new software environment "Altium Designer" at enhanced schematic design for programmable logic devices.**

***Index Terms*—CPLD, FPGA, horizontal and vertical counters, synchronization signals, micro-systems**

I. INTRODUCTION

The idea of designing this video interface came to me from development systems with microprocessors or microcontrollers and from old home computers. Such a development system is designed around one specific microprocessor or microcontroller family, with enough software and hardware resources to be universal, so it can be used in learning phase, in the development and debugging of software for dedicated systems, or for designing and testing some hardware external interfaces used in specific applications [1].

Generally, these kinds of systems have large external ROM and RAM memory to hold a monitor program in ROM which communicates by a serial link with a personal computer to load programs in RAM, to display resources and to receive commands, using only text modes. This communication is mandatory for code loading, but in testing phase it is quite incommode, because it limits learning possibilities and experimental applications to only command mode text display and the project cannot be tested in a real industrial environment without the presence of a PC.

My thought was: what if the development system also had a keyboard interface and a graphical display interface which connects it directly to a classic TV set or a PC monitor? An entire new field of learning possibilities, experimental applications and uses opens in this situation, all related to graphic display techniques. And further more, the system board will be no more forever dependant on the PC presence. The PC will be needed only for software download and even that could be eliminated if a powerful monitor program is used, with incorporated text editor, assembler and debugger, like old home computers had.

It should be noted that, even for the classic RS 232 serial link, some monitor programs with such capabilities already exist, like "Ultramon" for 8051 family of microcontrollers.

My second thought was at old computers, like the well known "ZX Spectrum" that had simple low resolution video interface for a TV-PAL display [2]. A modern implementation of such interface with programmable logic devices can be made in high integration scale, using only one FPGA or CPLD digital circuit for the entire design. This kind of implementation can be made universal enough to be compatible with most microprocessor and microcontroller systems, taking benefit of the programmable logic design versatility [3].

An actual medium sized FPGA circuit can hold even more than the logic of the interface: the video memory that holds the image information can also fit inside this chip if the resolution is not increased too much. And a larger FPGA can accommodate to hold the entire system, with microprocessor core, system memory and other hardware input-output interfaces.

This paper presents my original design solution of the hardware module which generates the synchronization signals for a TV-PAL mode display, often called the "synchro-generator" [2]. This part is practically the heart of the video interface, because it is the sequencer which defines the image resolution, format and size, controls the video memory scanning and the RGB serialization of image pixels.

The synchro-generator is also the most difficult module to design, because it works at very high speed and can be implemented only in hardware, and because all the waveforms of the signals that control the video interface are hard to imagine and to describe. For this reasons this kind of designs are very rarely approached.

On the other hand, this paper could also be considered original from the perspective of describing new design technologies in digital electronics, like the use of visual symbols at high abstraction level (which are already predefined in libraries) for FPGA or CPLD schematic design in some new software environment like "Altium Designer" [4].

II. DEFINING IMAGE PARAMETERS

The very first step in video interface design is to decide the resolution of the image which will be displayed, in pixels. Higher resolution means higher complexity and higher speed for video circuits and also means more processing power for the microprocessor or the

microcontroller in the system to use.

The interface described here is intended for small micro-systems, 8-bit like, which are most used in all kind of application fields. Such systems have limited processing power and cannot efficiently use high available resolutions.

Because of this, I decided to keep the "ZX Spectrum" original resolution of 256 x 192 pixels for graphic display and 32 x 24 characters for text display [2], which is enough for many applications. But because the "ZX Spectrum" had 4 colors only at character level (one character = 8 x 8 pixels) and this reduces the graphic abilities of the interface, I decided to raise the level of colors at 4 colors per each pixel. Anyway, this increase in color resolution will only affect the RGB scan generator and not the video sequencer described here. The old "ZX Spectrum" home computer also had a single color border margin around the active screen [2], to obtain a perfect rectangular image in the center, because the raster margins are not quite so linear on a TV display. This border was also good for some video effects, so I decided to keep it too.

The PAL TV standard [5] specifies a video image composed from 625 lines, with 25 Hz frame rate frequency, from which results a line frequency equal to 15625 Hz and a line time interval of 64 uS (microseconds), with 52 uS the visible part and 12 uS the horizontal retrace. Television images are normally divided into two interlaced fields of 625 / 2 = 312.5 lines each, one formed with odd lines and the other with even line lines, which are displayed in sequence, with a field frequency of 50 Hz. It is not usual for a computer generated picture to be interlaced, as this causes a lot of vertical jitter as the fields swap.

The low vertical resolution of the interface allowed me to use only one 312 lines field per frame, with a vertical synchronization frequency of 50 Hz. The lost half line allowed each field to be offset of the other when interlacing (it adds and forms one line needed to pass from the even to the odd field) and has no use in non-interlaced images.

Because the vertical retrace takes about 24 lines, only the remaining 288 lines from the total of 312 are really visible on the display. For a vertical resolution of 192 pixels, the visible border will last for 288 – 192 = 96 pixels, and can be divided in two symmetrical sections of 96 / 2 = 48 pixels each, one at the top and the other at the bottom of the screen.

For implementation optimization purposes, I decided later to add another 8 lines to the top border, which are normally invisible, so the total vertical height of the image becomes 296 = 56 + 192 + 48 pixels. To maintain image symmetry, a border of the same width should be used at the left and at the right of the screen, so the total width of the image must be 352 = 48 + 256 + 48 pixels.

The final result of my research is presented in Fig. 1. Image dimensions are given in pixels and in characters. The necessary time to display a pixel is called "pixel period" and is denoted by Tp. The character period Tc is 8 times greater than Tp, because a character is composed from 8 pixels per each line.

III. HORIZONTAL AND VERTICAL COUNTERS

The video interface works by scanning a video memory that holds the information at consecutive addresses. At the resolution required by this interface, the video memory can be organized in words of 8-bit width and must have double access path, one from the microprocessor of the system and the other from the video interface. This width also meets the architecture of the 8-bit micro-systems for which this interface is intended.

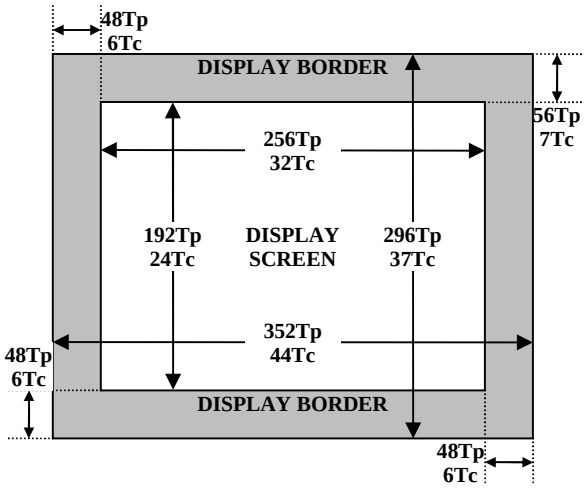

Figure 1. Dimensions of the display image in pixels and characters.

The video controller loads 8-bits (practically the first line of each character) simultaneously from the video memory and shifts it bit-by-bit to the RGB outputs. To generate the loading moments and to control the shifting, a 3-bit pixel counter driven by the pixel clock is required, which increments the pixel address within one character line from 0 to 7. This 3-bit counter also divides the pixel clock frequency by 8, to generate the character clock frequency.

The pixel clock period must be lower than the duration of a visible PAL TV line [5] divided by 352 (Tp < 52 uS / 352) to display at least all the image width mentioned above, including the border. Another condition for the pixel clock frequency is that it must be an exact multiple of the horizontal frequency (15625 Hz) to generate precise synchronization pulses. The smallest integer value that meets these requirements is 7 MHz (megahertz) for the pixel clock frequency. This value also generates square pixels on the screen at the considered resolution.

The 7 MHz pixel clock frequency is exactly 448 times the horizontal frequency, and this means that an entire video line will be exactly 448 pixels = 56 characters in length, from which are visible only 352 pixels = 44 characters on the display, the rest of 96 pixel = 12 character periods being necessary for the horizontal retrace time.

I used a horizontal 6-bit counter driven by the character clock, which increments from 0 to 55, to generate the address for the characters on a screen line and to determine the changing moments for the horizontal border, sync and blank signals, as in the time diagram shown in Fig. 2.

The active video line actually starts with the left border, immediately after the deactivation of the horizontal blanking signal HBLNK, but the counter starts with the screen line because it must generate the 5-bits address that selects one of the 32 screen characters to display. Only addresses that begin with zero (from 0 to 31 in this case) can be used to drive video memory address inputs directly.

As it can be seen from the timing diagram (Fig. 2.), the counter value is increased starting from zero and when it matches a specific value the control signals will change.

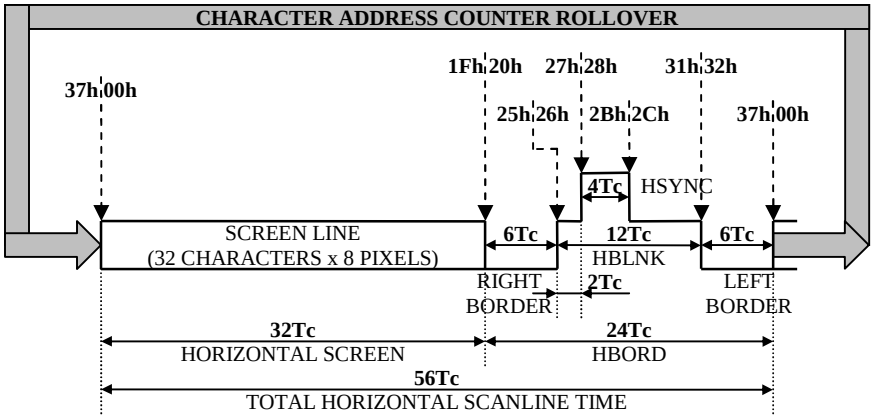

Figure 2. Character counter values which define changing moments of the horizontal signals.

In this way, by decoding the counter outputs, can be generated all the control signals of the video sequencer. Three signals must be generated to control the display on horizontal: the border signal (HBORD) that switches the display from screen to left and right borders, the blanking signal (HBLNK) that turns off the display spot during horizontal retrace and the synchronization signal (HSYNC) that starts this retrace at the end of a visible display line.

The counter must be synchronous to change all its outputs simultaneously and it must have a synchronous reset input too, so it can be restarted from zero in the precise moment when the character clock changes, exactly one period after reaching its final value.

To generate the address for the lines on a screen frame and to determine the changing moments for the vertical border, blank and sync signals, a 9-bit counter is needed, which increments after each line has passed, from 0 to 311. The counter must be driven by a line clock with period T_L = Tc / 56, that changes at the end of each line. The most convenient way to clock the vertical counter is by using the blanking signal generated by the horizontal counter, which activates at the end of a line, before the horizontal retrace.

Like for the horizontal counter, the first 8-bits of the vertical counter are also used to generate the address that selects one of the 192 screen lines to display. The counter starts with screen lines, because only the addresses that begin with zero (from 0 to 191) can be used to drive directly the remaining video memory address inputs, even if the real physical frame starts with the top border.

The video memory address will be formed from the least significant 8-bits of this vertical line counter and from the least significant 5-bits of the horizontal character counter, organized together in a 13-bit wide bus (LA[7..0],CA[4..0]).

This will give a total memory capacity of 8 Ko (kilo-octets) for each video page, even if this will not be entirely used, because the interface resolution needs only 6 Ko = 32 pixels x 192 lines capacity to hold the information for one color of the screen pixels. All the pages use the same video address bus, but with distinctive 8-bit data buses, to implement different colors per each screen pixel. To obtain 4 colors per pixel, the video interface will use 4 paged memory modules of 8 Ko, each with its own shift register driven by the pixel clock, for octet serialization to the RGB lines.

In Fig. 3. is presented the timing diagram with the values of the vertical counter which determines the changing moments for the vertical signals: the vertical border signal (HBORD) that switches the display from the screen to top and bottom borders, the blanking signal (VBLNK) that prevents the lines from being visible during vertical retrace and the synchronization signal (VSYNC) that brings the display spot back to the beginning of the next frame.

Conform to the PAL standard [5], the vertical blanking signal should be about 24 lines, that means with 8 lines wider than 16 T_L line periods, like is considered in the timing diagram from Fig. 3.

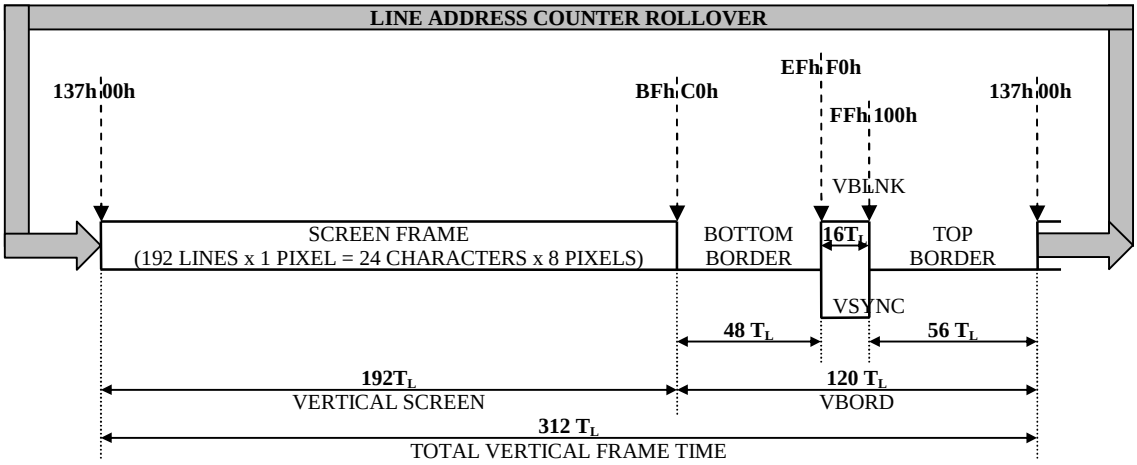

Figure 3. Line counter values which define changing moments of the vertical signals.

Nevertheless, the implementation complexity is directly dependant on the total number of counter value testing points and this approach would lead to one more changing point in the time diagram, which should be also detected. To optimize the design implementation and because the blanking signal begins at the same moment with the synchronization signal that starts the vertical retrace, I used a VBLNK signal that ends sooner by 8 lines, together with the VSYNC signal. This increased the duration of the top border from 48 to 56 lines, but this 8 additional lines are not normally visible on the screen, so this is a quite good solution.

IV. FPGA IMPLEMENTATION

To test different implementations of the video sequencer described above, I used a development board with a Spartan-3 FPGA device [6]. If the project is well designed from the beginning, an additional simulation stage before the physical implementation is not very useful, because the simulation presents the system functionality as it should be, without considering the real system behavior. Because the video interface is a quite complex design, it is better to use a real board to test the FPGA mapping and routing of the implementation directly using a development board and a TV monitor. Following the image on the display, I was able to correct some design mistakes and to modify the implementation to eliminate errors appeared due to the propagation and commutation delays in and between the CLBs (configurable logical blocks [7]) used inside the FPGA chip.

The classical implementation technique of a design that uses programmable digital circuits is based on the use of a hardware description language, like "Verilog" or "VHDL" to obtain a textual description of the design, which is then synthesized using specific tools, provided by the chip manufacturer [3]. Such a description must be realized at the register transfer level (RTL), because the behavioral approach do not always generate a synthesizable code and the designer has no direct control of the implementation.

The RTL description has no visible advantage over a schematic description, because it uses the same basic building pieces and offers no graphical view of the design. It is interesting that even the modern software development environments have actually moved to use a visual code design which offers better intuitive approach. A hardware design using text instead of schematic description is quite unusual, as the hardware engineer is more familiar with intuitive drawings than with the use of a textual code. Unfortunately, almost all software tools provided and developed by the main FPGA producers offers weak schematic support for the design.

I have made for this implementation a schematic description by using the new software environment "Altium Designer" [4], a tool well known by the electronic engineers which design schematics and printed circuit boards for electronic devices. This software has very special graphic facilities for schematic drawing, but it is less known that it supports FPGA design too. The "Altium Designer" environment can be used in CPLD and FPGA designs, by drawing the schematic at high level of abstraction with graphical symbols available to use from generic and specific libraries. The high level of abstraction resides in the variety of available schematic symbols for almost all the circuits and modules generally used in a hardware design, which have both single pin and bus versions.

After this schematic was drawn, it is then translated to a specific format and the software launches third party specific tools provided by the chip manufacturer for the synthesis process, the resource mapping, the placing and routing process and for downloading the obtained bit stream to a real live board [4]. In this way, the design process is independent from the specific tools knowledge, it can be used with most available FPGAs from various producers without any change and the optimal implementation is still maintained due to the background usage of the specific vendor tool, which is transparent to the designer. Another main advantage of this new software environment is that all the above processes can be easily launched by a simple click, allowing the designer to focus on its project and not on the tool usage learning.

Using an evaluation version of the "Altium Designer" software environment, together with the "Xilinx ISE WebPack" free tool which is launched in background, I have succeeded to implement a functional schematic for this video sinchro-generator, after some unsuccessful iterations. The implementation was tested on a "Digilent Spartan-3" development board, connected to a PC by the JTAG-3 parallel cable [8]. The clock generation part of the schematic that includes the implementation of the pixel counter, the horizontal and the vertical counters is presented in Fig. 4.

This board is based on a Xilinx XC3S200 from the Spartan-3 FPGA family [6] and has an integrated oscillator that provides 50 MHz clock frequency. To generate a 14 MHz (megahertz) clock, which is double of the pixel clock frequency, I used one special digital clock manager (DCM) module available in Spartan-3 FPGA series, which has a digital frequency synthesizer block (DFS) capable to derive other frequencies at the CLKFX output from the input frequency CLKIN, using he following formula [6]:

$$f_{CLKFX} = f_{CLKIN} \cdot \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE} \qquad (1)$$

The CLKFX_MULTIPLY and CLKFX_DIVIDE are 5-bit integer constants, ranging from 2 to 32 and from 1 to 32, respectively. For 14 MHz frequency output, I used a 7 / 25 factor in the formula (1) which was set by schematic symbol parameters. I have chosen to start from a double clock frequency because, if a CPLD implementation might be made in the future, an external clock oscillator would be needed, the CPLD circuits having no such DCM modules. An external oscillator with quartz crystal is easier to find at 14 MHz and this doubling allowed me to also use a 4-bit (divide-by-16) pixel counter available as one single graphical symbol, instead of a 3-bit one which is not.

All the counter symbols used in the schematic (Fig. 4.) are bus versions of synchronous counters with clock enable and synchronous reset. The pixel counter generates the pixel address outputs PA[3..0] which will be used for video memory load and serialization. The PA3 output is also used as an inverted clock signal to drive the horizontal counter, which is made from two 4-bit and 2-bit counters for optimization purposes, with buses joined together by special symbols to form the CA[5..0] character address bus.

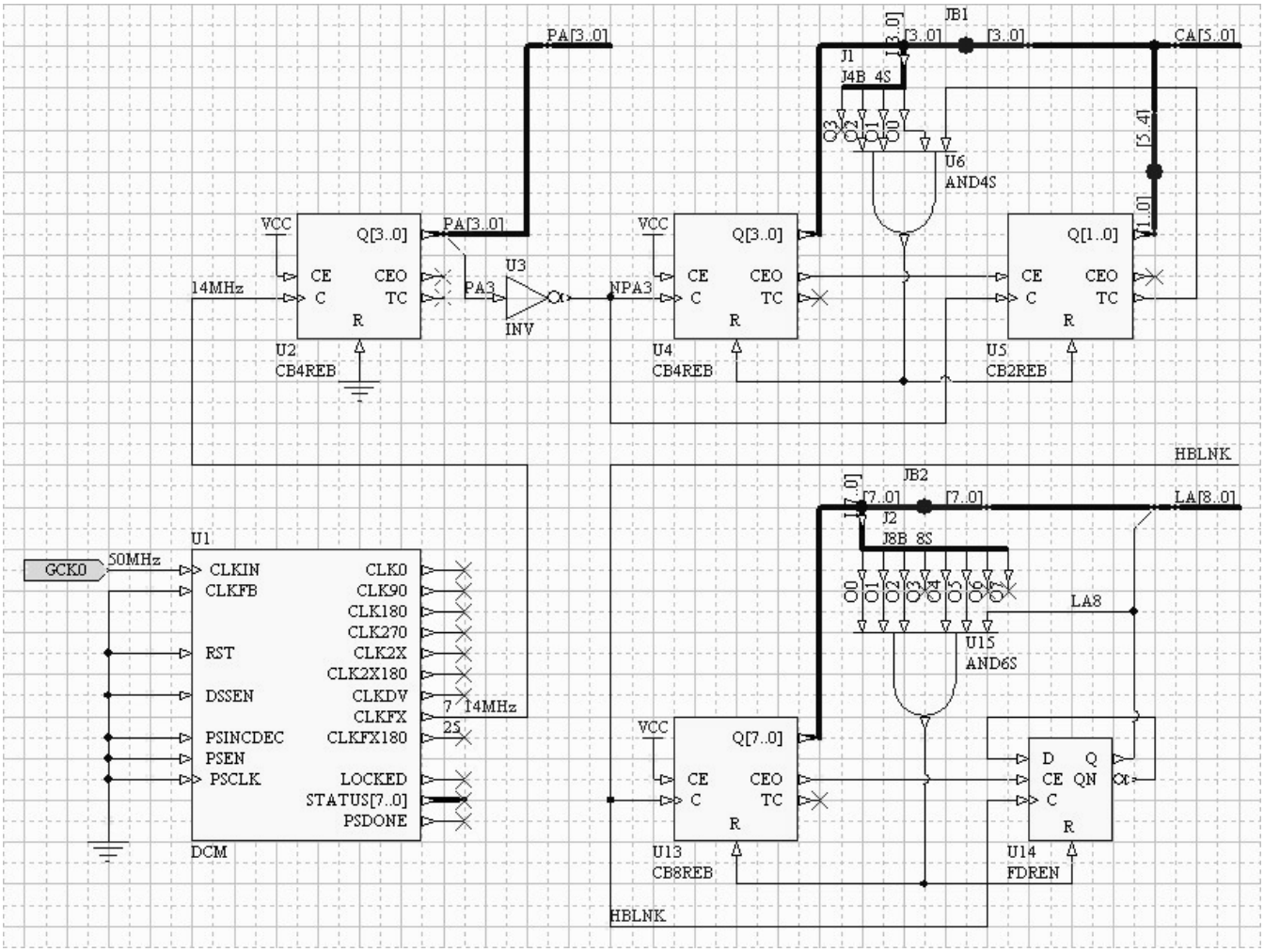

Figure 4. Schematic diagram used to implement the clock generator and the pixel-character-line counters.

The 9-bit vertical counter which generates the line address bus LA[8..0] is clocked by the HBLNK signal decoded from the horizontal counter and it is implemented using one 8-bit predefined counter and a distinctive edge triggered D-type flip-flop, with clock enable and synchronous reset too. As mentioned before, the horizontal counter divide-by-56 the pixel frequency to obtain the character frequency and the vertical counter divide-by-312 the character frequency from the first one to obtain the final 50 Hz frame frequency.

To reset the counters when they reach maximum values (55, respectively 311), I used an elegant method by detecting only the moment when the appropriate bits become logical "1" in the counter word combination. The counters only increase their values and there is no possibility to reach another binary combination with the same bits at logical "1" too, because they will rollover back to zero at first such detection. An AND gate detects the reset combination and activates the reset input for the counters, which will be considered immediately on the next clock positive edge. The splitting of the horizontal counter in two also allowed me to use the terminal count output from the second 2-bit part, which is active only when this counter part reaches "11". This reduced the number of required inputs for the AND gate by one.

The schematics used to generate the horizontal and vertical signals by detecting when the counter value is in a specific range are presented in Fig. 5. This optimal logic was obtained by inspecting values from the timing diagrams and not by using any kind of minimization algorithm. There are two approaches of signal activation while the counter is between two specific values [9].

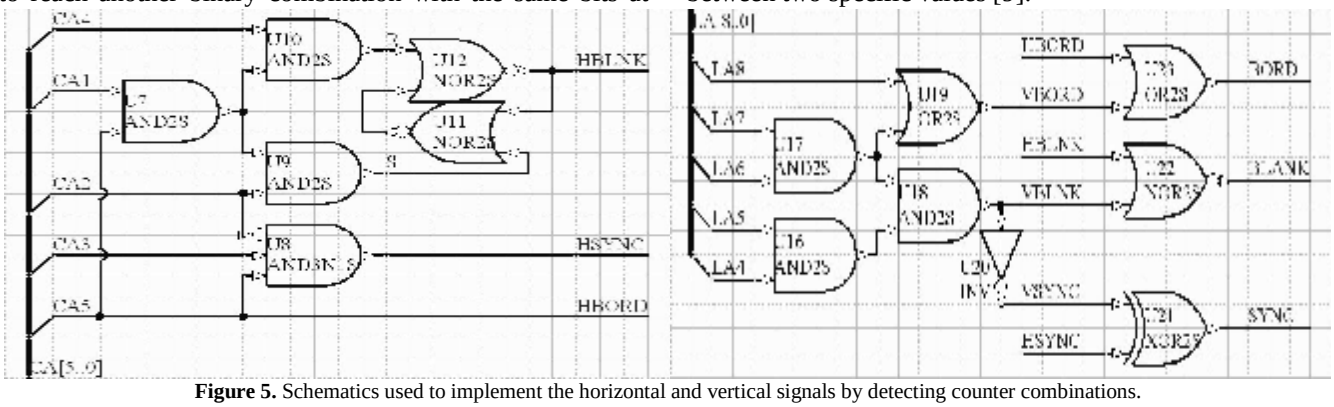

Figure 5. Schematics used to implement the horizontal and vertical signals by detecting counter combinations.

The first solution is to use a combinatorial logic to decode all the combinations from the activation interval. This approach is advantageous only when the value range is between powers of two, when the logic becomes simpler, but it can also generate short glitches due to the different propagation delays when more bits are changing simultaneously, as the counter increases (propagation hazard [9]). I used this method for HBORD and HSYNC signals, and for all the vertical signals, avoiding different propagation delays for a gate, when its inputs change simultaneously.

The second method is to use a sequential logic with flip-flops which are set when the counter reaches the activation combination of the signal and reset when the counter reaches the deactivation combination. This approach will lead to simpler decode logic and reduces the combinatorial hazard, but it consumes more sequential resources. I used this second method for HBLNK signal, because it is used to clock the vertical counter, and it must be glitch free.

The vertical decode schematic also contains the necessary logic to generate global BORD and inverted BLANK signals. The vertical synchronization signal is generated by inserting positive HSYNC pulses into the negative VBLNK signal with a XOR gate, to achieve the characteristics of the PAL sinchro-complex signal [5].

Figure 6. White central screen and black border on a small TV monitor.

V. CONCLUSION

The interface described here was tested on a small TV monitor with RGB and SYNC inputs, by adding a multiplexer that switches all the RGB outputs from logical "1" to ground with the selection signal controlled by global border signal BORD and with input enable controlled by the global blanking signal BLANK, which was generated active low. On the monitor display, a white rectangle was visible in the center, with black border around, like in Fig. 6. This confirms that the vireo sinchro-generator is working exactly as proposed.

The next work will be to design, implement and test the video memory scanner for this TV mode of the interface in the same FPGA or CPLD circuit, to use it with a real micro-system in a series of applications and the research to develop a new VGA monitor compatible mode of the entire interface.

Besides some applications that may have use of the interface itself, this paper is also a good example for interesting hardware design ideas and for new present technologies used in programmable logic design (FPGA or CPLD) implementation.

REFERENCES

[1] Stuart R. Ball, "The 8051 Microcontroller", 4th edition, Prentice Hall, 2006.
[2] J. Naylor, D. Rogers, "Inside Your Spectrum. An Introductory Guide to the Hardware", Sunshine, 1984.
[3] S. Kilts, "Advanced FPGA Design. Architecture, Implementation and Optimization", John Wiley & Sons, Inc., 2007.
[4] Training Manual, "Altium Designer FPGA, Software and Systems Development", Altium Limited, 2007, Available: http://www.altium.com/files/training/Module5FPGADesign.pdf.
[5] Recommendation ITU-R BT.470-6, "Conventional Television Systems", International Telecommunications Union, 1998, Available: http://www.itu.int/rec/R-REC-BT.470/en.
[6] Complete Data Sheet, "Spartan-3 1.2V FPGA family", Xilinx Inc., 2007, Available: http://www.xilinx.com/support/documentation/
[7] data_sheets/ds099.pdf.
[8] S. Brown, J. Rose, "FPGA and CPLD architectures: a tutorial", IEEE Design & Test of Computers, Vol. 13, No. 2, IEEE Computer Society, 1996.
[9] User Guide, "Spartan-3 Starter Kit Board", Digilent Inc., 2005, Available: http://www.digilentinc.com/Data/Products/S3BOARD/
[10] S3BOARD-rm.pdf.
[11] M. Morris Mano, Charles R. Kime, "Logic and computer design fundamentals", 3rd edition, Prentice Hall, 2004.