μAUS: A-mode Ultrasonic Imaging Subsystem

Group 4: James, Sun, Sahni
CSE 462M
April 30th, 2007

Abstract

Conventional A-mode imagining devices are large, expensive and not energy efficient. To overcome these shortcomings, we propose the μAUS, an A-mode Ultrasonic Imagining Subsystem for diagnostic ultrasound. The μAUS is low-cost, compact, and low power; therefore, it can potentially create new applications area for diagnostic ultrasound. The μAUS consists of three major components: an envelope detector for rectifying and filtering A-mode data, a VGA drawing unit for generating real-time A-mode display, and a USB interface unit for transferring A-mode vectors to a PC for display using custom software. We describe both the prototype construction using a Xilinx FPGA and FTDI USB chip, and the PC software written is Microsoft C++.

I. Introduction

μAUS is an A-mode Ultrasonic Imagining Subsystem for diagnostic ultrasound. It can determine eye length, a key factor in the diagnosis of common eye disorders, such as cataracts (the leading cause of blindness worldwide [1]). Removing a cataract involves replacing the lens with an artificial lens implant, A-mode (amplitude mode) displays assist in determining the optimal size of the replacement lens. Other A-mode imaging devices exist, but the μAUS is low-cost, relatively small, and USB-compatible.

To minimize cost, the μAUS uses commercially available hardware components. The μAUS uses a Xilinx XC3S200 FPGA [2] and the FTDI FT2232C chip for USB 2.0 support. Xilinx's Spartan-3 board [3] (Fig.1) prototypes the μAUS design, and the DLP-2232M evaluation module [4] (Fig. 2) which uses the FTDI FT2232C chip (Fig. 3)

prototypes the USB 2.0 interface. The Xilinx Spartan-3 board connects to DLP-2232M [5] evaluation module via a custom adapter board (Fig. 4). By keeping the design relative simple, we attempt to ensure the μAUS will occupy an approximate area of 2-3 square inches. Furthermore, the μAUS is USB 2.0 compatible. USB 2.0 is an industry standard bus that is low-cost, PC-compatible, and backward compatible with previous USB versions [4].

This report presents the architecture of the μAUS, its prototype implementation, and development of A-mode display software. In Section II, we give a background on A-mode imaging. In Section III, we describe the hardware architecture of the μAUS. In Section IV, we present VHDL implementing the design. Section V provides results of synthesizing our on Xilinx FPGA, including timing analysis. Section VI covers the software application used to display A-mode on the PC host. Finally, in Section VII, we suggest new application areas for the μAUS and suggest future enhancements.

II. A-Mode Imaging

A-mode imaging can detect defects in the eye (ophthalmology) by determining its depth and dimension. In A-mode imaging, spikes of different heights (amplitudes) represent the echoes received from an emitted ultrasound wave. The spikes are a one-dimensional representation of the depth (x-axis) and amplitudes (y-axis) of the echoes [6] (Fig. 5). A-mode imaging technology makes use of well-known ultrasound principles.

Ultrasound refers to sound waves that are inaudible to the human ear or equivalently sound waves of frequencies above 20 KHz [7]. Probes in A-mode devices

typically use sound waves of extremely high frequencies (approximately 10 MHz) to allow "for not only restrict depth of penetration of sound into the body but also excellent resolution of small structures" [7].

Since the eye is composed of both solids and liquids, A-mode imaging uses the basic principle that sound waves travel faster through solids than liquids [8]. An ultrasound probe tip emits an ultrasound beam that travels through the eye. The ultrasound wave moves through the "solid cornea, the liquid aqueous, the solid lens, the liquid vitreous, the solid retina, choroid, sclera, and then orbital tissue" [9]. An echo, produced at each solid-liquid interface, bounces back into the tip of the probe. The A-mode imaging device converts the echoes received from each interfaces to spikes (Fig. 6). If the difference between the media at the interface is large then the echo is strong and the corresponding spike is high. On the other hand, if the difference at the interface is less then the echo is weak and the spike is short [9].

III. System Architecture

The µAUS interfaces with the ultrasound backend, a PC host, and a 1024×768 VGA monitor (Fig. 7). Using a VHDL module, we simulate the pulser/receiver and the A/D in ultrasound backend. The DLP-2232M module implements the USB interface to the PC host, and a custom VGA drawing unit interfaces with the VGA monitor. The programmable logic components of the Xilinx FPGA implements the µAUS design including the simulated ultrasound backend.

The μAUS operates in two modes concurrently: host mode and VGA mode. In host mode, a start-byte (0x01) initiates the transfer of data and a stop-byte (0x00) terminates it. In VGA mode, a switch on the Spartan-3 board turns A-mode display on and off. The modes work together allowing the user to transfer data to the application on the PC host and view a real-time A-mode display on the local VGA monitor at the same time.

The μAUS performs envelope detection on the sampled RF data received from an ultrasound backend, uses USB 2.0 to transmit the processed data to a PC host for display using custom software, and generates a real-time A-mode display on a local VGA monitor (Fig. 8). Based on these functions we partition the subsystem into four components: an envelop detector (ED), a data controller (DC), a VGA controller (VC), and a system controller (SC) (Fig. 9, Fig. 10).

The envelop detector, ED, receives sampled RF data (ultrasound echoes) in the form of A-mode vectors from the A/D converter in the ultrasound backend. The ED performs envelope detection using rectification and low-pass filtering. A FIFO and a dual-port memory (DPM) store the envelope-detected data for transfer to the DLP-2232 module and for display on the VGA monitor respectively.

The data controller, DC, controls the transfer of data from the FIFO to the DLP-2232M module. It reads a byte from the FIFO and writes it to the DLP's transmit-buffer. This read (from FIFO) - write (to DLP) cycle continues until the FIFO is emptied. The DLP-2232M module, clocked at 50MHz, implements the USB interface to the PC host. The FTDI FT2232C, on the DLP-2232M module, operates in its FIFO IO mode and uses one of its two IO channels for data transfer. The FTDI FT2232C supports a data rate of up

to 1MB/s, which is sufficient for transferring 60 a modes vectors per second (approximately 1Kb/s). A custom software application running in a window on a PC host receives data via a USB port and generates an A-mode display. In addition, the PC software transmits either a start or a stop byte codes to DLP module over the USB connection to initiate and terminate the transfer of data respectively

The VGA controller, VC, interfaces with the DPM and a local 60 Hz, 1024 × 768 VGA monitor via a 15-pin VGA port. The DPM allows data to cross the 50MHz-65MHz clock boundary; the 50MHz clock domain is the write-side and the read-side is the 65MHz domain of the DPM. The VC reads 1024 bytes of data from the 65MHz side of the DPM and displays down-sampled (2048-byte mapped to 1024 pixels) A-mode vectors in real-time on the attached VGA monitor.

The system controller, SC, controls the acquisition of data from the ultrasound backend and the writing of data to the FIFO and the DPM. It acquires a 2048-byte, A-mode vector during the vertical retract period of a 60 Hz VGA display. The SC also asserts the write enable signals on the FIFO and the DPM to write 2048 bytes of data to them.

IV. VHDL Implementation

Using VHDL [10], we implemented each component of the μAUS separately. We used ModelSim [11] for simulation and Xilinx ISE [12] for synthesis. This modular approach enables the parallel implementation of components, facilitates change, and simplifies system integration. Each team member implements a different module concurrently. It is less complicated to find and fix errors in a module in isolation as

opposed to a module integrated into the system. Furthermore, this approach also gives us a better understanding of the individual parts, so integration, the sum of the parts, becomes clear.

However, modules interface with other components, so implementing a module in isolation is difficult. Thus, we develop "mock" versions, stubs, of the components connected to a module in order to test the module. These stub components implement simplified interfaces of the real components; the stubs implement only the functionality need to verify the correctness of the module. Using this approach, this section describes the design, code, and simulation of the envelop detector (ED100), the data controller (DC100), the VGA controller (VC100), and the system controller (SC100). Finally, we discuss the integration of the individual modules.

The Envelope Detector (ED100)

The ED100 performs envelop detection on the A-mode data received from the A/D in backend by rectification and low-pass filtering.

The rectification operation maps an 8-bit value that ranges from 0xFF-0x00 (255-0), X, to a value that ranges from 0x80-0x00 (128-0), Y. To rectify the X, we set the most significant (msb) bit of the result, Y[7], to 0, and assign the other 7 bits of Y, Y[6:0], based on the msb of X, X[7]. If X[7] is 1 then we set Y[6:0] to X [6:0]. Otherwise, we assign Y[6:0] to the inverse of X[0:1] (Fig. 11, Fig. 12). Using VHDL, we implement the rectification operation a single concurrent statement.

We use a FIR filter as our low pass filter. The general difference equation for a FIR filter is shown in Fig. 13, where d_n's are the data that comes out of the rectifier, and c_n's are predetermined constant coefficients ($c_1 + c_2 + \ldots + c_n = 1$). n represents the tap of the filter, which also means the window size that we use to process the data. In most cases, more taps means better filtering results. Since the Spartan3 board has 12 dedicated multipliers, we want to make use of them to save resources for other parts of our design. With 12 multipliers, we can only get a maximum of 23 tap filter with d1 and d23, d2 and d22 … d11 and d13 sharing the same coefficients. The new difference equation is now shown in Fig. 14. The block diagram of the FIR filter is shown in Fig. 15. In order to implement this FIR filter in VHDL, we first need 23 registers to hold the rectified data values. Then we add d_1 and d_{23}, d_2 and d_{22} … d_{11} and d_{13}, and sign extend to 9 bits. For pipelining purpose, we use a 9-bit register to create a 1 clock cycle delay before we feed the data into the multipliers. The multipliers are generated using Xilinx CoreGen with parallel coefficients, so that the coefficients can be changed for different applications. After multiplication, we sum the products from the twelve multipliers, which should be the last step of our filtering process. The result from this final adder is 17-bit long, so we need to perform rounding and truncating to map it back to an 8-bit A-mode data. Note that in reality, the multiplier coefficients should be floating numbers between -1 and 1; while there are only integer types in VHDL. To solve this problem, we first use 8-bit 2's compliment numbers to represent coefficients, which means that we multiply all the coefficients by 127. Thus we need to divide the result by 127 to produce correct filtered data. We choose to do this after the summation step in order to reduce rounding errors. Since it save many logic gates to perform division by 128 (shift right by 7 bits) than to divide by 127, and the error is not

noticeable in our application, we simply take the (15 downto 8) portion of the 17-bit final sum as the filtered result. We also consider .5 rounding by checking the 7^{th} bit of the final sum, which means that if the 7^{th} bit is 1, then the filtered result has a fraction greater than or equal to .5, thus we round up 1 for the A-mode data; if the 7^{th} bit is 0, then the filtered result has a fraction less than .5, thus we do not round up for the A-mode data. The VHDL implementation for this rounding and truncating block is shown in Fig. 16.

The Data Controller (DC100)

The DC100 interfaces with the FIFO and the DLP module. Using an IP core from Xilinx's Core Generator [13], we implement the FIFO with a 2048-byte depth and 8-bit data width. Using one of the two expansion connectors on the Spartan 3 [3, pp. 47-51], we connect the DLP to module to the Spartan 3 via the custom adapter board (Fig. 4). To interface with the DLP, we have to meet the timing specifications for its 245 FIFO Interface mode [14, pp. 19-20]. The DC100 has input signals: DLP_RXF_L, DLP_TXE_L, FIFO_EMPTY, and a reset signal (RST_L). The DC100 output signals include DLP_RD_L, DLP_WR, FIFO_RD_EN, WR_EN, and RD_EN (Fig. 24).

The DC100 controls the reading from and writing to the DLP module. First, it reads a byte from the DLP module when there is data available (DLP_RXF_L is low) by forcing DLP_RD_L low then high. Then if it can write (DLP_TXE_L is low) and there is data in the FIFO (FIFO_EMPTY is high), it repeatedly writes data a byte at a time from the FIFO to the DLP by forcing FIFO_RD_EN and DLP_WR low then high. It stops writing if either new data available from the DLP or the FIFO becomes empty. It also enables a tri-state buffer connected to the DLP data bus when writing to the DLP module by forcing WR_EN

high, and enables a flip-flop to store a byte when reading from the DLP by forcing RD_EN high. Using a finite state machine, we implement this read-(read/write) cycle and the timing protocol for the DLP's interface (Fig. 25). The FSM uses the two-process pattern to describe the FSM in VHDL (Fig. 26, Fig. 27). Back-to-back synchronizers stabilize the DLP_RXF_L, DLP_TXE_L, and RST_L inputs that are asynchronous the FSM's clock, and an output flops removes extra "wiggles" (due to a race condition) from the FSM's outputs (DLP_RD_L, DLP_WR, WR_EN, and FIFO_RD_EN).

Using component stubs for the ED100, VC100, and SC100, we simulated and synthesized, the DC100 (Fig. 21). The ED100 stub, ED50, rectifies the backend data. The VC100, stub, VC50 outputs a vertical sync (VS) at a 60Hz frequency. Lastly, the SC50 pulses the backend when VS is high and enables the DPM when GO_H is high. We verify that the DC100 reads data from DLP and writes data to the DPM during the vertical retrace period of the VC50 (Fig. 29).

The VGA Controller (VC100)

The VC100 interfaces with the DPM, the SC, and the VGA monitor. Using an IP core from Xilinx's Core Generator [15], we implement the DPM as a 2048-byte dual-port block memory module with an 8-bit data width. Using the its COL output, the VC100 reads 1024 of the 2048 bytes from the DPM to generate a down-sampled A-mode display on the monitor. The SC100 uses the VS output from the VC100 to determine when to acquire data by pulsing the BE. A high VS signal from the VC100 indicates monitor is performing vertical retrace; that is, the monitor's beams are off as it returns to the upper left-hand corner of the screen. Lastly, it interfaces with the VGA monitor through the 15-pin VGA

port on the Spartan 3 board [3, pp. 21-25] by using the five VGA control signal: red (R), green (G), blue (B), vertical sync (VS), horizontal sync (HS) (Fig. 30).

The VC100 includes a VGA timing generator, a plotting algorithm, two hardware-cursors, and a character display. Based on "standard" timing values for vertical (pixels) and horizontal (lines), we generate a sync pulse, active video region, front porch, and back porch [16]. We offset the horizontal and vertical sync counters to start the count (from 0) at the beginning of the active video region [17, p. 42] (Fig. 31). The offset counters simplify the plotting algorithm, which works by comparing the values of two consecutive A-mode values to decide whether to "shade" up or down (Fig. 32, Fig. 33). Although the plotting algorithm uses 1024 points (equal to the pixel width), we use a 2048-byte DPM to simplify the System Controller that writes 2048 to the FIFO for the DC100.

The cursors are implemented using three counters, two 11 bit counters for keeping their horizontal location across the monitor and a 26 bit counter to generate a slower clock to make it easier to use the cursors at a manageable rate. Under the pixel counter process which is guided by the 65 MHz clock in VC100 we iterate the 26 bit counter to its maximum value and then invert the tick of the slower clock. Another process is added to implement the cursor events which is guided by the slower clock generated in pixel counter. Cursors move left and right using the 4 buttons provided on the Saprtan-3 board. Right cursor's movement is controlled by button 0 for moving right and button 1 to move left whereas left cursor's movement is controlled by button 2 for moving right and button 3 to move left. Now to show the cursor on the screens we added two extra if conditions

which compared the left and right cursor counter to the current pixel and if the condition is true then turning that respective pixel on to show the cursors.

To show the distance between the cursors we needed to come up with a way to display four numbers on the screen. In order to display these characters we chose to represent each character using 8 columns and 8 rows. This adds up to managing 64 pixels for every character. Therefore, we hand computed 64 bit representation of each number from 0 to 9 and stored it in char_rom package in which every 1 represented pixel on and a 0 represented pixel off (Fig. 34). Now to turn the pixels on and off we added a conditional statement checking that if the pixel is in the region of character display i.e. line 50 to 57 and pixel 550 to 590. Then we used a case statement to check for the current line number and pixel to help us determine the exact current position upon which we called the procedure that we created to check the digit needed to be displayed and to use their respective bit pattern to set the pixel's state.

Using component stubs for the ED100 and SC100, we simulated and synthesized, the VC100 (Fig. 35, Fig. 36, and Fig. 37). The ED50 stub rectifies the backend data. During the vertical retrace period, the SC50 stub first pulses the backend, then asserts write enable (WEA) on the DPM until the byte counter value equals 2047. To allow user to turn on and off the A-mode display, we use a switch on the Spartan board (GO_H). The switch enables the write (ENA) of the DPM. To enable the read side (ENB), we take the logical AND of the HS and VS signals, which gives an approximation of the horizontal and vertical active video region.

The System Controller (SC100)

The SC100 interfaces with every component except the envelope detector and the DC100 (Fig. 40). Inputs to the controller include COUNT (from the byte counter), USB_GO_H (from the DLP module), and VC_VS (from the VC100). Its outputs include CNTR_RST, FIFO_WR_EN, BE_PH, BE_PL, and DPM_WEA (Fig. 41). The SC100 controls the acquisition of data from the BE and the storage of data to the FIFO and DPM.

During the vertical retrace period (VC_VS is high), the SC100 acquires a 2048-byte, A-mode vector from the backend. “Pulse high” (BE_PH) and “pulse low” (BE_PL) initiate the transfer of a data vector clocked at 50 MHz from the ultrasound backend. BE_PH and BE_PL both remain active for 40ns with an active BE_PL occurring after an active BE_PH. Using a byte counter to count clock cycles, the SC100 writes a total of 2048 bytes to the FIFO and the DPM at the rate of one byte per cycle. The FIFO_WR_EN and DPM_WEA outputs enable the FIFO and the DPM respectively. The SC100 forces DPM_WEA high if VGA_GO_H is high and drives FIFO_WR_EN high if USB_GO_H is high. Using a finite state machine, we implement the SC100’s read-write (pulse-enable) cycle (Fig. 42). Similar to the DC100’s FSM, we use the two-process pattern to describe the SC100’s FSM in VHDL (Fig. 43). Back-to-back synchronizers stabilize the VC_VS input that is asynchronous the FSM’s clock, and output flops on the DPM_WR_EN, and FIFO_WR_EN remove extra “wiggles” due to possible race conditions. We simulated the FSM to verify its operation (Fig. 44, Fig. 45, and Fig. 46).

System Integration

Combining the DC100, VC100, and SC100 was easy since we had designed, simulated, and synthesized the test architecture for the VC100 and the DC100 separately. The μAUS architecture (Fig. 47) combines these test architectures (Fig. 28 and Fig. 37). We replace the NAND gate used in the DC100's test architecture for enabling the DPM's write-side with its logically equivalent NOR gate. Using structural VHDL, we connect the components together (Fig. 48). We simulated the integrated design to verify that it operates correctly (Fig. 54, Fig. 55, Fig. 56, and Fig. 57).

V. Timing Analysis/Results

In this project, we designed and implemented the μAUS, a portable, low-cost, and USB-compatible A-mode Ultrasonic Imaging Subsytem for ophthalmology diagnosis by determining eye length. Our system supports cursors to calculate eye length. The system is composed of an envelope detection unit, VGA drawing unit, and USB interface. Using VHDL, we implemented the envelope detector and the VGA drawing unit on the Xilinx XC3S200 FPGA. Using the FT2232C IC, we implemented the USB interface. A custom application, developed in Microsoft Visual C++, running on a PC receives all data transfer from the μAUS via USB and generates an A-mode display in a window.

Using the Xilinx Spartan-3 board and DLP-22332M Module from DLP Designs, we prototyped the μAUS design. Our design consumed 1,288 LUTs or 33% of the total FPGA resources. The design exceeds the 50 MHz timing constraint and runs at 85 MHz (Fig. 59). To increase the clock rate, we can reduce the combinational logic depth of our circuit.

Although our prototype used a simulated backend, we have a real analogy backend designed for the final product.

We want to package and estimate the cost of our μAUS system so that we can commercialize it in the future. Our system shall be deployed on a 1" x 3" circuit board with the following major hardware components: a Xilinx XC3S200-4FTG256C FPGA, a FTDI FT2232C USB Dual USB UART/FIFO IC, a USB type-B connector, a VGA DB15 connector, two voltage regulators, a 50MHz oscillator, a 65MHz oscillator, and a backend which includes a pulser/receiver and an A/D converter. It is also necessary to have two power control switches and four hardware cursor control buttons. Fig. 60 shows a prototype layout of the major hardware components on the circuit board.

We estimated the cost of the hardware components based on the retail price from Digi-Key Corporation [18] and Mouser Electronics, Inc. [19]. Fig. 61 shows the list of prices or each component considering the largest volume purchase discount. The total price of all the components on the list is $36.02, which indicates that the hardware cost for the μAUS system should be no more than $50. This price can be further reduced by using a cheaper version of the FPGA, Xilinx XC3S200-4TQG144C, which only costs $12.33. Since we estimate the cost based on retail price, mass production may also bring down the cost by a significant amount.

VI. Software Application

Our software application is developed in Microsoft .NET Visual C++ [20]. Its purpose is to communicate with the DLP's VCP driver using the windows VCOMM API

calls defined in the DLL and the library files provided royalty free by FTDI. It plots the A-mode data received from the FTDI USB interface and to plot this data we decided to use a trial version of a graphing package provided by Dundas [21] for Visual Studio 2005.

In order to use the files provided by FTDI in Visual C++, we first imported the DLL and then translated the data types utilized in the methods required for our purposes. Secondly, we redefined all the basic FTDI methods using Common Language Runtime Interop layer concealing the differences between managed code (C++) and unmanaged code (C).

Technical specifications require the application to plot the data at 60 Hz which translates to about 16 milliseconds frame refresh rate or 120,000 points per second. Therefore, we implemented a timer in our application with an interval of 16ms. Our application then achieved the 60 Hz constraint by plotting the complete data vector in stored in a global buffer by first fetching it into a chart buffer, series, completely and then painting the chart on every timer tick. The chart buffer is emptied out before every fetch cycle to minimize the memory usage of our system. The reading method was altered during the course of implementation as we realized that reading the data and plotting it sequentially presents us with a bottleneck and eventually leads us to lose data. Therefore, we implemented a thread dedicated to just reading of the data that would loop itself and which would run simultaneously with plotting.

Our application tries to automatically establish a connection with the FTDI USB at the startup and displays the connection status in the status bar. During the regular operation this status bar is used to indicate different issues that may arise in the process of reading

and writing from the USB. Our application is completely controlled by the user with the exception of establishing the connection with the chip initially. Application's GUI gives user the ability to send read, write, and stop signals from the application to the chip. To further enhance the user experience we implemented the application in such a way so that the user can change visual format of the graph being plotted and zoom in/out of it easily during runtime. The zooming feature also serves as an advanced form cursors implemented in the VGA interface since zoomed in distance is shown on the status bar.

We tested the software under several different configurations and it managed to keep up with the plotting of the data verifying it to be a suitable tool for our prototype.

VII. Conclusions

In today's competitive medical environment, doctors need the flexibility to diagnose patients outside the confines of the traditional office. The extremely small size of the μAUS makes is easy for doctors to use A-mode imaging in "mobile diagnosis." Moreover, the low cost of the μAUS makes it accessible to small medical centers or third-world countries where high-priced ultrasound devices are luxuries. Technologically, the μAUS shows that FPGA-based frameworks are a viable platform for development of ultrasonic imagining devices. Our modular developmental approach enabled the concurrent development of modules, facilitated updates to modules, and simplified system integration. Finally, all μAUS hardware is commercially available, so it will be cheap to manufacture the final product.

In the future, we plan to improve and optimize the current design and investigate the possibility of adding other forms of ultrasound imaging (B-mode, 3D-ultrasound). We also plan to look into other efficient forms of envelope detection such as methods based on the Hilbert transform, and consider other VGA drawing algorithms.

VIII. References

[1] Kupfer C: Bowman Lecture, *The conquest of cataract: A global challenge*. Trans Ophthal Soc UK 104:1, 1984.

[2] Spartan-3 FPGA Family: Complete Data Sheet: Xilinx, April 26, 2006.

[3] *Xilinx Spartan-3 Starter Kit Board User Guide*, Vers. 1.1, Xilinx, May13, 2005.

[4] *DLP-2232M Module / Evaluation Kit Preliminary*, Rev. 1.4, DLP Design, Nov, 2005.

[5] *Universal Serial Bus Specification*, Rev 2.0, Compaq, Hewlett-Packard, et al. Revision 2.0 April 27, 2000.

[6] "History of the AIUM, *http://www.aium.org/aboutAIUM/timeline/1950.asp*, March 20, 2007.

[7] Rhonda G Waldron, “A-Scan Biometry,” *http://www.emedicine.com/oph/topic486.htm*, January 8, 2007.

[8] Rhonda G Waldron, “A-Scan Biometry,” *http://www.emedicine.com/oph/topic486.htm*, January 8, 2007.

[9] Rhonda G Waldron & Thomas M Aaberg, “Ultrasound Principles” *http://www.emedicine.com/oph/topic486.htm*, March 20, 2007.

[10] Peter J. Ashenden, *The Student’s Guide to VHDL*. San Francisco: Morgan Kaufmann Publishers, Inc, 1998 .

[11] *ModelSim EE/Plus Refrence Manual: Model Technology Inc*, Ver. 5.2, April 1999.

[12] *Xilinx ISE 8.1i Software Manuals and Help*: Xilinx, 2007.

[13] *FIFO Generator User Guide*: *LogiCORE*, v3.2, Xilinx, September 21, 2006

[14] *Preliminary*: *DLP-2232M Module/ Evaluation Kit*, DLP Design, Rev 1.1, Aug 2004.

[15] *Dual-Port Block Memory Core Product Specification* : *LogiCORE*, v6.3, Xilinx, August 31, 2005

[16] Nathan Ikes, “VGA Video”, *http://www-mtl.mit.edu/Courses/6.111/labkit/vga.shtml*, April 29, 2004

[17] Enoch Hwang, “Build a VGA Monitor Controller,” *Circuit Cellar*, Issue 172,pp. 38-43, November 2004.

[18] “Digi-Key Catalog T072”, Digi-Key Corporation *http://dkc1.digikey.com/us/en/pdf/Current.html*, Arp 30, 2007

[19] “All Products “, Mouser Electronics, Inc, *http://www.mouser.com/search/Default.aspx*, Arp 30, 2007

[20] *Visual C++.* Computer software. Microsoft Corporation, 2005 *http://msdn2.microsoft.com/en-us/visualc/default.aspx*

[21] *Dundas Chart for .NET.* Computer software. Dundas Data Visualization, Inc, IBM-PC, 2007

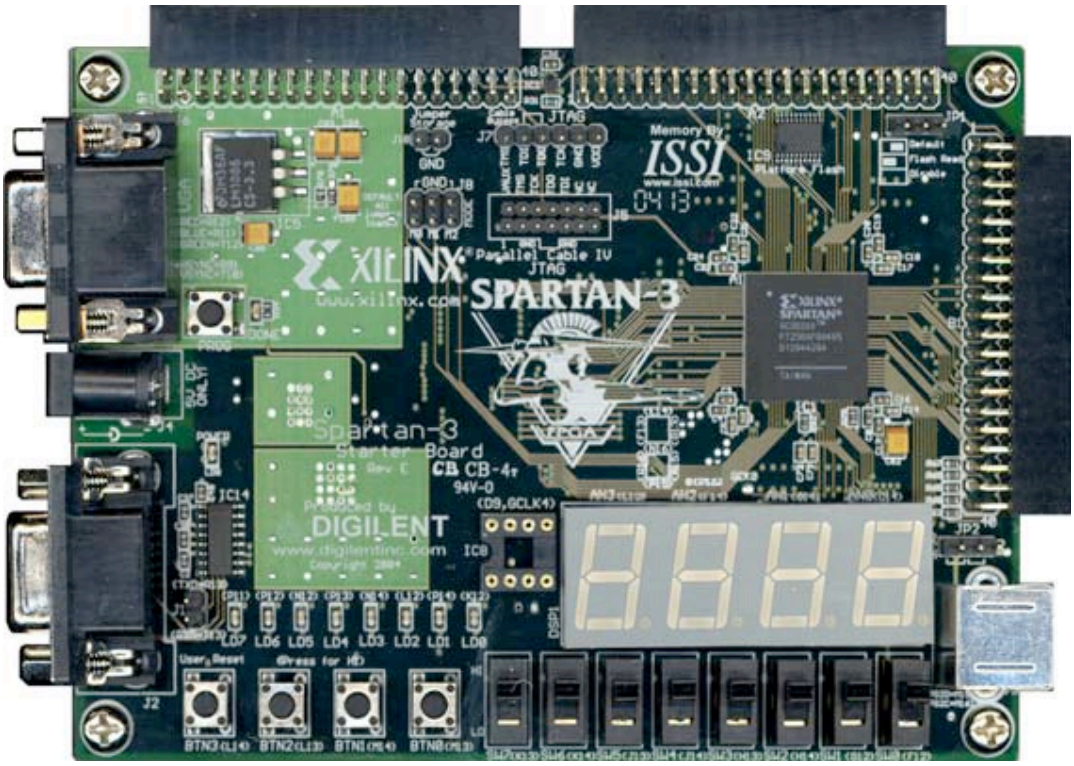

Fig.1. Xilinx's Spartan-3 Board is used to prototype the µAUS design. (Source: Digilent Spartan-3 System Board Brochure)

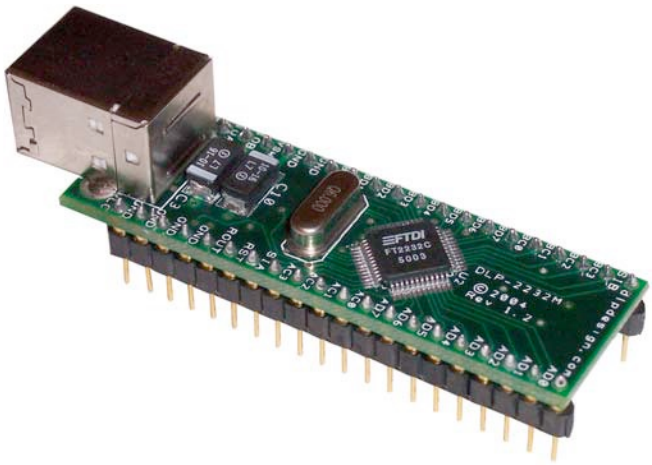

Fig. 2. DLP-2232M Module is used to prototype the USB interface. The DLP-2232M utilizes the FTDI-2232C chip. (Source DLP-2232M DLP Design, Inc.)

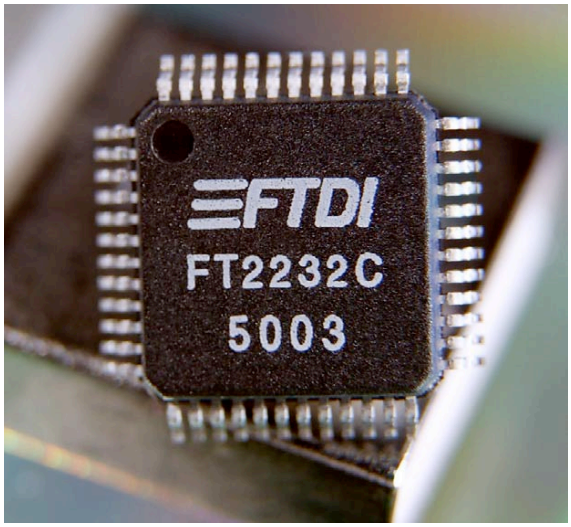

Fig. 3. FTDI-2232C chip provides a "single-chip" USB solution for the µAUS design. (Source: DS2232C v1.3, FTDI)

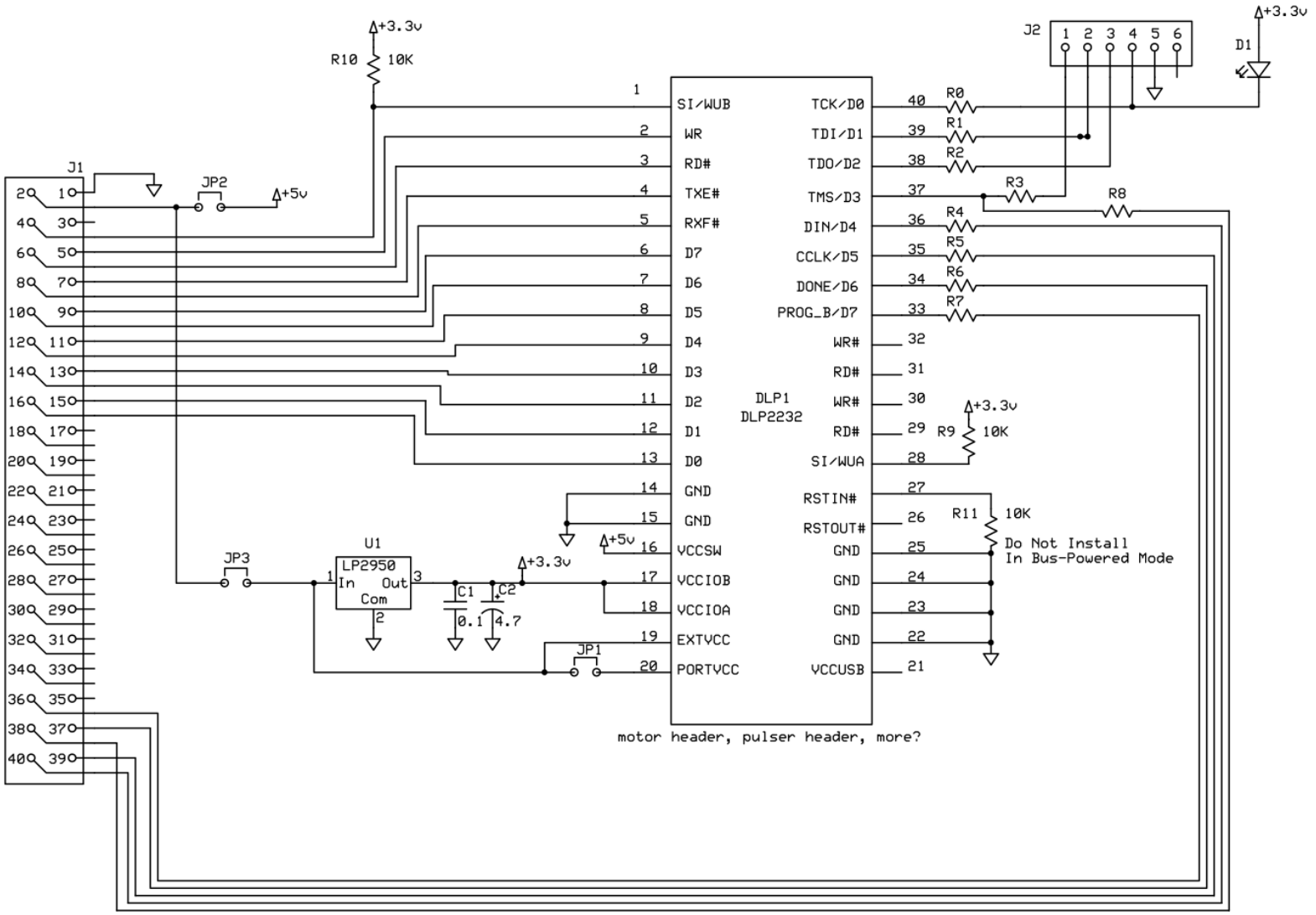

Fig. 4. The schematic for the custom adapter board used to connect the DLP-2232M module to the Spartan-3 board. The adapter connects to one of the board's three 40-pin expansion connectors. (Designed by WDR- William David Richard)

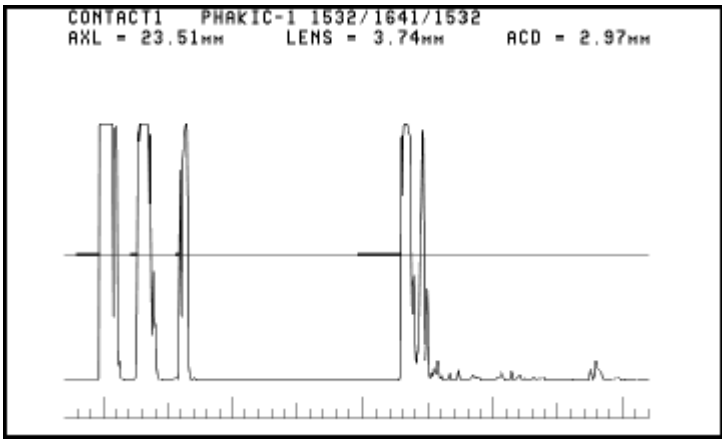

Fig. 5. A-mode display of an eye (with its natural lens) measured with contact A-scan biometry. (Source: Intraocular Lens Power Calculations, H. John Shammasa)

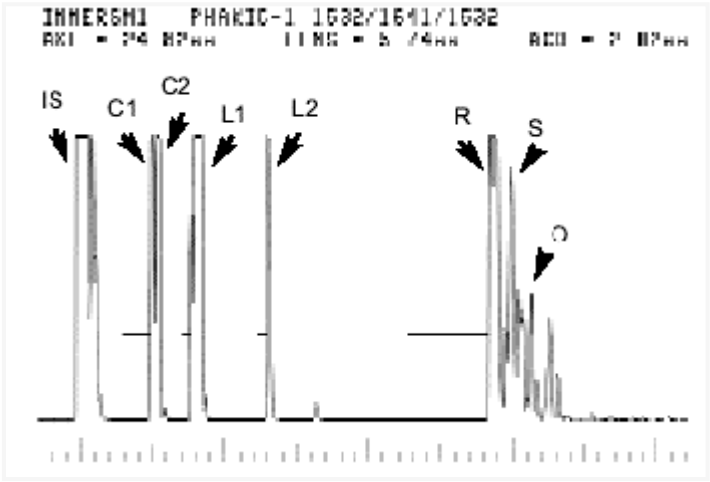

Fig. 6. A-mode ultrasound display of the different “echo spikes” .The initial spike (IS), the anterior (C1) and posterior (C2) corneal surfaces, the anterior (L1) and posterior (L2) lens surfaces, the retina (R), sclera (S), and orbital tissues (O). (Source: Intraocular Lens Power Calculations, H. John Shammasa)

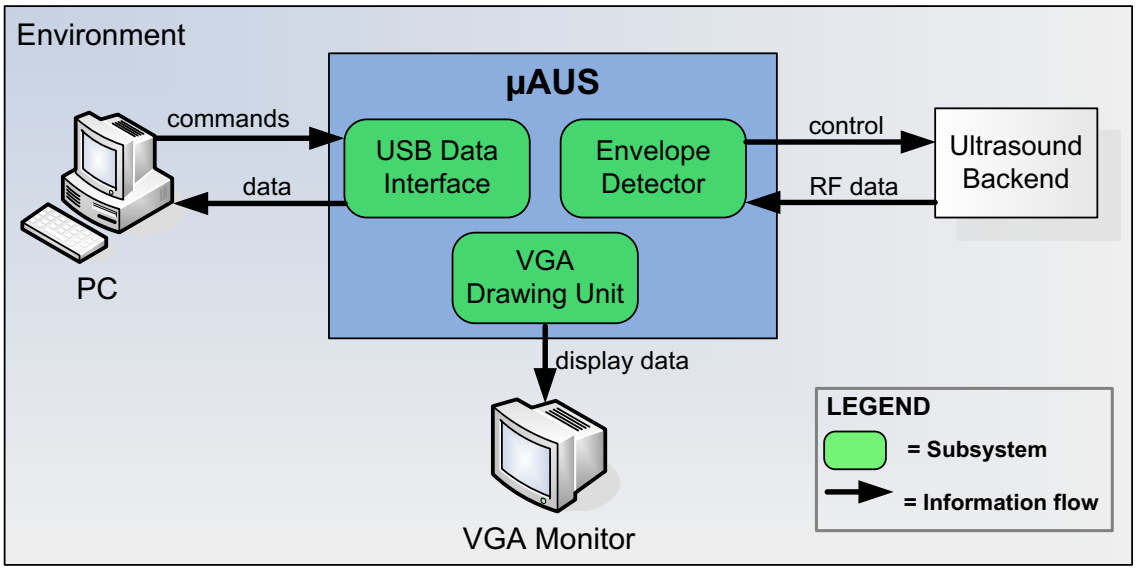

Fig. 7. System context diagram showing a highest-level view of the system. The input and outputs from/to external actors indicate the relationship and connections to the environment.

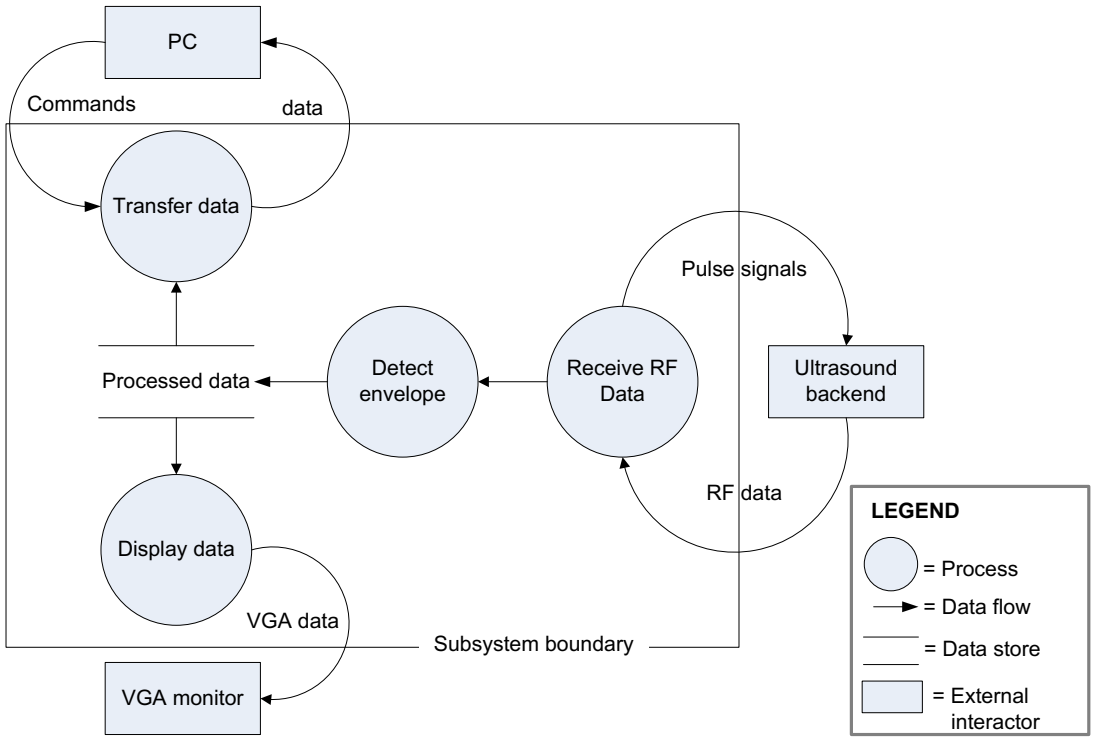

Fig. 8. Context-level, data flow diagram (DFD) showing the flow of data within the μAUS. The DFD also shows functions performed on the ultrasound data as if flows from the backend to the PC and the VGA monitor.

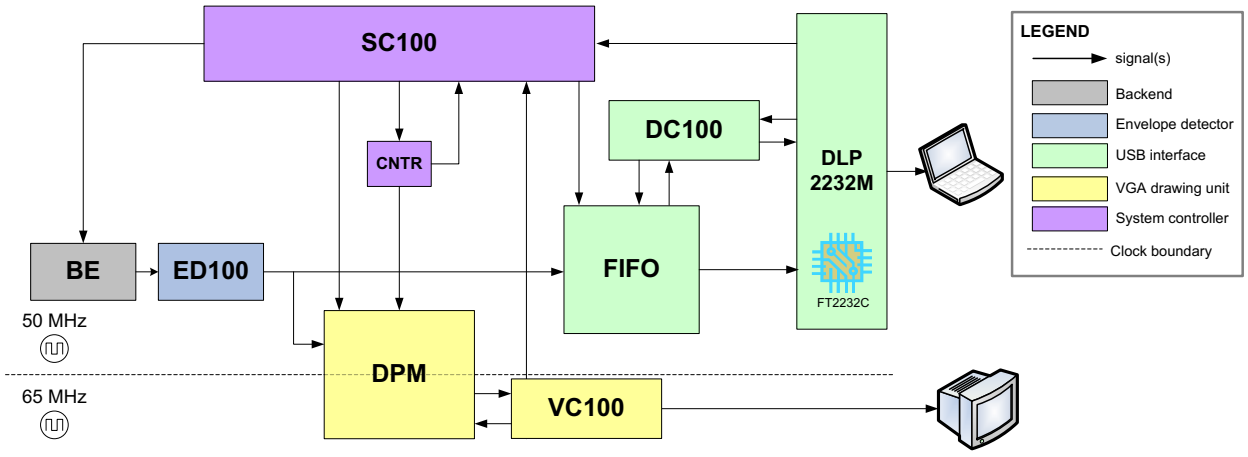

Fig. 9. Block diagram of the µAUS showing its sub-components. The dual-port memory (DPM) and the VGA controller (VC100) implement the VGA drawing unit. The FIFO, data controller (DC100), and the DLP-2232M module implement the USB interface. The Byte counter (CNTR) and the SC100 implement the system controller.

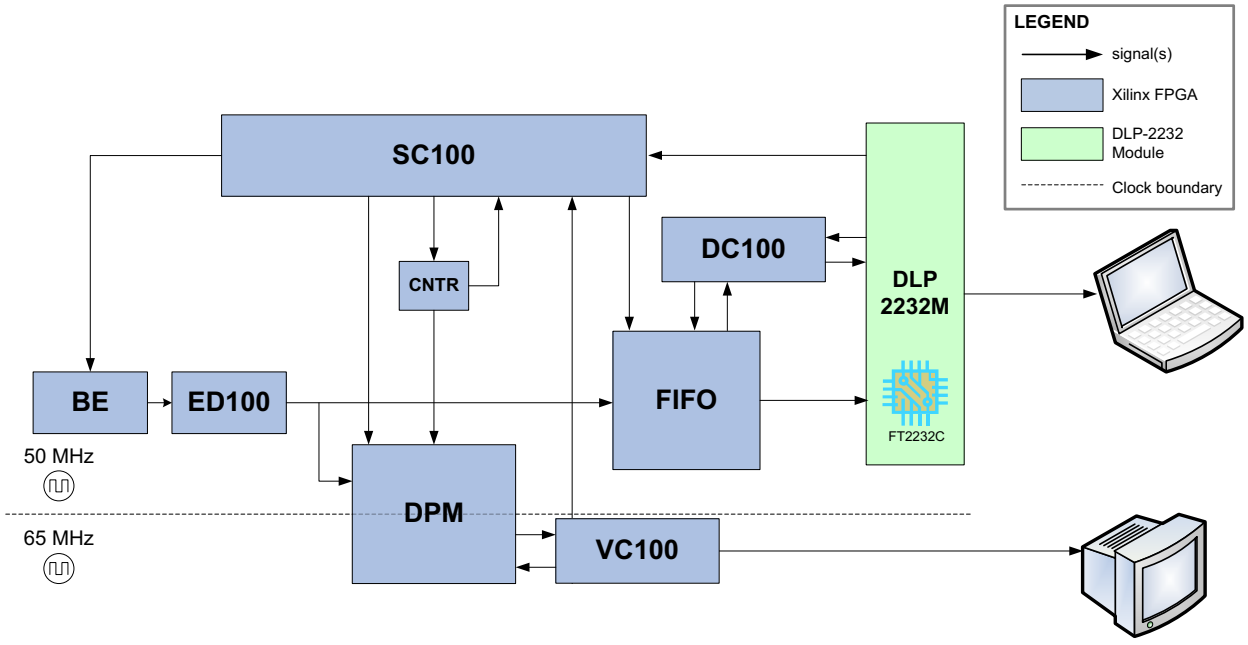

Fig. 10. Block diagram of the µAUS showing how its sub-components design map to hardware. The Xilinx FPGA implements the ultrasound backend (BE), envelope detector (ED100), dual-port memory (DPM), FIFO, VGA controller (VC100), data controller (DC100), system controller (SC100), and byte counter (CNTR). The DLP-2232M uses the FTDI2232C to support the USB interface.

X[7:0] (Dec)	X[7:0] (Bin)	Y[7:0] (Bin)	Y[7:0] (Dec)
255	**1**111 1111	**0**111 1111	127
10	**0**000 1010	**0**111 0101	117
127	**0**111 1111	**0**000 0000	98
64	**0**100 0000	**0**011 1111	32
128	**1**000 0000	**0**111 1111	0
195	**1**100 0011	**0**011 1100	67
85	**0**101 0101	**0**010 1010	42

Fig. 11. Table showing the rectification procedure; X represents the unrectified data received from the backend, and Y represents the result of rectification: Assign Y[7] (in **bold**) to 0, and assign the other 7 bits of Y, Y[6:0], based on the X[7] (in **bold**). If X[7] is 1 then we set Y[6:0] to X [6:0], otherwise assign Y[6:0] to the inverse of X[0:1].

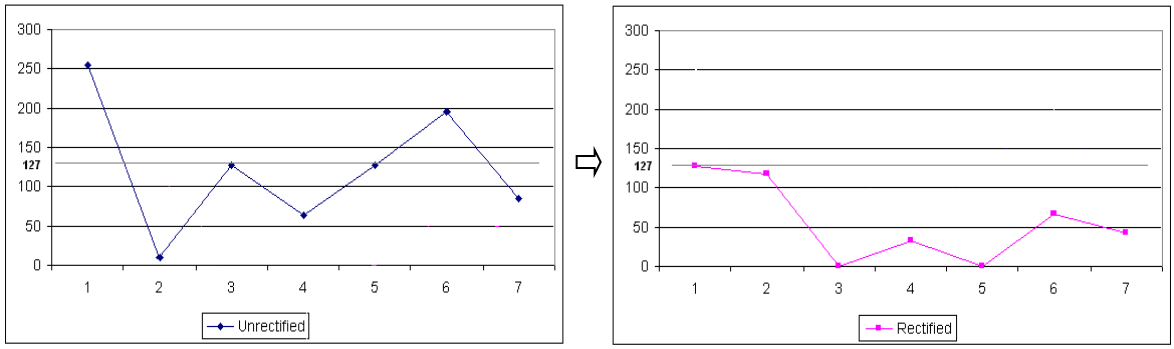

Fig. 12. Graph showing the rectification procedure applied to a series of value: the unrectified values ranging from 0-255 (shown on the left) map to a range of 0-128 (on the right).

$$y[n] = c_1*d_{n-1} + c_2*d_{n-2} + \ldots + c_{n-1}*d_{n-P-1} + c_n*d_{n-P}$$

Fig. 13. General difference equation for the FIR filter. The difference equation describes the relationship between the output signal with the input signal y[n] is the output signal, dn's are the input signals, cn's are the filter coefficients, P is the tap order.

$$y = c_1*(d_1+d_{23}) + c_2*(d_2+d_{22}) + \ldots + c_{12}*d_{12}$$

Fig. 14. Difference equation for the FIR filter in the µAUS system. y is the 8-bit output data, d1 to d23 are the 8-bit input data, c1 to c12 are the filter coefficients. We want to utilize the 12 built-in 18x18-bit multipliers on the Xilinx Spartan-3 board while achieving maximum tap order, thus we let d1 and d23 , d2 and d22 ... share the same coefficient respectively. This is a 23-tap filter.

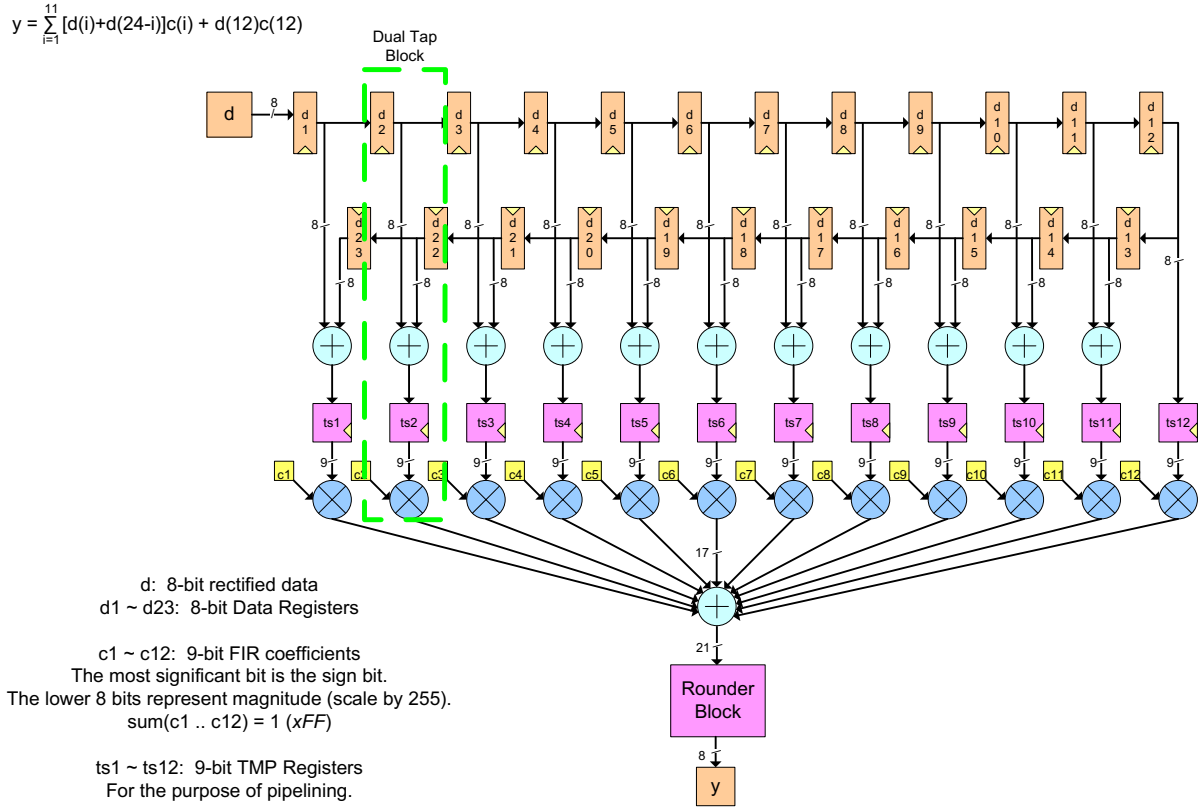

Fig. 15. Block Diagram showing FIR filter design

```
-- Add all the dual tap outputs (multiplier outputs) and save in register tmp_r1
final_add : adder_12_17 port map(t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,tmp_r1);

-- Divide tmp_r1 by 128 to adjust the coefficient values.
tmp_r2 <= tmp_r1(15 downto 8);

rounding:  process(tmp_r1, tmp_r2)
begin
   if tmp_r1(15) = '1' then   -- truncate all the negatives
          y <= (others => '0');
   elsif tmp_r1(7) = '1' then -- round .5
          y <= signed(tmp_r2) + 1;
   else
          y <= tmp_r2;
   end if;
end process rounding;
```

Fig. 16. VHDL implementation for the rounder block. This piece of code shows how the raw sum from the final adder is rounded to be an 8-bit filter output.

```
--------------------------------------------------------------------------
-- ed100                                                                --
-- File  : ed100.vhd                                                    --
--                                                                      --
-- Purpose:                                                             --
--  This module implements the envelope detector(ED).                   --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY ed100 is
       port (
              din  : IN std_logic_vector(7 downto 0);
              rst_l: IN std_logic;
              clk  : IN std_logic;
              dout : OUT std_logic_vector(7 downto 0) );
END ed100;


ARCHITECTURE mixed OF ed100 is

       component fir
       port (
              d              : in std_logic_vector(7 downto 0);  -- data
              y              : out std_logic_vector(7 downto 0); -- output
              clock, reset_l : in std_logic);
       end component;

       signal fir_din       : std_logic_vector (7 downto 0);

       -- rst_l sync signals
       signal rst_l_tmp     : std_logic;
       signal srst_l        : std_logic;

BEGIN
       -- synchronizes rst_l
       rst_sff: process( clk )
       begin
              if (clk = '1' and clk'event) then
                     rst_l_tmp  <= rst_l;
                     srst_l     <= rst_l_tmp;
              end if;
       end process rst_sff;

      -- rectification
       fir_din <= "0" & din(6 downto 0) when din(7) = '1'
                  else "0" & not din(6 downto 0);

       -- low pass filtering
       low_pass:  fir port map ( fir_din, dout, clk, srst_l );

END mixed ;
```

Fig. 17. VHDL implementation of the ED100 performing rectification (concurrent statement) and low pass filtering (component).

```
--------------------------------------------------------------------------
-- FIR
-- File  : fir.vhd                                                       --
--                                                                       --
-- Purpose:                                                              --
--  This module the FIR filter.                                          --
--------------------------------------------------------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity fir is
      port (
            d   : in std_logic_vector(7 downto 0);   -- data
            y   : out std_logic_vector(7 downto 0);   -- output
            clock, reset_l : in std_logic);
end fir;

architecture description OF fir is
   component reg_8
   port (
        input : in std_logic_vector(7 downto 0);
       output : out std_logic_vector(7 downto 0);
        reset_l : in std_logic;
        clk : in std_logic);
   end component;

   component dualtap
      port (
      data_a : in std_logic_vector(7 downto 0);
      data_b : in std_logic_vector(7 downto 0);
      coeff  : in std_logic_vector(7 downto 0);    -- FIR coefficient
      tapout : out std_logic_vector(16 downto 0);  -- tap output
      reset_l : in std_logic;
      clk : in std_logic);
   end component;

    component adder_12_17 is
       port (
            input1 : in std_logic_vector(16 downto 0);
            input2 : in std_logic_vector(16 downto 0);
            input3 : in std_logic_vector(16 downto 0);
            input4 : in std_logic_vector(16 downto 0);
            input5 : in std_logic_vector(16 downto 0);
            input6 : in std_logic_vector(16 downto 0);
            input7 : in std_logic_vector(16 downto 0);
            input8 : in std_logic_vector(16 downto 0);
            input9 : in std_logic_vector(16 downto 0);
            input10 : in std_logic_vector(16 downto 0);
            input11 : in std_logic_vector(16 downto 0);
            input12 : in std_logic_vector(16 downto 0);
            sum      : out std_logic_vector(20 downto 0)
       );
    end component;
```

```
	signal c1 : std_logic_vector(7 downto 0);	-- FIR coefficients
	signal c2 : std_logic_vector(7 downto 0);
	signal c3 : std_logic_vector(7 downto 0);
	signal c4 : std_logic_vector(7 downto 0);
	signal c5 : std_logic_vector(7 downto 0);
	signal c6 : std_logic_vector(7 downto 0);
	signal c7 : std_logic_vector(7 downto 0);
	signal c8 : std_logic_vector(7 downto 0);
	signal c9 : std_logic_vector(7 downto 0);
	signal c10 : std_logic_vector(7 downto 0);
	signal c11 : std_logic_vector(7 downto 0);
	signal c12 : std_logic_vector(7 downto 0);

	signal d1 : std_logic_vector(7 downto 0);	-- 23-Tap Input Data
	signal d2 : std_logic_vector(7 downto 0);
	signal d3 : std_logic_vector(7 downto 0);
	signal d4 : std_logic_vector(7 downto 0);
	signal d5 : std_logic_vector(7 downto 0);
	signal d6 : std_logic_vector(7 downto 0);
	signal d7 : std_logic_vector(7 downto 0);
	signal d8 : std_logic_vector(7 downto 0);
	signal d9 : std_logic_vector(7 downto 0);
	signal d10 : std_logic_vector(7 downto 0);
	signal d11 : std_logic_vector(7 downto 0);
	signal d12 : std_logic_vector(7 downto 0);
	signal d13 : std_logic_vector(7 downto 0);
	signal d14 : std_logic_vector(7 downto 0);
	signal d15 : std_logic_vector(7 downto 0);
	signal d16 : std_logic_vector(7 downto 0);
	signal d17 : std_logic_vector(7 downto 0);
	signal d18 : std_logic_vector(7 downto 0);
	signal d19 : std_logic_vector(7 downto 0);
	signal d20 : std_logic_vector(7 downto 0);
	signal d21 : std_logic_vector(7 downto 0);
	signal d22 : std_logic_vector(7 downto 0);
	signal d23 : std_logic_vector(7 downto 0);

	signal t1 : std_logic_vector(16 downto 0);	-- Multipiler Output
	signal t2 : std_logic_vector(16 downto 0);
	signal t3 : std_logic_vector(16 downto 0);
	signal t4 : std_logic_vector(16 downto 0);
	signal t5 : std_logic_vector(16 downto 0);
	signal t6 : std_logic_vector(16 downto 0);
	signal t7 : std_logic_vector(16 downto 0);
	signal t8 : std_logic_vector(16 downto 0);
	signal t9 : std_logic_vector(16 downto 0);
	signal t10 : std_logic_vector(16 downto 0);
	signal t11 : std_logic_vector(16 downto 0);
	signal t12 : std_logic_vector(16 downto 0);

	signal tmp_r1 : std_logic_vector(20 downto 0);
	signal tmp_r2 : std_logic_vector(7 downto 0);

begin
	load_d01: reg_8 port map(d, d1, reset_l, clock);
```

```
      load_d02: reg_8 port map(d1, d2, reset_l, clock);
      load_d03: reg_8 port map(d2, d3, reset_l, clock);
      load_d04: reg_8 port map(d3, d4, reset_l, clock);
      load_d05: reg_8 port map(d4, d5, reset_l, clock);
      load_d06: reg_8 port map(d5, d6, reset_l, clock);
      load_d07: reg_8 port map(d6, d7, reset_l, clock);
      load_d08: reg_8 port map(d7, d8, reset_l, clock);
      load_d09: reg_8 port map(d8, d9, reset_l, clock);
      load_d10: reg_8 port map(d9, d10, reset_l, clock);
      load_d11: reg_8 port map(d10, d11, reset_l, clock);
      load_d12: reg_8 port map(d11, d12, reset_l, clock);
      load_d13: reg_8 port map(d12, d13, reset_l, clock);
      load_d14: reg_8 port map(d13, d14, reset_l, clock);
      load_d15: reg_8 port map(d14, d15, reset_l, clock);
      load_d16: reg_8 port map(d15, d16, reset_l, clock);
      load_d17: reg_8 port map(d16, d17, reset_l, clock);
      load_d18: reg_8 port map(d17, d18, reset_l, clock);
      load_d19: reg_8 port map(d18, d19, reset_l, clock);
      load_d20: reg_8 port map(d19, d20, reset_l, clock);
      load_d21: reg_8 port map(d20, d21, reset_l, clock);
      load_d32: reg_8 port map(d21, d22, reset_l, clock);
      load_d23: reg_8 port map(d22, d23, reset_l, clock);

      -- dualtap(data_a, data_b, coeff, tapout, reset_l, clk)
      dualtap01: dualtap port map(d1, d23, c1, t1, reset_l, clock);
      dualtap02: dualtap port map(d2, d22, c2, t2, reset_l, clock);
      dualtap03: dualtap port map(d3, d21, c3, t3, reset_l, clock);
      dualtap04: dualtap port map(d4, d20, c4, t4, reset_l, clock);
      dualtap05: dualtap port map(d5, d19, c5, t5, reset_l, clock);
      dualtap06: dualtap port map(d6, d18, c6, t6, reset_l, clock);
      dualtap07: dualtap port map(d7, d17, c7, t7, reset_l, clock);
      dualtap08: dualtap port map(d8, d16, c8, t8, reset_l, clock);
      dualtap09: dualtap port map(d9, d15, c9, t9, reset_l, clock);
      dualtap10: dualtap port map(d10, d14, c10, t10, reset_l, clock);
      dualtap11: dualtap port map(d11, d13, c11, t11, reset_l, clock);
      dualtap12: dualtap port map(d12, (others => '0'), c12, t12, reset_l,
clock);

    final_add : adder_12_17 port map(t1, t2, t3, t4, t5, t6, t7, t8, t9, t10,
t11, t12, tmp_r1);
    tmp_r2 <= tmp_r1(15 downto 8);

    rounding:  process(tmp_r1, tmp_r2)
    begin
      if tmp_r1(15) = '1' then      -- truncate all the negatives
            y <= (others => '0');
      elsif tmp_r1(7) = '1' then    -- round .5
            y <= signed(tmp_r2) + 1;
      else
            y <= tmp_r2;
      end if;
    end process rounding;

    c1(7 downto 0) <= "11111111";   -- -01
    c2(7 downto 0) <= "11111110";   -- -02
```

```
    c3(7 downto 0) <= "11111111";   -- -01
    c4(7 downto 0) <= "00000000";   -- 00
    c5(7 downto 0) <= "00000011";   -- 03
    c6(7 downto 0) <= "00000111";   -- 07
    c7(7 downto 0) <= "00001100";   -- 0C
    c8(7 downto 0) <= "00010001";   -- 11
    c9(7 downto 0) <= "00010111";   -- 17
    c10(7 downto 0) <= "00011011";  -- 1B
    c11(7 downto 0) <= "00011110";  -- 1E
    c12(7 downto 0) <= "00011111";  -- 1F

end description;
```

Fig. 18. VHDL implementation of the FIR filter.

```
-------------------------------------------------------------------------
-- reg_8                                                                --
-- File  : reg_8.vhd                                                    --
--                                                                      --
-- Purpose:                                                             --
--  This is a generic 8-bit register.                                   --
-------------------------------------------------------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity reg_8 is
       port (
       input : in std_logic_vector(7 downto 0);
       output : out std_logic_vector(7 downto 0);
       reset_l : in std_logic;
       clk : in std_logic);
end reg_8;

architecture description of reg_8 is
  signal output_int : std_logic_vector(7 downto 0);
begin

  flipflop: process(clk)
  begin
    if clk'event and clk = '1' then
      if reset_l = '0' then
        output_int <= (others => '0');
      else
        output_int <= input;
      end if;
    end if;
  end process flipflop;

  output <= output_int;

end description;
```

Fig. 19. VHDL implementation of 8-bit register used in the FIR filter.

```
--------------------------------------------------------------------------
-- Dual tap                                                             --
-- File  : dualtap.vhd                                                  --
--                                                                      --
-- Purpose:                                                             --
--  This module the dual tap component used in the FIR filter.          --
--------------------------------------------------------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity dualtap is
       port (
       data_a : in std_logic_vector(7 downto 0);
       data_b : in std_logic_vector(7 downto 0);
       coeff  : in std_logic_vector(7 downto 0);    -- FIR coefficient
       tapout : out std_logic_vector(16 downto 0);  -- tap output
       reset_l : in std_logic;
       clk : in std_logic);
end dualtap;

architecture description of dualtap is

  component adder_8 is   -- 8-bit adder for d1+d23, d2+d22, ...
  port (
       input1 : in std_logic_vector(7 downto 0);
       input2 : in std_logic_vector(7 downto 0);
       sum       : out std_logic_vector(8 downto 0)
  );
  end component;

  component reg_9 is      -- Flip-Flop for pipelining
  port (
       input : in std_logic_vector(8 downto 0);
       output : out std_logic_vector(8 downto 0);
       reset_l : in std_logic;
       clk : in std_logic);
  end component;

  component mult_9 is      -- multiply the FIR coefficient with the data sum
  port (
       clk: in std_logic;
       a: in std_logic_VECTOR(8 downto 0);      -- input data
       b: in std_logic_VECTOR(7 downto 0);      -- coefficient
       p: out std_logic_VECTOR(16 downto 0));   -- product
  end component;

  signal sum_9   : std_logic_vector(8 downto 0);
  signal tmpsum  : std_logic_vector(8 downto 0);

begin
  add_data:   adder_8 port map(data_a, data_b, sum_9);
  pipeline:   reg_9   port map(sum_9, tmpsum, reset_l, clk);
  mult_coeff: mult_9  port map(clk, tmpsum, coeff, tapout);

end description;
```

Fig. 20. VHDL implementation of the dual tap component used in the FIR filter.

```
--------------------------------------------------------------------------
-- 8-bit adder                                                          --
-- File  : adder_8.vhd                                                  --
--                                                                      --
-- Purpose:                                                             --
--  This is an 8-bit adder.  The output is 9 bits.                      --
--------------------------------------------------------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity adder_8 is
      port (
            input1 : in std_logic_vector(7 downto 0);
            input2 : in std_logic_vector(7 downto 0);
            sum      : out std_logic_vector(8 downto 0)
      );
end adder_8;

ARCHITECTURE description OF adder_8 IS
      SIGNAL sum_int         : std_logic_vector(8 downto 0);
BEGIN
      adder: process(input1, input2)
      BEGIN
            sum_int <= unsigned("0"&input1) + unsigned("0"&input2);
      END PROCESS adder;

      sum <= sum_int;
END description;
```

Fig. 21. VHDL implementation of the 8-bit adder used in the FIR filter.

```
--------------------------------------------------------------------------
-- 9-bit register                                                       --
-- File  : reg_9.vhd                                                   --
--                                                                      --
-- Purpose:                                                             --
--  Ths is a generic 9-bit register to hold the sum output for  --
--  pipelining.                                                          --
--------------------------------------------------------------------------

LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity reg_9 is
      port (
      input : in std_logic_vector(8 downto 0);
      output : out std_logic_vector(8 downto 0);
      reset_l : in std_logic;
      clk : in std_logic);
end reg_9;
```

```
architecture description of reg_9 is
  signal output_int : std_logic_vector(8 downto 0);
begin

  flipflop: process(clk)
  begin
    if clk'event and clk = '1' then
      if reset_l = '0' then
        output_int <= (others => '0');
      else
        output_int <= input;
      end if;
    end if;
  end process flipflop;

  output <= output_int;

end description;
```

Fig. 22. VHDL implementation of a generic 9-bit register used in the FIR filter to hold the sum output for pipelining.

```
--------------------------------------------------------------------------
-- 8-bit adder                                                          --
-- File  : adder_8.vhd                                                  --
--                                                                      --
-- Purpose:                                                             --
--  This is an 17-bit adder.  It takes 12 17-bit inputs and outputs a  --
--  21-bit value.                                                       --
--------------------------------------------------------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity adder_12_17 is
      port (
            input1 : in std_logic_vector(16 downto 0);
            input2 : in std_logic_vector(16 downto 0);
            input3 : in std_logic_vector(16 downto 0);
            input4 : in std_logic_vector(16 downto 0);
            input5 : in std_logic_vector(16 downto 0);
            input6 : in std_logic_vector(16 downto 0);
            input7 : in std_logic_vector(16 downto 0);
            input8 : in std_logic_vector(16 downto 0);
            input9 : in std_logic_vector(16 downto 0);
            input10 : in std_logic_vector(16 downto 0);
            input11 : in std_logic_vector(16 downto 0);
            input12 : in std_logic_vector(16 downto 0);
            sum      : out std_logic_vector(20 downto 0)
      );
end adder_12_17;

architecture description OF adder_12_17 is
      signal sum_int   : std_logic_vector(20 DOWNTO 0);
begin
      adder: process(input1, input2, input3, input4, input5, input6,
         input7, input8, input9, input10, input11, input12)
```

```
        begin
                sum_int <=
                   unsigned("0000" & input1) +
                   unsigned("0000" & input2) +
                   unsigned("0000" & input3) +
                   unsigned("0000" & input4) +
                   unsigned("0000" & input5) +
                   unsigned("0000" & input6) +
                   unsigned("0000" & input7) +
                   unsigned("0000" & input8) +
                   unsigned("0000" & input9) +
                   unsigned("0000" & input10) +
                   unsigned("0000" & input11) +
                   unsigned("0000" & input12);
        end PROCESS adder;
        sum <= sum_int;
end description;
```

Fig. 23. VHDL implementation of a generic 17-bit adder used in the FIR filter. It takes 12 17-bit inputs and outputs a 21-bit value.

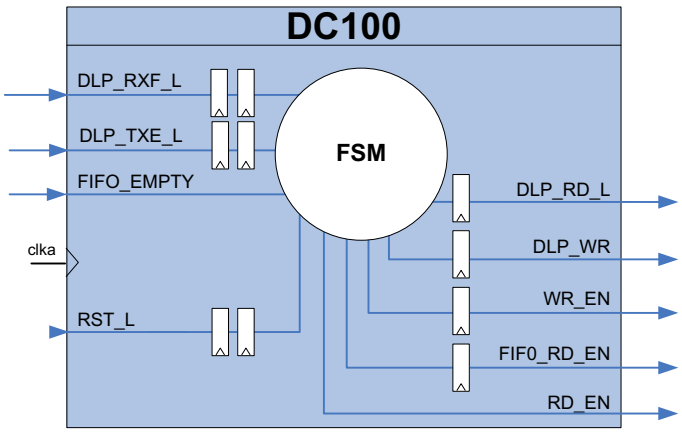

Fig. 24. Block diagram showing the inputs and output to the DC100 module. Back-to-back synchronizers stabilize the **DLP_RXF_L**, **DLP_TXE_L**, and **RST_L** inputs that are asynchronous the FSM's clock. An output synchronizer removes extra "wiggles" (due to a race condition) from the FSM's outputs: **DLP_RD_L**, **DLP_WR**, **WR_EN**, and **FIFO_RD_EN**.

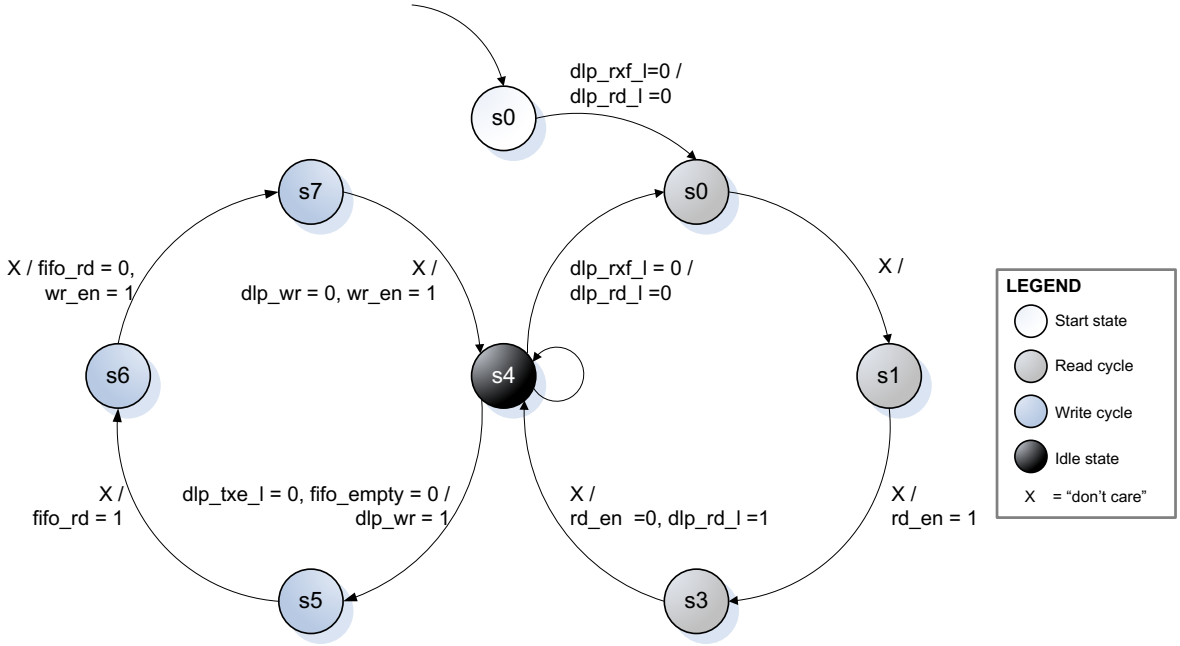

Fig. 25. State Diagram showing the DC100's read-(read/write) cycle and the timing protocol for the DLP's control interface. First, we read a byte from the DLP module when there is data available (**DLP_RXF_L** is low) by forcing **DLP_RD_L** low then high and enabling the flip-flop (**RD_EN=1**). Then if it can write (**DLP_TXE_L** is low) and there is data in the FIFO (**FIFO_EMPTY** is high), we repeatedly write data a byte at a time from the FIFO to the DLP by forcing **FIFO_RD_EN** and **DLP_WR** low then high, and enabling the tri-state buffer (**WR_EN=1**).

```
--------------------------------------------------------------------------
-- DC100                                                                --
-- File  : dc100.vhd                                                    --
--                                                                      --
-- Purpose:                                                             --
--  This module implements the Data Controller (DC) for interfacing     --
--  with the DLP-2232M and reading from the FIFO.                       --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dc100 is
       port (
              clk       : IN   std_logic;
              rst_l     : IN   std_logic;
              dlp_txe_l : IN   std_logic;
              dlp_rxf_l : IN   std_logic;
              fifo_empty: IN   std_logic;
              dlp_rd_l  : OUT std_logic;
              dlp_wr    : OUT std_logic;
              fifo_rd_en: OUT std_logic;
              wr_en     : OUT std_logic;
              rd_en     : OUT std_logic );
END dc100;
```

```
ARCHITECTURE behavioral of dc100 is

       type   states is ( init, read_0, read_1, read_2, idle,
                          write_0, write_1, write_2 ) ;
       signal state     : states := init ;
       signal nxt_state : states := init ;

       signal rxf_l_tmp     : std_logic;
       signal srxf_l        : std_logic;
       signal txe_l_tmp     : std_logic;
       signal stxe_l        : std_logic;
       signal dlp_rd_l_tmp  : std_logic;
       signal dlp_wr_tmp    : std_logic;
       signal fifo_rd_en_tmp: std_logic;
       signal wr_en_tmp     : std_logic;

BEGIN
       -- synchronizes dlp input, and removes "wiggles" on fsm outputs
       sff: process( clk )
       begin
              if ( clk = '1' and clk'event ) then
                     if ( rst_l = '0' ) then
                            rxf_l_tmp  <= '1';
                            srxf_l     <= '1';
                            txe_l_tmp  <= '1';
                            stxe_l     <= '1';
                            dlp_rd_l   <= '1';
                            dlp_wr     <= '0';
                            fifo_rd_en <= '0';
                            wr_en      <= '0';
                     else
                            rxf_l_tmp  <= dlp_rxf_l;
                             srxf_l     <= rxf_l_tmp;
                            txe_l_tmp  <= dlp_txe_l;
                            stxe_l     <= txe_l_tmp;
                            dlp_rd_l   <= dlp_rd_l_tmp;
                            dlp_wr     <= dlp_wr_tmp;
                            fifo_rd_en <= fifo_rd_en_tmp;
                            wr_en      <= wr_en_tmp;
                     end if;
              end if;
       end process sff;

       -- implements state register
       state_ff: process( clk )
       begin
              if ( clk = '1' and  clk'event ) then
                     if ( rst_l = '0' ) then
                            state <= init;
                     else
                           state <= nxt_state;
                     end if;
              end if;
       end process state_ff;

       -- determines the next state
       state_trans: process( stxe_l, srxf_l, fifo_empty, state )
       begin
              nxt_state <= state;
```

```
            case state is
                  when init   => if ( srxf_l = '0' ) then
                                       nxt_state <= read_0;
                                 end if;
                  when read_0 => nxt_state <= read_1;
                  when read_1 => nxt_state <= read_2;
                  when read_2 => nxt_state <= idle;
                  when idle   => if ( stxe_l = '0' and fifo_empty ='0' ) then
                                       nxt_state <= write_0;
                                 elsif ( srxf_l = '0' ) then
                                       nxt_state <= read_0;
                                 end if;
                  when write_0 => nxt_state <= write_1;
                  when write_1 => nxt_state <= write_2;
                  when write_2 => nxt_state <= idle;
            end case;
      end process state_trans;

      -- output values
      dlp_rd_l_tmp   <= '0' when ( (state = init and  srxf_l = '0') or
                                   (state = idle and  srxf_l = '0') or
                                  state = read_0 or state = read_1 ) else '1' ;
      dlp_wr_tmp     <= '1' when ( (state = idle and stxe_l= '0' and fifo_empty ='0')
                                   or state = write_0 or state = write_1 ) else '0';
      rd_en          <= '1' when ( state = read_1  ) else '0';
      fifo_rd_en_tmp <= '1' when ( state = write_0 ) else '0';
      wr_en_tmp      <= '1' when ( state = write_1 ) else '0';

END behavioral;
```

Fig. 26. VHDL code showing the two-process implementation of the DC100's FSM. The **READ_0**, **READ_1**, **READ_2** and the **WRITE_0**, **WRITE_1**, **WRITE_2** states generate a 60ns active pulse width for **DLP_RD_L** and **DLP_WR** output respectively. The timing constraints for the DLP module specify a min 50ns active pulse width for the **DLP_RD_L** and **DLP_WR** strobes.

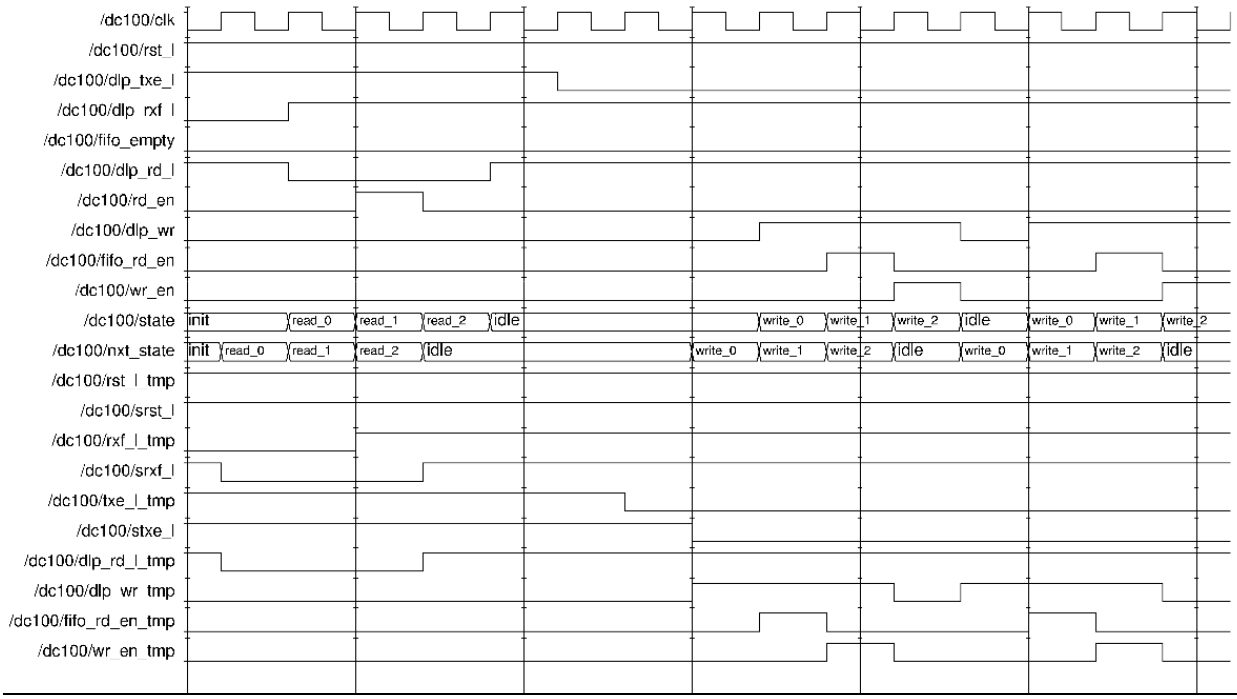

Fig. 27. Timing diagram verifying the operation the FSM for the DC100. The clock pulse width is 20ns (50MHz). For the read cycle: when the **DLP_RXF_L** signal goes low, the **DLP_RD_L** is active for 60ns and the **RD_EN** signal (flip-flop enable) is active for 20ns of the read cycle. For the write cycle: while the **DLP_TXE_L** is low, the **DLP_WR** signal is active for 60ns, and the **WR_EN** (tri-state buffer enable) is active for the last 20ns of the write cycle.

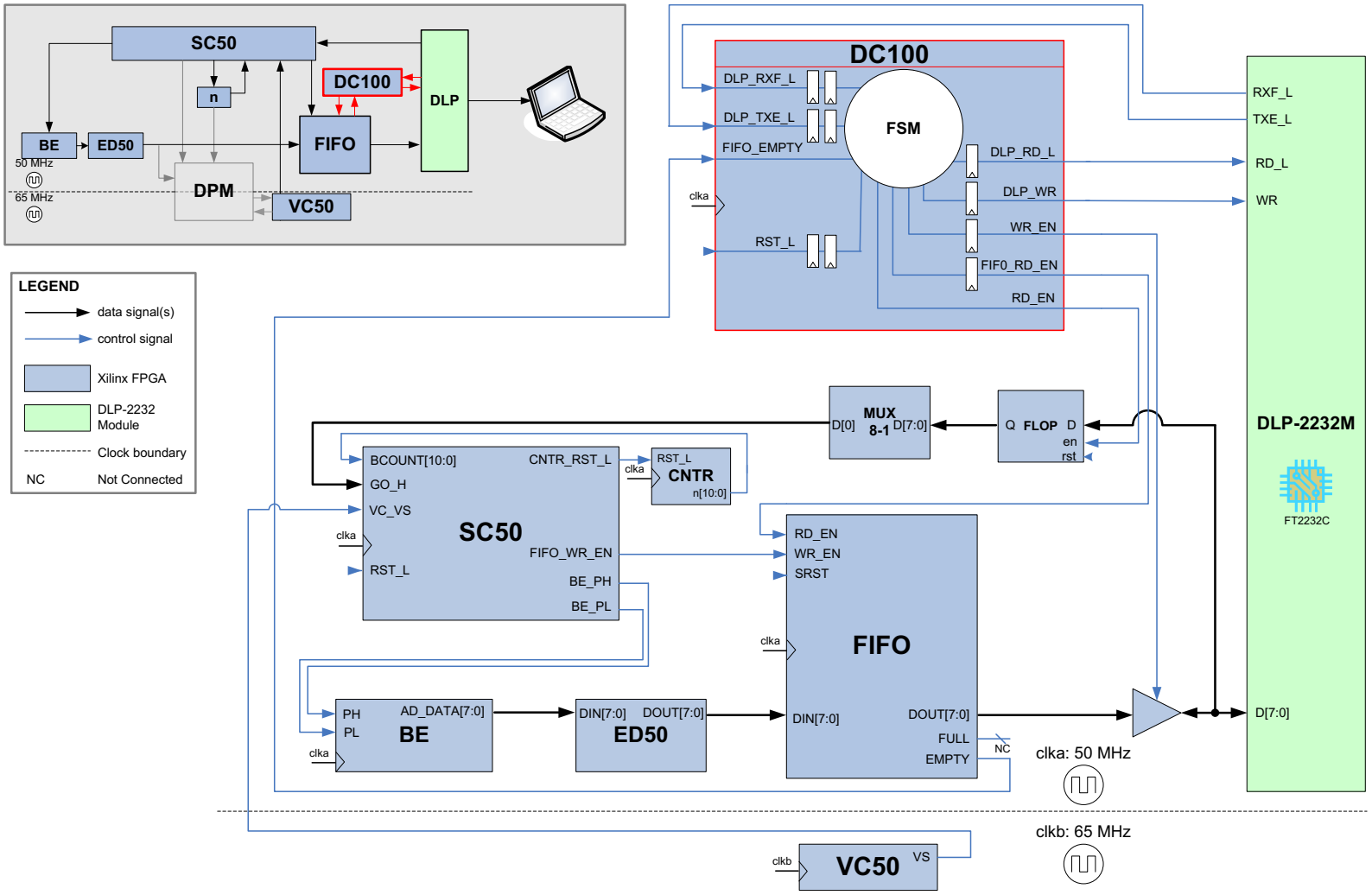

Fig. 28. Block diagram of the DC module and its test architecture. To simulate and synthesize the DC100, we implemented simplified versions of the ED (ED50), SC (SC50), and VC (VC50. The insert shows that DC100 interfaces with the FIFO and DLP.

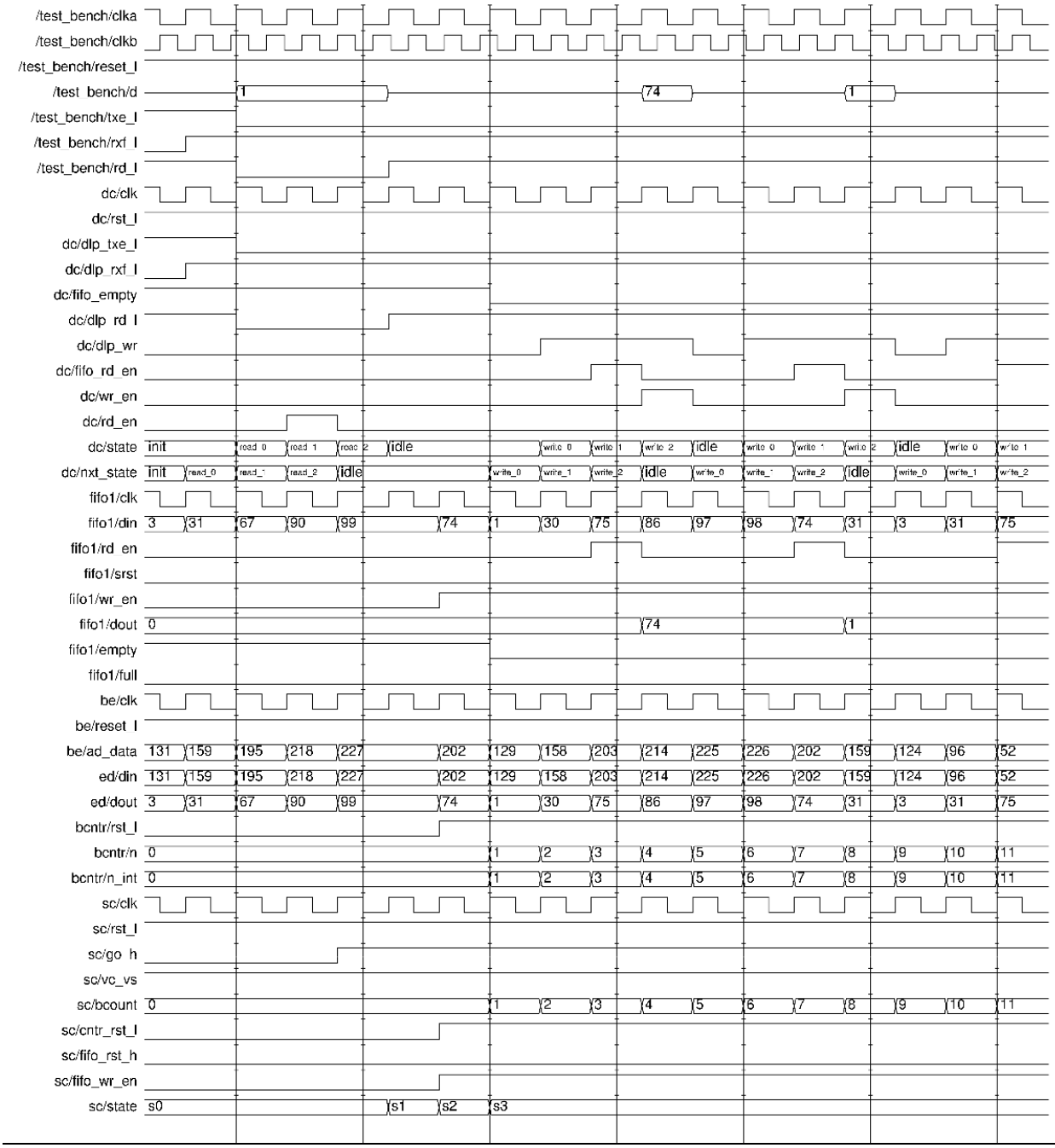

Fig. 29. Timing diagram verifying the operation of the DC100 within its test architecture: 1 is read from the DLP, and 74, 1 read from the FIFO and written to the DLP.

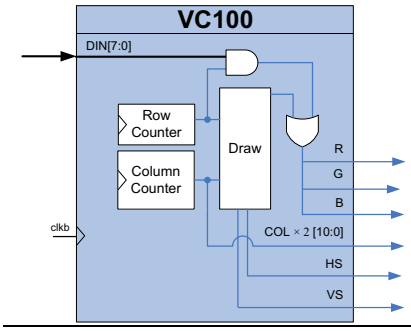

Fig. 30. Block diagram showing the inputs and output to the VC100 module. The DIN input comes from the DPM. R, G, B, HS, and VS go to the local VGA monitor. COL×2 supplies the read address for the 2048-byte DPM; the algorithm uses 1024 value of the 2048 values.

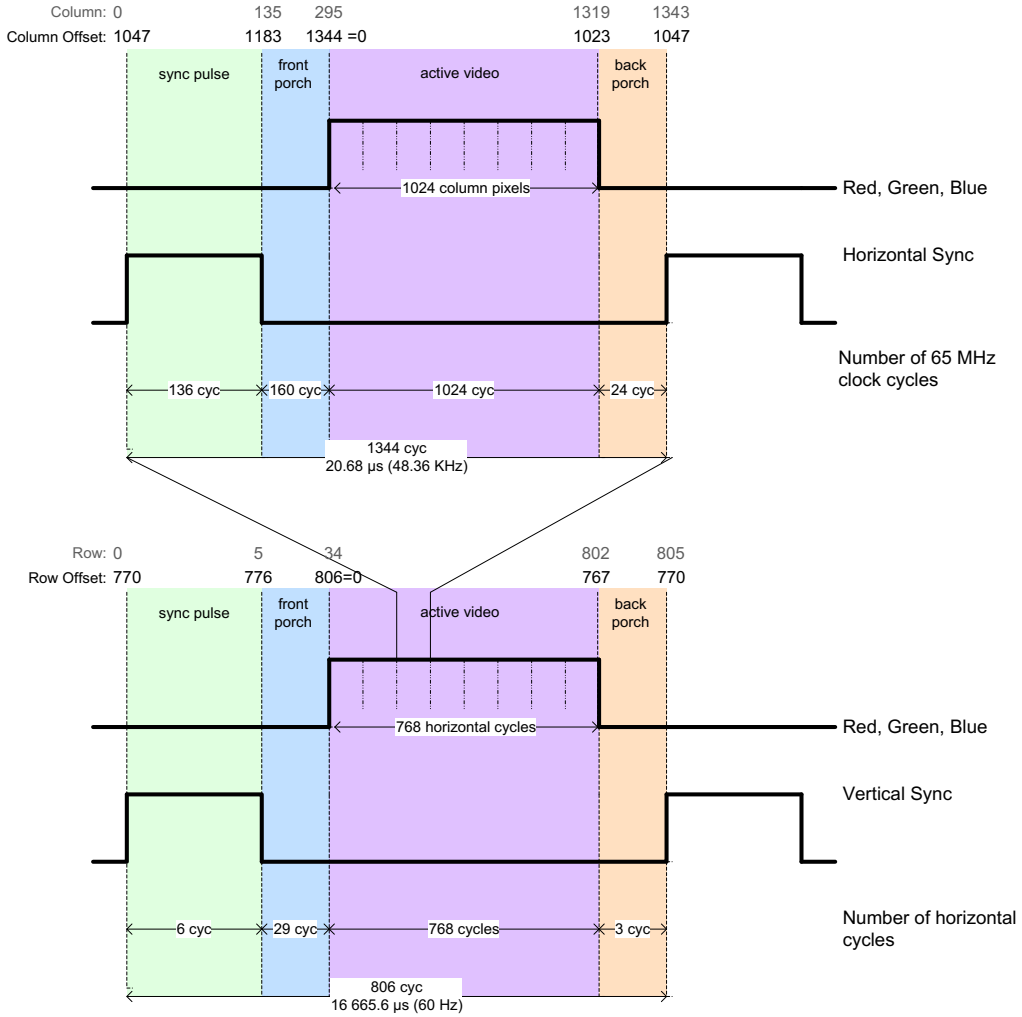

Fig. 31. Timing diagram showing the "standard" timing values for a 60 Hz, 1024 × 768 VGA and the offsetting of the horizontal (pixel) and vertical (line) counters to start the count (from 0) at the beginning of the active video region.

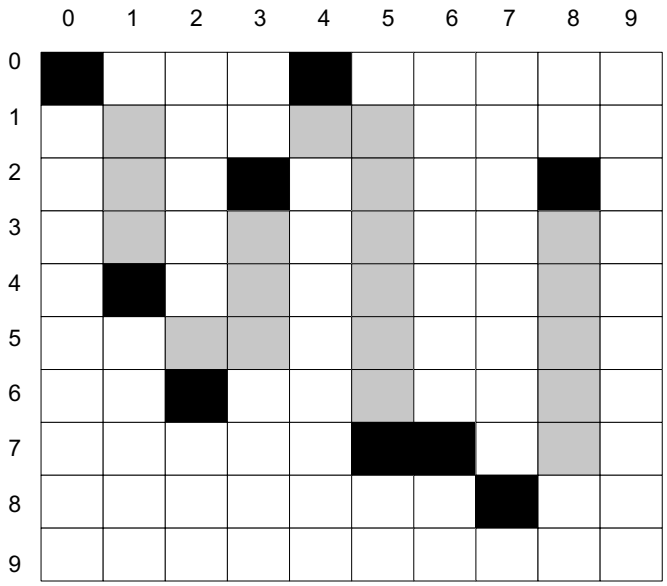

Fig. 32. Grid showing how the VC100's plotting algorithm works using three parameters: the current line number (line), the previous a-mode value (y0), and the current a-mode value (y1). The black squares (pixels) represent the input values, and the plotting algorithm fills in the gray squares (pixels).

```
-- case 0
if (y0 > y1)
  if ( y1 < line <= y0 )
     pixel_on <= '1';
  else
     pixel_on <= '0';

-- case 1
elsif (y0 < y1) then
  if ( y0 < line <= y1 )
     pixel_on <= '1';
  else
     pixel_on <= '0';

-- case 2
else    -- y0 = y1
  if ( line = y0 )
     pixel_on <= '1';
  else
     pixel_on <= '0';
```

Fig. 33. Pseudo-code explaining the plotting algorithm: by comparing the values (amplitudes) of two consecutive inputs, **`Y0`** and **`Y1`**, and the current line, the plotting algorithm determines whether to turn on/off the current pixel.

```
--------------------------------------------------------------------------
-- char_rom                                                             --
-- File  : char_rom.vhd                                                 --
--                                                                      --
-- Purpose:                                                             --
```

```
--  64 bit representations of 8x8 characters.                              --
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package char_rom is

       -- offset = active video + back porch - 1
       constant H_OFFSET  : std_logic_vector(10 downto 0):= "10000010111"; --  1047
       constant V_OFFSET  : std_logic_vector(9 downto 0) := "1100000010";  -- 770

       --Binary Character Values
       constant zero:  std_logic_vector(63 downto 0):=
"0001100000100100010000100100001001000010010000100010010000011000";
       constant one:   std_logic_vector(63 downto 0):=
"0001100000101000010010000000100000001000000010000000100000001000";
       constant two:   std_logic_vector(63 downto 0):=
"0011100001000100000001000000100000010000001000000110000011111100";
       constant three: std_logic_vector(63 downto 0):=
"0011100001000100000001000001100000000100000001000100100010000111000";
       constant four:  std_logic_vector(63 downto 0):=
"0000100000011000001010000100100001111110000010000000100000001000";
       constant five:  std_logic_vector(63 downto 0):=
"0011110000100000001000000011100000000100000001000010010000011000";
       constant six:   std_logic_vector(63 downto 0):=
"0001100000100100010000000101100001100100010001000100010000111000";
       constant seven: std_logic_vector(63 downto 0):=
"0011110000100100000001000000100000001000000010000000100000110000";
       constant eight: std_logic_vector(63 downto 0):=
"0001100000100100001001000001100000011000001001000010010000011000";
       constant nine:  std_logic_vector(63 downto 0):=
"0001100000100100001001000011110000000100000001000010010000011000";

end package char_rom;
```

Fig. 34. VHDL char_rom package that includes 64 bit binary representations decimal numbers used for 8 × 8 character display. A binary value of 1 indicates that the pixel at this position is on, and a 0 indicates that the pixel is off.

```
--------------------------------------------------------------------------------
-- VC100                                                                      --
-- File: vc100.vhd                                                          --
--                                                                            --
-- Purpose:  This module implements the VGA controller(VC).                   --
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;
use work.char_rom.all;

ENTITY vc100 is
     port(
     clk   : IN  std_logic;
     rst_l : IN  std_logic;
     din   : IN  std_logic_vector(7 downto 0);
     btn   : IN  std_logic_vector(3 downto 0);
     col   : OUT std_logic_vector(10 downto 0);
     hs    : OUT std_logic;
     vs    : OUT std_logic;
```

```
     r     : OUT std_logic;
     g     : OUT std_logic;
     b     : OUT std_logic);
END vc100;

ARCHITECTURE behavioral of vc100 is

       signal pixel : std_logic_vector(10 downto 0) := H_OFFSET;
       signal line  : std_logic_vector(9 downto 0)  := V_OFFSET;
       signal y0    : std_logic_vector(9 downto 0)  := "0000000000";
       signal y1    : std_logic_vector(9 downto 0)  := "0000000000";
       signal pixel_on  : std_logic; -- pixel on
       signal din_10: std_logic_vector(9 downto 0);

       signal leftCursor: std_logic_vector(10 downto 0):= "00000000100";
       signal rightCursor: std_logic_vector(10 downto 0):= "01111111100";
       signal clk1: std_logic;
       signal clkCount: std_logic_vector(19 downto 0);

       --Character Signals
       signal number1: integer:= 1;
       signal number2: integer:= 0;
       signal number3: integer:= 1;
       signal number4: integer:= 6;
       signal number: natural:= 0;
       signal index: natural;
       signal char_pixel: natural:= 63;

BEGIN

       vga: process( clk )

       -- Character Selection
       procedure characterSel is
       begin
               case number is
                        when 0 => pixel_on <= zero(index);
                        when 1 => pixel_on <= one(index);
                        when 2 => pixel_on <= two(index);
                        when 3 => pixel_on <= three(index);
                        when 4 => pixel_on <= four(index);
                        when 5 => pixel_on <= five(index);
                        when 6 => pixel_on <= six(index);
                        when 7 => pixel_on <= seven(index);
                        when 8 => pixel_on <= eight(index);
                        when 9 => pixel_on <= nine(index);
                        when others => pixel_on <= '0';
               end case;
       end procedure characterSel;

       --Pixel Selection for Character Display
       procedure pixelSel is
       begin
           if (pixel > 512 AND pixel < 521) then
                       index <= to_integer(unsigned(char_pixel - (pixel-513)));
                       number <=  number1;
                       characterSel;
           elsif (pixel > 522 AND pixel < 531) then
                       index <= to_integer(unsigned(char_pixel - (pixel-523)));
                       number <=  number2;
                       characterSel;
```

```
    elsif (pixel > 532 AND pixel < 541) then
                index <= to_integer(unsigned(char_pixel - (pixel-533)));
                number <=  number3;
                characterSel;
    elsif (pixel > 542 AND pixel < 551) then
                index <= to_integer(unsigned(char_pixel - (pixel-543)));
                number <=  number4;
                characterSel;
    else
                pixel_on <= '0';
    end if;
end procedure pixelSel;

begin
        if ( clk = '1' and clk'event ) then
                if ( rst_l = '0' ) then
                        hs <= '0';
                        vs <= '0';
                else
                   -- row and col counters, and hs/vs timing generators
                  if ( pixel = 1182) then    -- end hs pulse, start front porch
                        hs <= '0';
                        pixel <= pixel + 1 ;
                  elsif ( pixel = 1343 ) then  -- reset count: start active video
                        pixel <= "00000000000" ;
                   -- end back porch, start hs pulse (new line)
                  elsif ( pixel = 1046 ) then
                        hs <= '1' ;
                        pixel <= pixel + 1 ;
                        if ( line = 769 ) then    -- start vs pulse
                             vs <= '1' ;
                             line <= line + 1;
                        -- end hs pulse, start front porch
                        elsif ( line = 775 ) then
                             vs <= '0' ;
                             line <= line + 1 ;
                        -- reset count: start v active video
                        elsif ( line = 806 ) then
                             line <= "0000000000" ;
                        else
                             line <= line + 1 ;
                        end if ;
                  else
                        pixel <= pixel + 1 ;
                  end if ;
                end if;

                -- 2 * col output
                col <= SHL(pixel, x"01");

                --PIXEL ON OR OFF
                if ( pixel < 1024 and line < 768 ) then   -- h and v active video
                    --Left and Right Cursor
                    if (pixel = leftCursor AND line < 650 AND line > 100) then
                        pixel_on <= '1';
                    elsif (pixel=rightCursor AND line < 650 AND line > 100) then
                        pixel_on <= '1';
                    --Character Display
                    elsif (line >=50 AND line <= 57 AND pixel > 512 AND pixel
                                < 551) then
                        case line is
```

```
                                when "0000110010" => char_pixel <= 63;
                                when "0000110011" => char_pixel <= 55;
                                when "0000110100" => char_pixel <= 47;
                                when "0000110101" => char_pixel <= 39;
                                when "0000110110" => char_pixel <= 31;
                                when "0000110111" => char_pixel <= 23;
                                when "0000111000" => char_pixel <= 15;
                                when "0000111001" => char_pixel <= 7;
                                when others => char_pixel <= 63;
                             end case;
                                     pixelSel;
                             -- start drawing lines
                            elsif ( pixel = 0 ) then -- initialize
                                y0   <= din_10;
                                y1   <= din_10;
                                pixel_on <= '0';
                            else   -- light up the dots!
                                y0  <= y1;
                                y1  <= din_10;
                            if ( y0 > y1 ) then
                                if ( line <= y0 and line > y1 ) then
                                   pixel_on <= '1';
                                else
                                   pixel_on <= '0';
                                end if;
                            elsif ( y0 < y1 ) then
                                if (line <= y1 and line > y0) then
                                   pixel_on <= '1';
                                else
                                   pixel_on <= '0';
                                end if;
                            else   -- y0 = y1
                                if (line = y0) then
                                   pixel_on <= '1';
                                else
                                   pixel_on <= '0';
                                end if;
                            end if;
                        end if;
                         else -- line >=768 or pixel >= 1024
                              pixel_on <= '0';
                         end if;
                        --SLOWER CLOCK FOR THE CURSORS
                        if(clkCount >= "1111111111111111111") then
                                         clk1 <= not(clk1);
                                clkCount <= (others=> '0');
                        else
                                clkCount <= clkCount + "00000000000000000001";
                        end if;
                end if;
        end process vga;

        -- offset and translate
        din_10 <= "01" & (x"80" - din); -- 512 + (128-din);

        r   <= pixel_on;
        g   <= pixel_on;
        b   <= pixel_on;

--CURSOR IMPLEMENTATION
cursor:process (clk1)
```

```
  --Add to increment the distance between the Cursors.
  procedure add is
  begin
     if(number4 < 9) then
        number4 <= number4 + 1;
     else
        number4 <= 0;
        if(number3 < 9) then
           number3 <= number3 + 1;
        else
           number3 <= 0;
           if(number2 < 9) then
              number2 <= number2 + 1;
           else
              number2<= 0;
              number1 <= 1;
           end if;
        end if;
     end if;
  end procedure add;

  --Substract to deccrement the distance between the Cursors.
  procedure subtract is
  begin
     if(number4 > 0) then
        number4 <= number4 - 1;
     else
        number4 <= 9;
        if(number3 > 0) then
           number3 <= number3 - 1;
        else
           number3 <= 9;
           if(number2 > 0) then
              number2 <= number2 - 1;
           else
              number2 <= 9;
              number1 <= 0;
           end if;
        end if;
     end if;
  end procedure subtract;

begin
--Spartan 3 button press events.
if (clk1 = '1' AND clk1'event) then
     if btn(3) = '1' then
         if (leftCursor > 3) then
             leftCursor <= leftCursor - 1;
             add;
         end if;
     elsif btn(2) = '1' then
         if (leftCursor < rightCursor) then
             leftCursor <= leftCursor + 1;
             subtract;
         end if;
     elsif btn(1) = '1' then
         if (rightCursor > leftCursor) then
             rightCursor <= rightCursor - 1;
             subtract;
         end if;
     elsif btn(0) = '1' then
```

```
          if (rightCursor < 1021) then
              rightCursor <= rightCursor + 1;
              add;
          end if;
      end if;
   end if;
 end process cursor;
END behavioral;
```

Fig. 35. VHDL implement of the VC100.

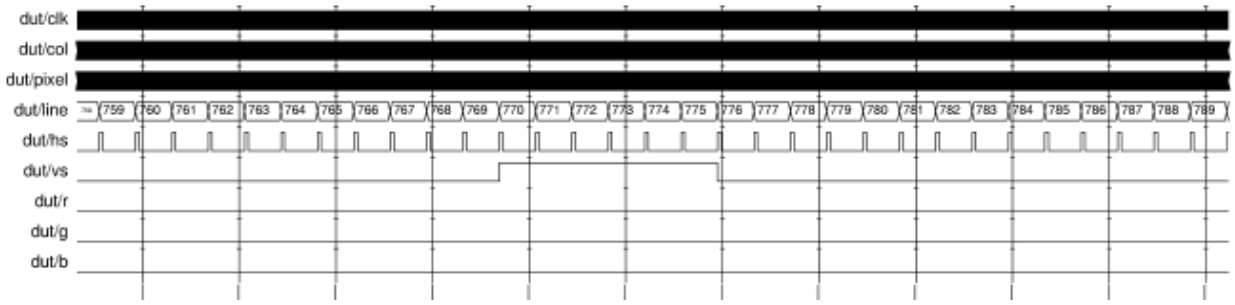

Fig. 36. Timing diagram verifying the operation of the VGA timing generator. The vertical sync signals (**VS**) rising and falling edge correspond to the rising and edges of the horizontal sync signal (**HS**); the **HS** frequency is 48KHz and the **VS** is 60Hz.

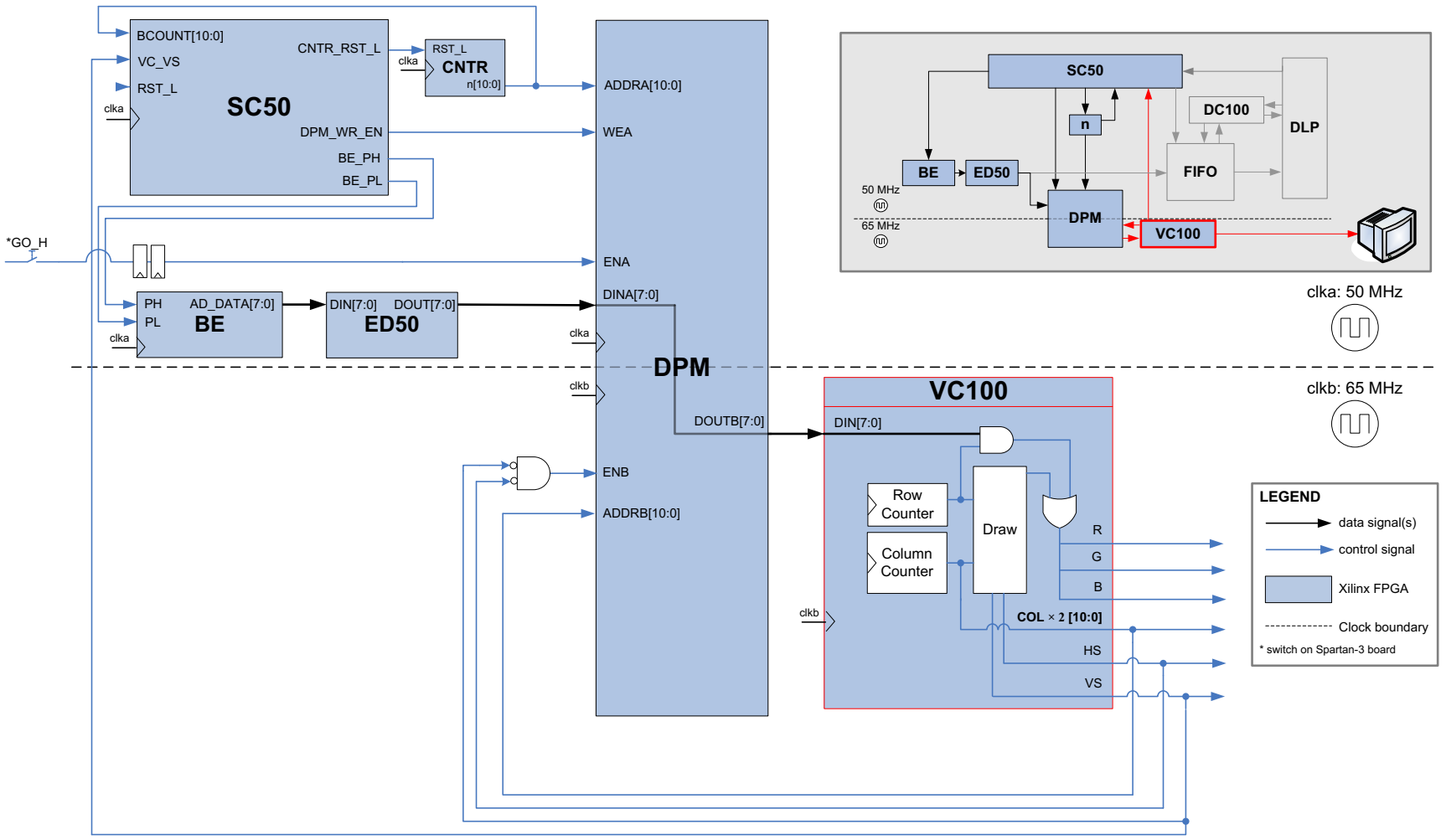

Fig. 37. Block diagram of the VC100 module and its test architecture. To simulate and synthesize the VC100, we implemented simplified versions of the ED (ED50), and SC (SC50). The insert shows that DC100 interfaces with the DPM, the SC50, and the VGA monitor.

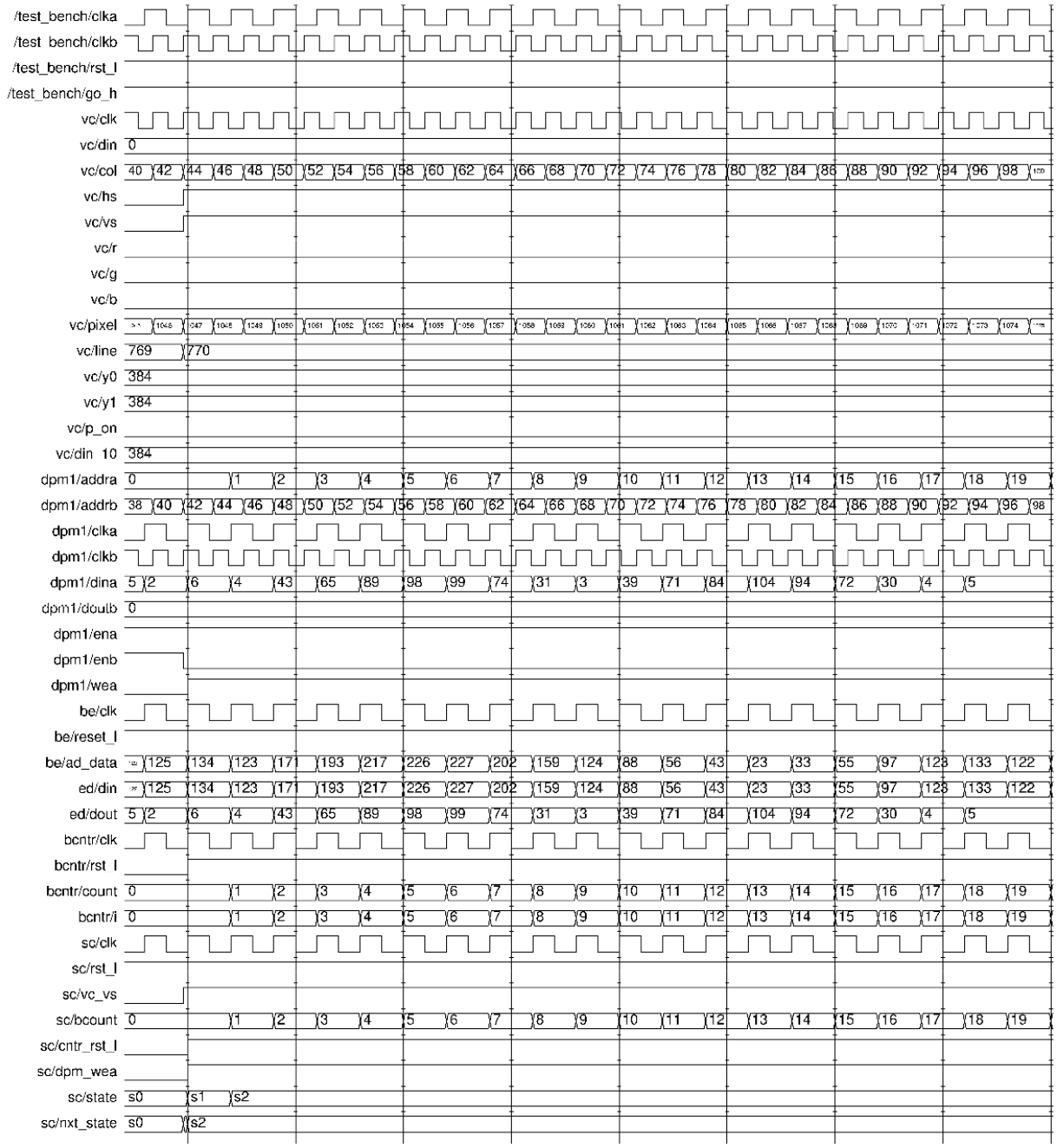

Fig. 38. Timing diagram verifying the operation of the VC100 in its test architecture: during the vertical retrace period of the VC (**`VC/VS = 1`**), the SC writes the sequence of bytes (**`DMP1/DINA`**) 6,4,43,65,89,98,99,74,31,3,39… to the DPM.

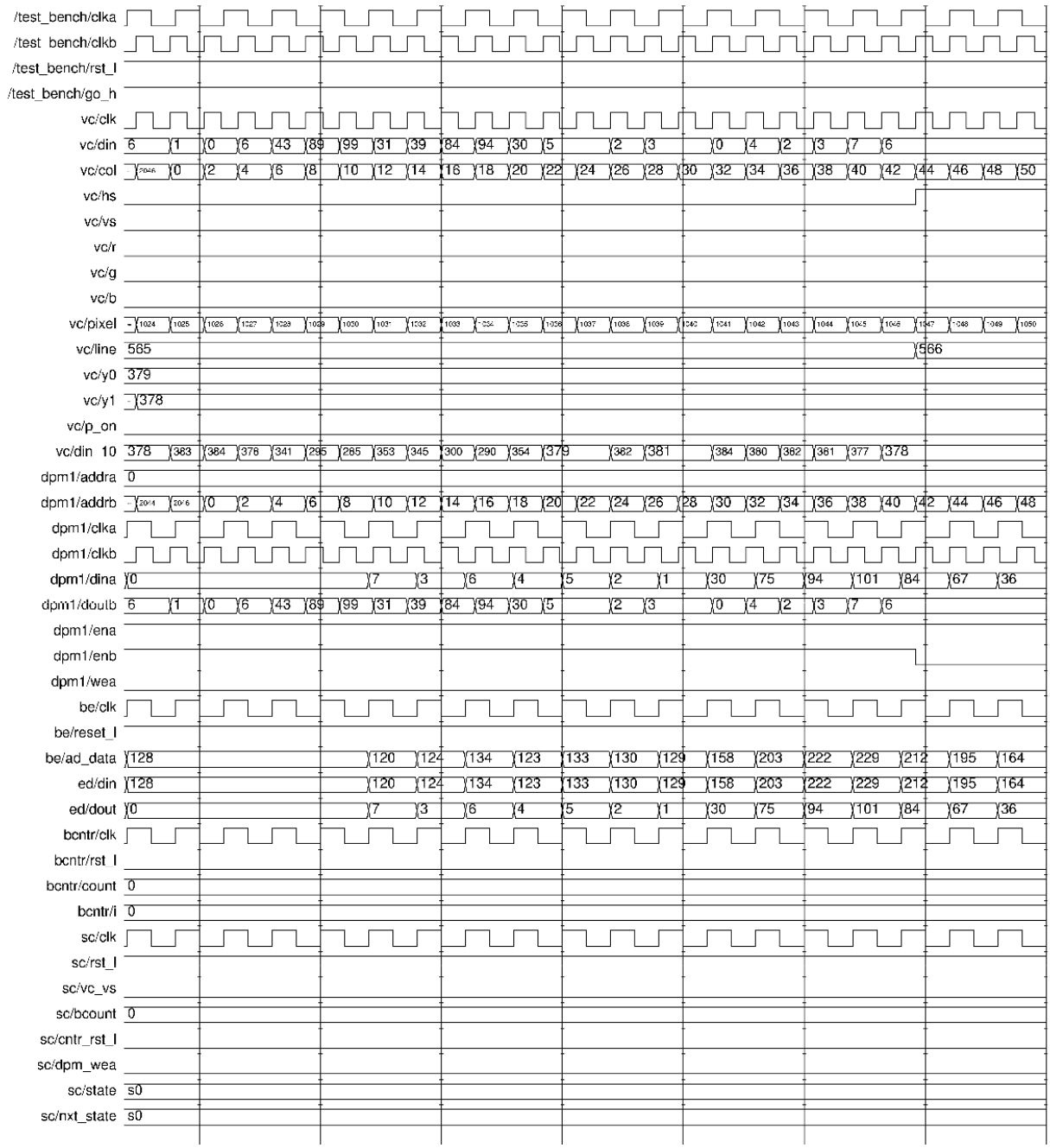

Fig. 39. Timing diagram verifying the operation of the VC100 in its test architecture: during the VC100's active video region, **`2 X COL`** provides the read address to the DPM (**`DPM1/ADDRB`**), which outputs (**`DMP1/DOUTB`**) bytes 6, 43, 89, 99,31,39…

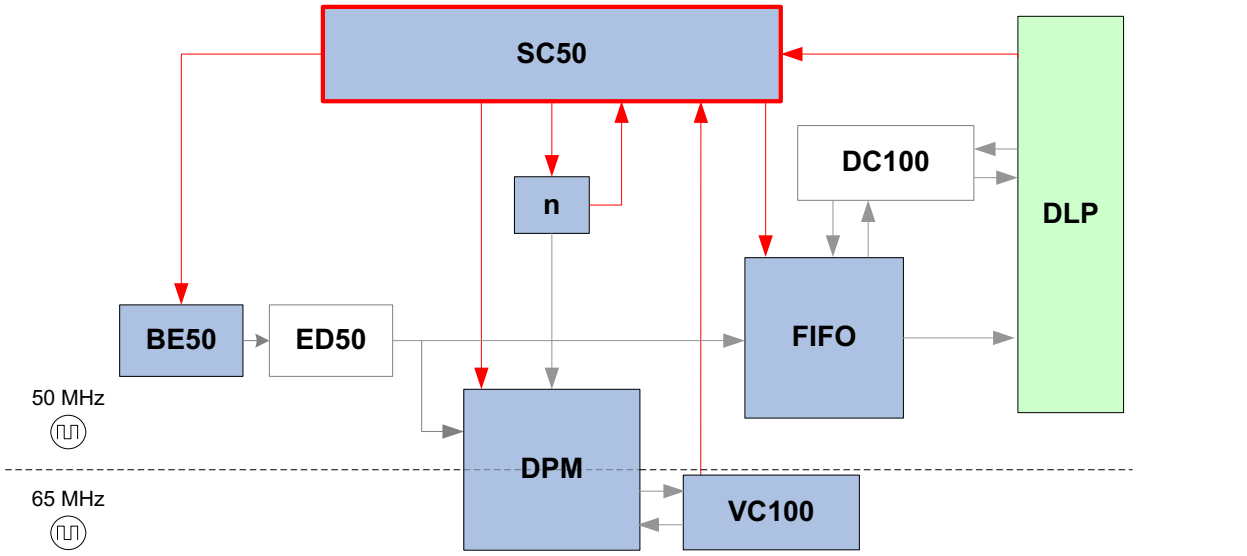

Fig. 40. Block diagram showing the SC100 interfaces with every component except the envelope detector and the DC100.

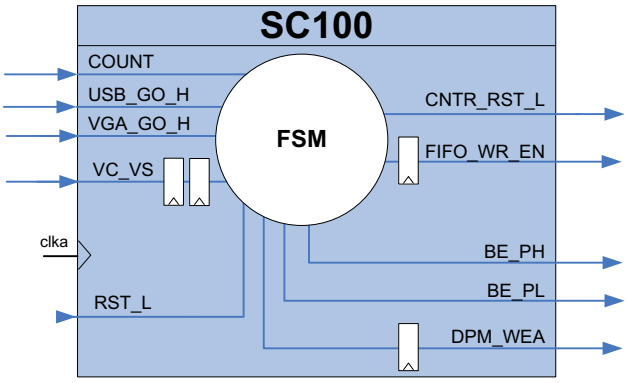

Fig. 41. Block diagram of SC100 showing its inputs and outputs used to control the acquisition and storage of data. Inputs to the controller include **COUNT** (from the byte counter), **USB_GO_H** (from the DLP module), **VGA_GO_H** (from a switch on the Spartan-3 board), and **VC_VS** (from the VC100). It outputs include **CNTR_RST_L** (to byte counter), **FIFO_WR_EN** (to the FIFO), **BE_PH** and **BE_PL** (to the backend), and **DPM_WEA** (to the dual port memory).

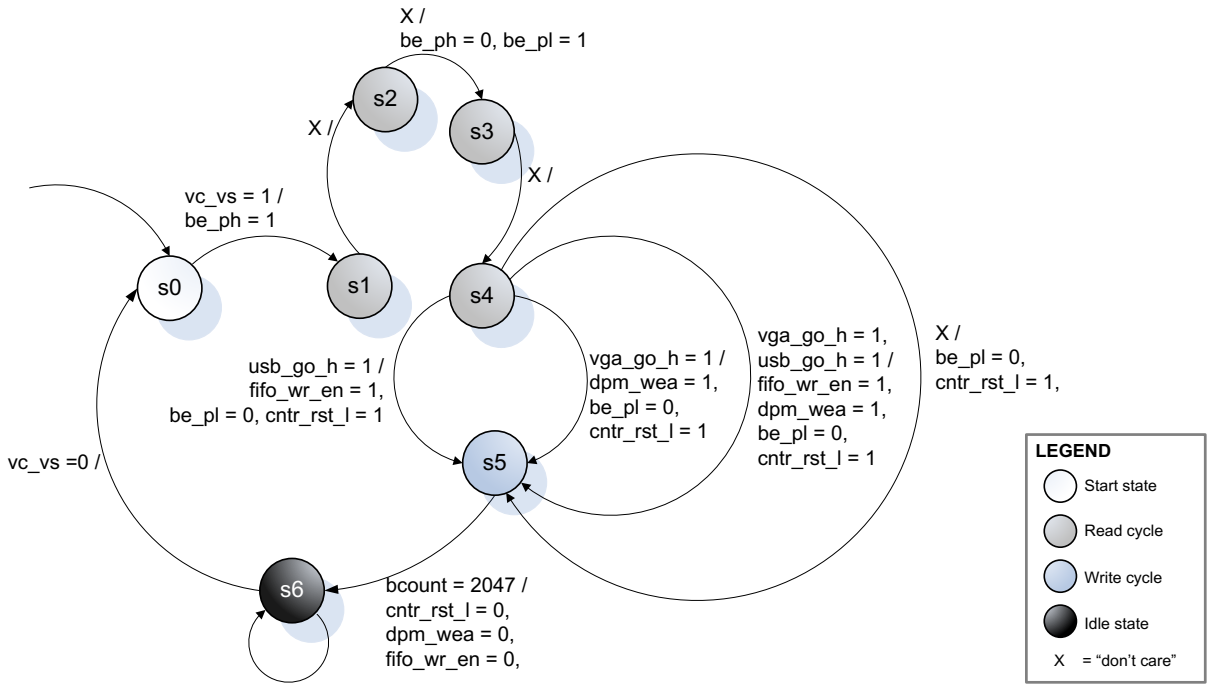

Fig. 42. State Diagram showing the SC100's read-write (pulse-enable) cycle. First, during the vertical retrace period (**VC_VS** = 1) of the VGA monitor, we pulse the backend (**BE_PH**, and **BE_PL**) to obtain a new A-mode vector. Then we have four possible transitions: (i) **USB_GO_H** is high, (ii) **VGA_GO_H** is high, (iii) both **USB_GO_H** and **VGA_GO_H** are high, and (iv) "don't care" transition. In all transitions, we reset the byte counter to zero. In transition (i) **USB_GO_H** is high, so we force the FIFO's write enable high (**FIFO_WR_EN**). In (ii) **VGA_GO_H** is high, so we enable the write site of the DPM **(DMP_WEA = 1**). In (iii) both **USB_GO_H** and **VGA_GO_H** are high, so we enable both FIFO_WR_EN and DMP_WEA. We take transition (iv), if both **VGA_GO_H** and USB_GO_H are low. Lastly, we disable the write-enable on both the FIFO and the DPM and wait until the next vertical retrace period to start over.

```
--------------------------------------------------------------------------
-- SC100                                                                --
-- File  : sc100.vhd                                                    --
--                                                                      --
-- Purpose:                                                             --
--  This module implements the system controller(SC).                   --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY sc100 is
       port (
              clk       : IN  std_logic;
              rst_l     : IN  std_logic;
              vga_go_h  : IN  std_logic;
              usb_go_h  : IN  std_logic;
```

```
            vc_vs     : IN  std_logic;
            count     : IN  std_logic_vector(10 downto 0);
            cntr_rst_l: OUT std_logic;
            fifo_wr_en: OUT std_logic;
            dpm_wea   : OUT std_logic;
            be_ph     : OUT std_logic;
            be_pl     : OUT std_logic );
END sc100;


ARCHITECTURE behavioral of sc100 is

      type   states is ( init, pulse_h_0, pulse_h_1, pulse_l_0, pulse_l_1, write_en,
idle );
      signal state     : states := init;
      signal nxt_state : states := init;

 signal vs_tmp        : std_logic;
      signal svs           : std_logic;
      signal dpm_wea_tmp   : std_logic;
      signal fifo_wr_en_tmp: std_logic;
BEGIN
      -- synchronizes vsync input, and removes "wiggles" on fsm outputs
      sff: process( clk )
      begin
            if ( clk = '1' and clk'event ) then
                  if ( rst_l = '0' ) then
                        vs_tmp     <= '0';
                        svs        <= '0';
                        dpm_wea    <= '0';
                        fifo_wr_en <= '0';
                  else
                        vs_tmp     <= vc_vs;
                        svs        <= vs_tmp;
                        dpm_wea    <= dpm_wea_tmp;
                        fifo_wr_en <= fifo_wr_en_tmp;
                  end if;
            end if;
      end process sff;

      -- implements state register
      state_flops: process( clk )
      begin
            if ( clk = '1' and  clk'event ) then
                  if ( rst_l = '0' ) then
                        state <= init;
            else
                  state <= nxt_state;
            end if;
      end if;
      end process state_flops ;

      -- determines the next state
      state_trans: process( svs, count, state )
      begin
            nxt_state <= state;
            case state is
                  when init =>  if ( svs = '1' ) then
                                                nxt_state <= pulse_h_0;
                                          end if;
                  when pulse_h_0 =>  nxt_state <= pulse_h_1;
```

```
                    when pulse_h_1 =>  nxt_state <= pulse_l_0;
                    when pulse_l_0 =>  nxt_state <= pulse_l_1;
                    when pulse_l_1 =>  nxt_state <= write_en;
                    when write_en  =>  if ( count = 2047 ) then
                                                  nxt_state <= idle;
                                       end if;
                    when idle =>  if ( svs = '0' ) then
                                                  nxt_state <= init;
                                       end if ;
            end case;
      end process state_trans;

      -- output values
      -- read
      be_ph     <= '1' when ( (state = init and svs = '1') or state = pulse_h_0 ) else
'0';
      be_pl     <= '1' when ( state = pulse_h_1 or state = pulse_l_0 ) else '0';
      -- write
      dpm_wea_tmp    <= '1' when ( vga_go_h = '1' and
                           (state = pulse_l_1 or state = write_en) ) else '0';
      fifo_wr_en_tmp <= '1' when ( usb_go_h = '1' and
                           (state = pulse_l_1 or state = write_en) ) else '0';
      cntr_rst_l     <= '1' when ( state = pulse_l_1 or state = write_en ) else '0';

END behavioral ;
```

Fig. 43. VHDL code showing the two-process implementation of the SC100's FSM. **BE_PH** and **BE_PL** both remain active for 40ns with an active **BE_PL** occurring after an active **BE_PH**. The SC100 forces **DPM_WEA** high if **VGA_GO**_H is high and drives **FIFO_WR_EN** high if **USB_GO_H** is high.

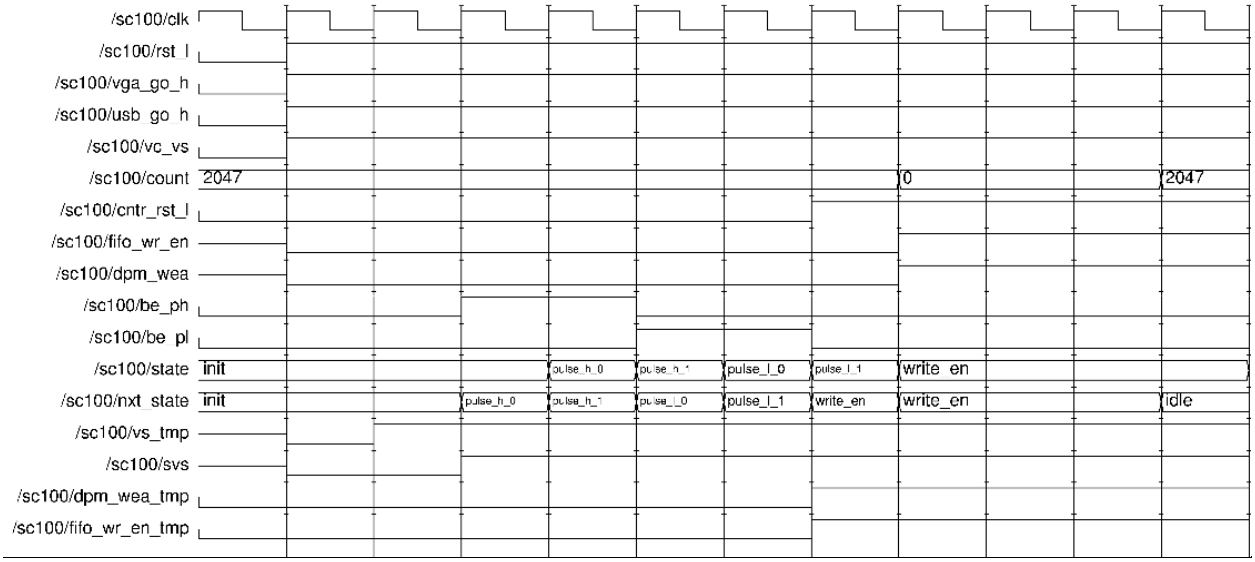

Fig. 44. Timing diagram verifying the operation of the SC100's FSM. The read-write (pulse-enable) cycle begins when **VC_VS** goes high and end when **COUNT = 2047**. **BE_PH** is active for 40ns and is followed by an active **BE_PL** for 40ns. Also, the DPM (**DMP_WEA**) and the FIFO (**FIFO_WR_EN**) are enabled in the **WRITE_EN** state since both **VGA_GO_H** and **USB_GO_H** are high respectively.

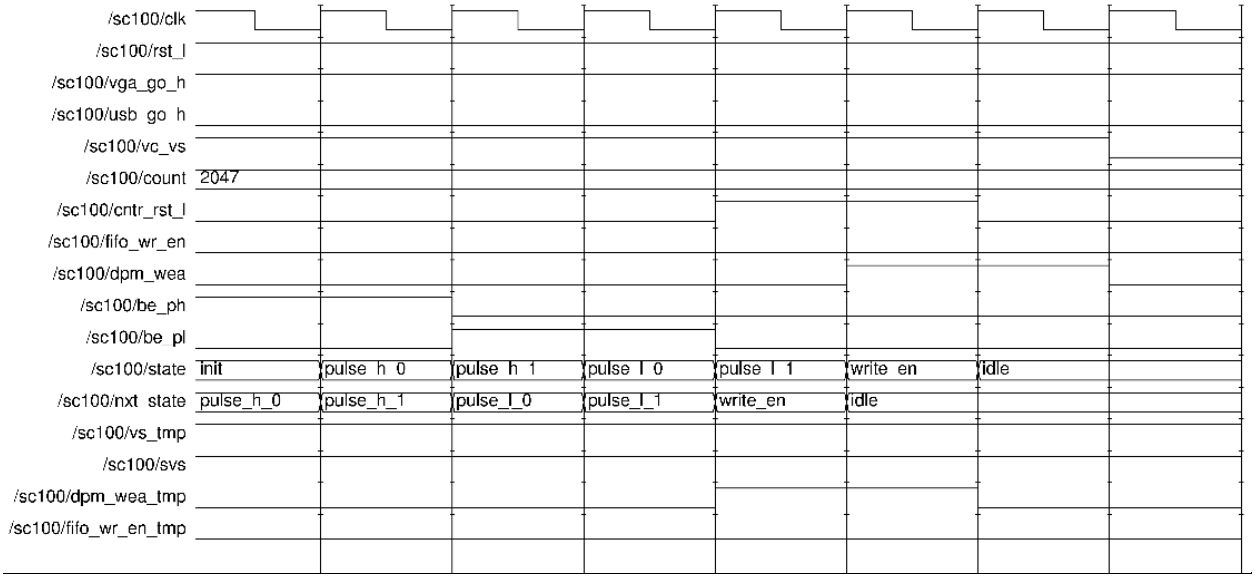

Fig. 45. Timing diagram verifying the operation of the SC100's FSM. In the **`WRITE_EN`** state, only the DPM (**`DMP_WEA`**) is enabled since **`VGA_GO_H`** is high and **`USB_GO_H`** is low.

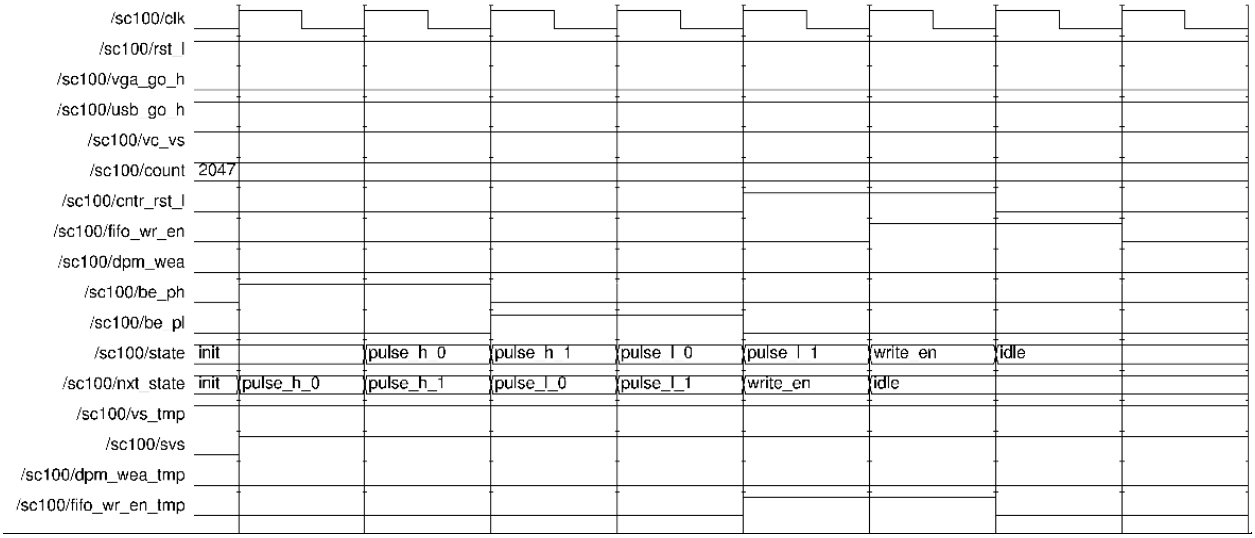

Fig. 46. Timing diagram verifying the operation of the SC100's FSM. In the **`WRITE_EN`** state, only the FIFO (**`FIFO_WR_EN`**) is enabled since **`USB_GO_H`** is high and **`VGA_GO_H`** is low.

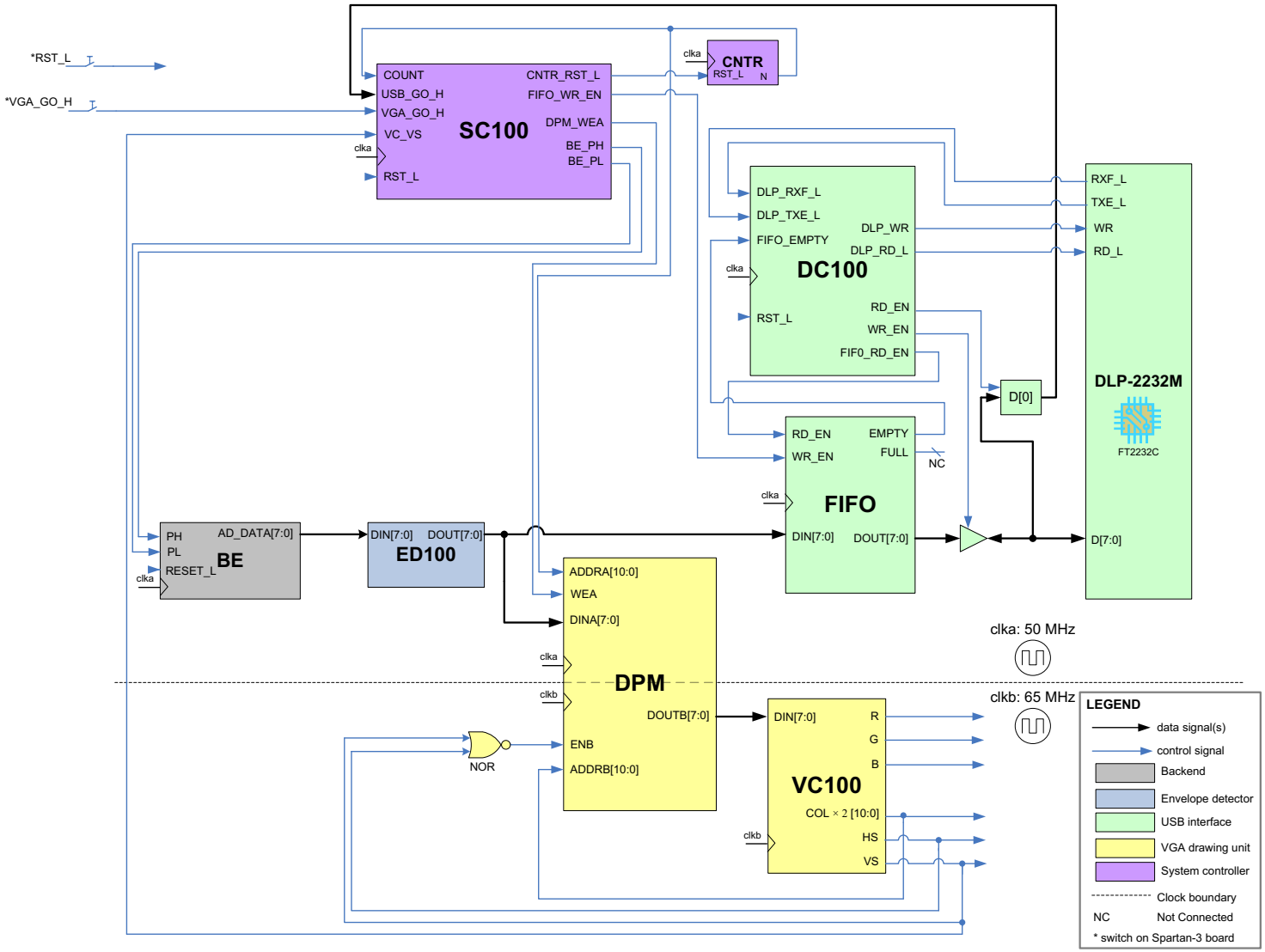

Fig. 47. Block diagram of the µAUS architecture, which is based on the SC100 and the test architectures of the V100 and DC100.

```
--------------------------------------------------------------------------
-- uAUS                                                                 --
-- File  : uaus.vhd                                                     --
--                                                                      --
-- Purpose:                                                             --
--  This module implements the uAUS.                                    --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY uaus is
       port (
              clka     : IN  std_logic;
              clkb     : IN  std_logic;
              rst_l    : IN  std_logic;
              vga_go_h : IN  std_logic;
              dlp_dbus : INOUT std_logic_vector(7 downto 0);
              dlp_txe_l: IN  std_logic;
              dlp_rxf_l: IN  std_logic;
              btn      : IN std_logic_vector(3 downto 0);
              dlp_rd_l : OUT std_logic;
              dlp_wr   : OUT std_logic;
              hs       : OUT std_logic;
              vs       : OUT std_logic;
              r        : OUT std_logic;
              g        : OUT std_logic;
              b        : OUT std_logic );
END uaus ;

ARCHITECTURE structural of uaus is

       -- system controller
       component sys_cntr is
       port (
              clk       : IN  std_logic;
              rst_l     : IN  std_logic;
              vga_go_h  : IN  std_logic;
              usb_go_h  : IN  std_logic;
              vc_vs     : IN  std_logic;
              bcount    : OUT std_logic_vector(10 downto 0);
              fifo_wea  : OUT std_logic;
              dpm_wea   : OUT std_logic;
              ph        : OUT std_logic;
              pl        : OUT std_logic );
       end component ;

       -- backend
       component bend
       port (
              clk    : IN  std_logic;
              rst_l  : IN  std_logic;
              ph     : IN  std_logic;
              pl     : IN  std_logic;
              ad_data: OUT std_logic_vector(7 downto 0) );
       end component;

       -- envelope detector
       component ed100
       port (
              din   : IN  std_logic_vector(7 downto 0);
              rst_l : IN std_logic;
              clk   : IN std_logic;
              dout  : OUT std_logic_vector(7 downto 0) );
```

```
	end component ;

	-- usb interface
	component usb_int is
	port (
		clk	: IN  std_logic;
		rst_l	: IN  std_logic;
		din	: IN  std_logic_vector(7 downto 0);
		wr_en	: IN  std_logic;
		dlp_dbus : INOUT std_logic_vector(7 downto 0);
		dlp_txe_l : IN  std_logic;
		dlp_rxf_l : IN  std_logic;
		dlp_rd_l  : OUT std_logic;
		dlp_wr	: OUT std_logic;
		d0	: OUT std_logic );
	end component ;

	-- vga drawing unit
	component vga_draw is
	port (
		clka   : IN  std_logic;
		clkb   : IN  std_logic;
		rst_l  : IN  std_logic;
		din    : IN  std_logic_vector(7 downto 0);
		wr_en  : IN  std_logic;
		wr_addr: IN  std_logic_vector(10 downto 0);
		btn    : IN std_logic_vector(3 downto 0);
		hs     : OUT std_logic;
		vs     : OUT std_logic;
		r      : OUT std_logic;
		g      : OUT std_logic;
		b      : OUT std_logic);
	end component ;

	-- byte counter output
	signal bcount    : std_logic_vector(10 downto 0);

	-- system controller outputs
	signal fifo_wea : std_logic;
	signal dpm_wea  : std_logic;
	signal be_ph    : std_logic;
	signal be_pl    : std_logic;

	-- backend outputs
	signal be_data  : std_logic_vector(7 downto 0);

	-- envelope detector outputs
	signal ed_dout  : std_logic_vector(7 downto 0);

	-- usb interface output
	signal usb_go_h   : std_logic;

	-- vga drawing unit output
	signal vga_vs : std_logic;

BEGIN

	system_controller:
		sys_cntr port map( clka, rst_l, vga_go_h, usb_go_h, vga_vs, bcount,
				   fifo_wea, dpm_wea, be_ph, be_pl );

	backend:
		bend port map ( clka, rst_l, be_ph, be_pl, be_data );
```

```
	envelope_detect:
		ed100 port map ( be_data, rst_l, clka, ed_dout );

	usb_interface:
		usb_int port map ( clka, rst_l, ed_dout, fifo_wea, dlp_dbus,
				   dlp_txe_l,dlp_rxf_l,dlp_rd_l,dlp_wr, usb_go_h );

	vga_drawing:
		vga_draw port map ( clka, clkb, rst_l, ed_dout, dpm_wea, bcount,
				    btn, hs, vga_vs, r, g, b );

	-- outputs
	vs <= vga_vs;

END structural;
```

Fig. 48. Structural VHDL describing the μAUS architecture.

```
--------------------------------------------------------------------------
-- system_controller                                                    --
-- File  : sys_cntr.vhd                                                 --
--                                                                      --
-- Purpose:                                                             --
--  This module implements the system controller for the uAUS.          --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sys_cntr is
	port (
		clk	: IN  std_logic;
		rst_l	: IN  std_logic;
		vga_go_h : IN  std_logic;
		usb_go_h : IN  std_logic;
		vc_vs	: IN  std_logic;
		bcount	: OUT std_logic_vector(10 downto 0);
		fifo_wea : OUT std_logic;
		dpm_wea  : OUT std_logic;
		ph	: OUT std_logic;
		pl	: OUT std_logic );
END sys_cntr;

ARCHITECTURE mixed of sys_cntr is

	-- system controller
	component sc100 is
	port (
		clk	: IN  std_logic;
		rst_l	: IN  std_logic;
		vga_go_h  : IN  std_logic;
		usb_go_h  : IN  std_logic;
		vc_vs	: IN  std_logic;
		count	: IN  std_logic_vector(10 downto 0);
		cntr_rst_l: OUT std_logic;
		fifo_wr_en: OUT std_logic;
		dpm_wea	: OUT std_logic;
		be_ph	: OUT std_logic;
		be_pl	: OUT std_logic );
	end component ;

	-- byte counter
```

```
	component cntr is
	port (
		clk  : IN  std_logic;
		rst_l: IN  std_logic;
		n    : OUT std_logic_vector(10 downto 0) );
	end component ;

	-- byte counter output
	signal cntr_n     : std_logic_vector(10 downto 0);

	-- system controller outputs
	signal sc_cntr_rst_l: std_logic;

	-- rst_l sync signals
	signal rst_l_tmp     : std_logic;
	signal srst_l        : std_logic;

BEGIN
	-- system controller
	sc: sc100
		port map (
			clk       => clk,
			rst_l     => srst_l,
			vga_go_h  => vga_go_h,
			usb_go_h  => usb_go_h,
			vc_vs     => vc_vs,
			count     => cntr_n,
			cntr_rst_l=> sc_cntr_rst_l,
			fifo_wr_en=> fifo_wea,  -- to FIFO
			dpm_wea   => dpm_wea,   -- to dual-port memory
			be_ph     => ph,        -- backend pulse-high
			be_pl     => pl );

	-- byte counter
	bcntr: cntr
		port map (
			clk   => clk,
			rst_l => sc_cntr_rst_l,
			n     => cntr_n);

	-- outputs
	bcount <= cntr_n;

	-- synchronizes rst_l
	rst_sff: process( clk )
	begin
		if (clk = '1' and clk'event) then
			rst_l_tmp  <= rst_l;
			srst_l     <= rst_l_tmp;
		end if;
	end process rst_sff;

END mixed;
```

Fig. 49. VHDL describing the μAUS system controller.

```
--------------------------------------------------------------------
-- CNTR                                                           --
-- File  : cntr.vhd                                               --
-- Purpose: This module implements a counter.                     --
--------------------------------------------------------------------
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY cntr is
      port (
            clk   : IN   std_logic;
            rst_l : IN  std_logic;
            n : OUT std_logic_vector(10 downto 0) );
END cntr;


ARCHITECTURE behavioral of cntr is

      signal n_int : std_logic_vector(10 downto 0);

BEGIN

      cnt: process( clk )
      begin
            if ( clk = '1' and clk'event ) then
                  if ( rst_l = '0' ) then
                        n_int <= "00000000000";
                  else
                        n_int <= n_int + 1;
                  end if;
            end if ;
      end process cnt ;

      n <= n_int;
END behavioral ;
```

Fig. 50. VHDL describing the byte counter used in the μAUS system controller.

```
--------------------------------------------------------------------------
-- usb_interface                                                        --
-- File  : usb_int.vhd                                                  --
--                                                                      --
-- Purpose:                                                             --
--  This module implements the usb interface.                           --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY usb_int is
      port (
            clk      : IN  std_logic;
            rst_l    : IN  std_logic;
            din      : IN  std_logic_vector(7 downto 0);
            wr_en    : IN  std_logic;
            dlp_dbus : INOUT std_logic_vector(7 downto 0);
            dlp_txe_l: IN  std_logic;
            dlp_rxf_l: IN  std_logic;
            dlp_rd_l : OUT std_logic;
            dlp_wr   : OUT std_logic;
            d0       : OUT std_logic );
END usb_int;
```

```
ARCHITECTURE mixed of usb_int is

	-- data controller
	component dc100 is
	port (
		clk	: IN  std_logic;
		rst_l	: IN  std_logic;
		dlp_txe_l : IN  std_logic;
		dlp_rxf_l : IN  std_logic;
		fifo_empty: IN  std_logic;
		dlp_rd_l  : OUT std_logic;
		dlp_wr    : OUT std_logic;
		fifo_rd_en: OUT std_logic;
		wr_en	: OUT std_logic;
		rd_en	: OUT std_logic );
	end component ;


	-- fifo
	component fifo
	port (
		clk  : IN std_logic;
		din  : IN std_logic_vector(7 downto 0);
		rd_en: IN std_logic;
		rst  : IN std_logic;
		wr_en: IN std_logic;
		dout : OUT std_logic_vector(7 downto 0);
		empty: OUT std_logic;
		full : OUT std_logic );
	end component;

	-- data controller outputs
	signal dc_wr_en	: std_logic;
	signal dc_rd_en	: std_logic;
	signal dc_fifo_rd_en: std_logic;

	-- fifo outputs
	signal fifo_dout  : std_logic_vector(7 downto 0) ;
	signal fifo_empty : std_logic;
	signal fifo_full  : std_logic;  -- not connected

	-- rst_l sync signals
	signal rst_l_tmp	: std_logic;
	signal srst_l		: std_logic;
	signal srst_h		: std_logic;

BEGIN
	-- data controller
	dc: dc100
		port map(
			clk		=> clk,
			rst_l		=> srst_l,
			dlp_txe_l	=> dlp_txe_l,
			dlp_rxf_l	=> dlp_rxf_l,
			fifo_empty	=> fifo_empty,
			dlp_rd_l	=> dlp_rd_l,
			dlp_wr		=> dlp_wr,
			fifo_rd_en	=> dc_fifo_rd_en,
			wr_en		=> dc_wr_en,
			rd_en		=> dc_rd_en ) ;

	-- FIFO
```

```
	fifo1: fifo
		port map (
			clk   => clk,
			din   => din,  -- from envelope detector
			rd_en => dc_fifo_rd_en,
			rst   => srst_h,
			wr_en => wr_en, -- from system controller
			dout  => fifo_dout,
			empty => fifo_empty,
			full  => fifo_full );

	-- synchronizes rst_l
	rst_sff: process( clk )
	begin
		if (clk = '1' and clk'event) then
			rst_l_tmp  <= rst_l;
			srst_l     <= rst_l_tmp;
			srst_h     <= not rst_l_tmp;

		end if;
	end process rst_sff;

	-- store byte lsb of byte read from dlp_dbus
	ff: process( clk )
	begin
		if ( clk = '1' and clk'event ) then
			if ( rst_l = '0' ) then
				d0  <= '0';
			elsif ( dc_rd_en ='1' ) then
				d0 <= dlp_dbus(0);
			end if ;
		end if ;
	end process ff;

	-- tri-state buffer
	dlp_dbus <=  fifo_dout when dc_wr_en = '1' else (others => 'Z');

END mixed;
```

Fig. 51. VHDL describing the μAUS USB interface.

```
-------------------------------------------------------------------------
-- vga_draw                                                            --
-- File  : vga_draw.vhd                                                --
--                                                                     --
-- Purpose:                                                            --
--  This module implements the VGA drawing unit.                       --
-------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY vga_draw is
	port (
		clka     : IN  std_logic;
		clkb     : IN  std_logic;
		rst_l    : IN  std_logic;
		din      : IN  std_logic_vector(7 downto 0);
		wr_en    : IN  std_logic;
		wr_addr  : IN  std_logic_vector(10 downto 0);
		btn       : IN  std_logic_vector(3 downto 0);
		hs       : OUT std_logic;
```

```
            vs       : OUT std_logic;
            r        : OUT std_logic;
            g        : OUT std_logic;
            b        : OUT std_logic );
END vga_draw;

ARCHITECTURE mixed of vga_draw is

      -- vga controller
      component vc100 is
      port (
            clk   : IN  std_logic;
            rst_l : IN  std_logic;
            din   : IN  std_logic_vector(7 downto 0);
            btn    : IN std_logic_vector(3 downto 0);
            col   : OUT std_logic_vector(10 downto 0);
            hs    : OUT std_logic;
            vs    : OUT std_logic;
            r     : OUT std_logic;
            g     : OUT std_logic;
            b     : OUT std_logic);
      end component ;

      -- dual port memory
      component dpm
      port (
            addra: IN std_logic_vector(10 downto 0);
            addrb: IN std_logic_vector(10 downto 0);
            clka : IN std_logic;
            clkb : IN std_logic;
            dina : IN std_logic_vector(7 downto 0);
            doutb: OUT std_logic_vector(7 downto 0);
            enb  : IN std_logic;
            wea  : IN std_logic);
      end component;

      -- vga controller outputs
      signal vc_vs      : std_logic;
      signal vc_hs      : std_logic;
      signal vc_col     : std_logic_vector(10 downto 0);

      -- dpm signals
      signal dpm_enb    : std_logic;
      signal dpm_doutb  : std_logic_vector(7 downto 0);

      -- rst_l sync signals
      signal rst_l_tmp     : std_logic;
      signal srst_l        : std_logic;

BEGIN

      -- vga controller
      vc: vc100
            port map(
                  clk  => clkb,
                  rst_l=> srst_l,
                  din  => dpm_doutb,
                  btn  => btn,
                  col  => vc_col,
                  hs   => vc_hs,
                  vs   => vc_vs,
                  r    => r,
                  g    => g,
```

```
                    b    => b );

      -- dual-port memory
      dpm1: dpm
            port map (
                  addra => wr_addr,  -- from byte counter
                  addrb => vc_col,
                  clka  => clka,
                  clkb  => clkb,
                  dina  => din,
                  doutb => dpm_doutb,
                  enb   => dpm_enb,
                  wea   => wr_en ); -- from system controller

      -- synchronizes rst_l
      rst_sff: process( clkb )
      begin
            if (clkb = '1' and clkb'event) then
                  rst_l_tmp  <= rst_l;
                  srst_l     <= rst_l_tmp;
            end if;
      end process rst_sff;

      -- dual-port memory read enable
      dpm_enb <= vc_hs nor vc_vs;

      -- outputs
      vs <= vc_vs;
      hs <= vc_hs;

END mixed;
```

Fig. 52. VHDL describing the μAUS VGA drawing unit.

```
--------------------------------------------------------------------------
-- backend wrapper                                                      --
-- File  : bend.vhd                                                     --
--------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bend is
      port (
            clk     : IN  std_logic;
            rst_l   : IN  std_logic;
            ph      : IN  std_logic;
            pl      : IN  std_logic;
            ad_data : OUT std_logic_vector(7 downto 0) );
END bend;

ARCHITECTURE mixed of bend is

      -- backend
      component backend is
      port (
            clk     : IN  std_logic;
            reset_l : IN  std_logic;
            ph      : IN  std_logic;
            pl      : IN  std_logic;
            ad_data : OUT std_logic_vector(7 downto 0) );
      end component ;
```

```
	-- rst_l sync signals
	signal rst_l_tmp     : std_logic;
	signal srst_l        : std_logic;

BEGIN

	be: backend port map ( clk, srst_l, ph, pl, ad_data );

	-- synchronizes rst_l
	rst_sff: process( clk )
	begin
		if (clk = '1' and clk'event) then
			rst_l_tmp  <= rst_l;
			srst_l     <= rst_l_tmp;
		end if;
	end process rst_sff;

END mixed ;
```

Fig. 53. VHDL wrapper for the simulated backend.

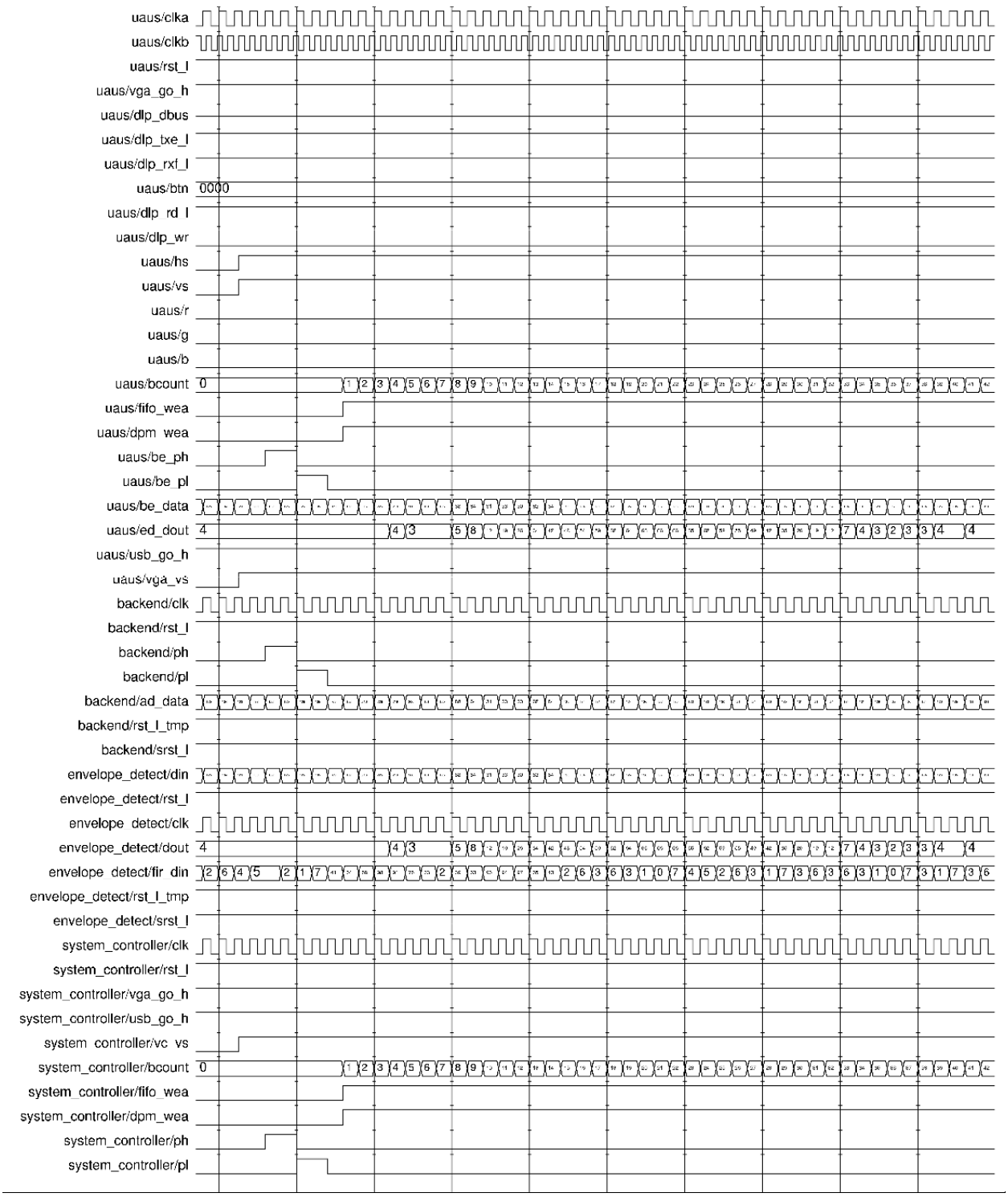

Fig. 54. Timing diagram verifying the operation of µAUS system controller. During the vertical retrace period (**`UAUS/VGA_VS=1`**), the system controller pulses the backend to acquire a new A-mode vector, and enables the FIFO (**`SYSTEM_CONTROLLER/FIFO_WEA=1`**), and the DPM (**`SYSTEM_CONTROLLER/DMP_WEA=1`**).

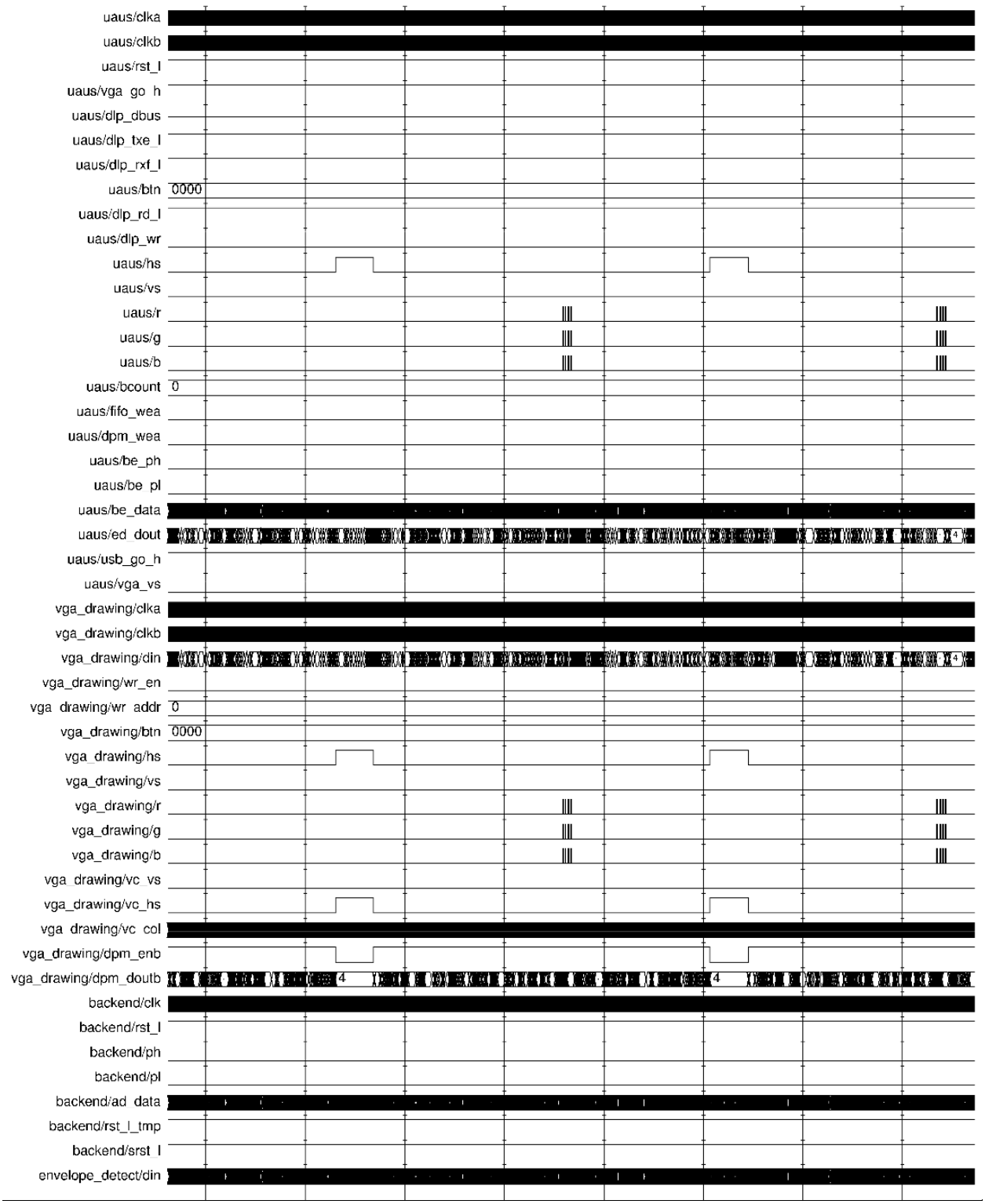

Fig. 55. Timing diagram verifying the operation of µAUS VGA drawing unit. Roughly during the active video region of the VGA monitor, the VGA drawing unit (**`VGA_DRAW`**) reads data from the DPM.

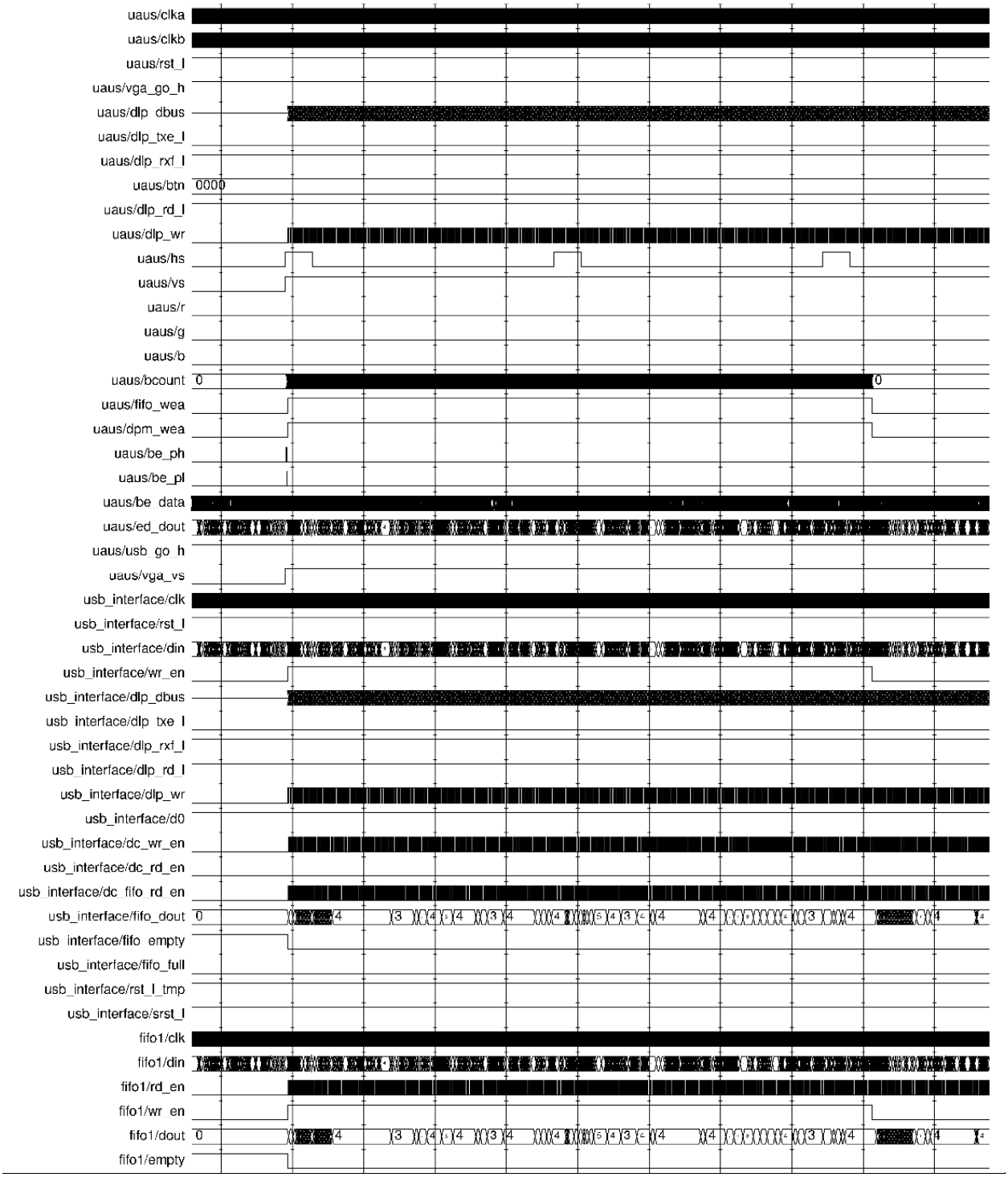

Fig. 56. Timing diagram verifying the operation of µAUS's USB interface. During the vertical retrace period, the USB interface writes data from the FIFO to the DLP module.

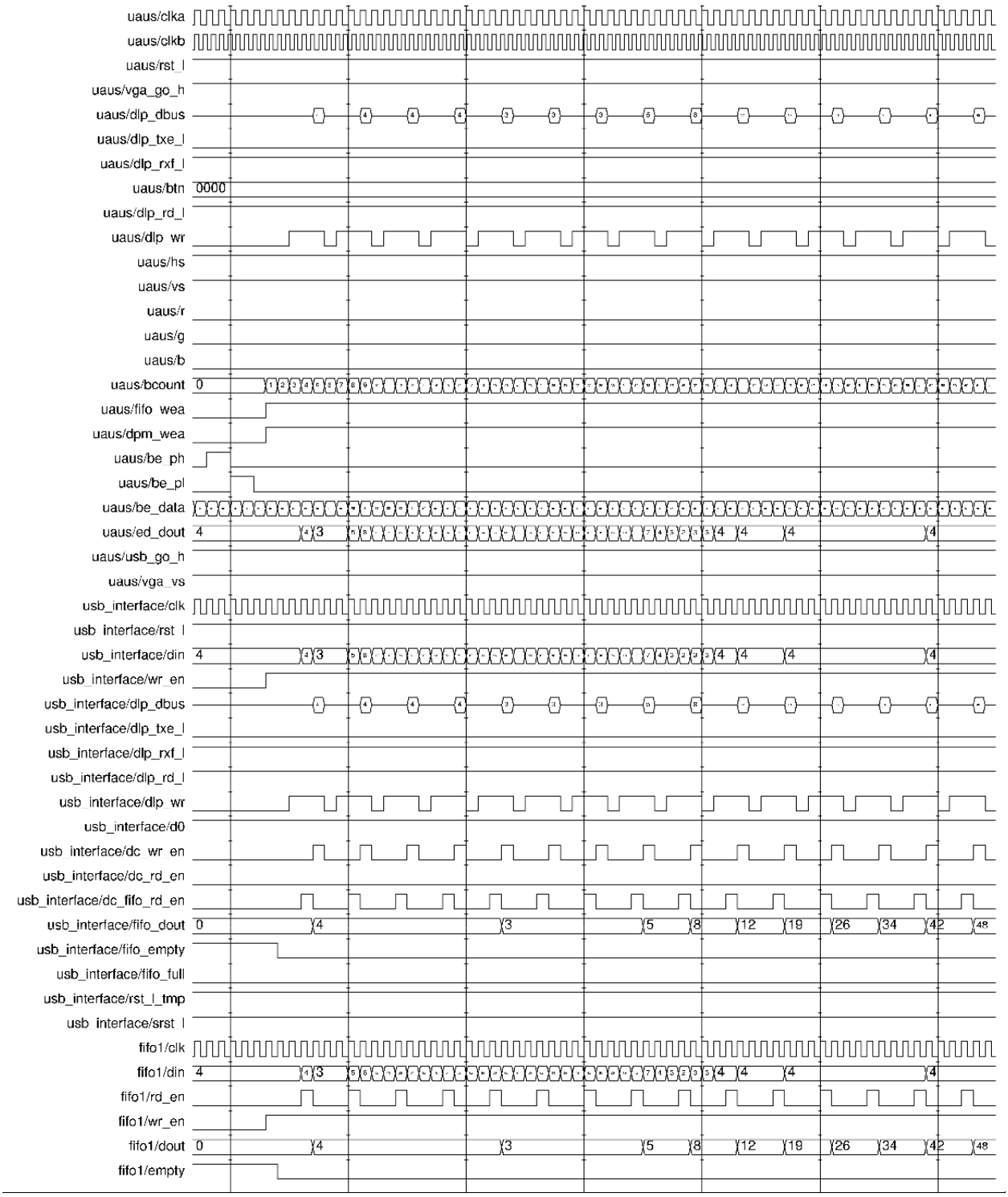

Fig. 57. More detailed timing diagram verifying the operation of µAUS's USB interface. During the vertical retrace period, the USB interface writes data from the FIFO to the DLP module.

```
Release 8.2.01i Trace
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.

C:\Xilinx\bin\nt\trce.exe -ise
C:/Documents and Settings/CSE462/Desktop/uaus/uaus.ise -intstyle ise -e 3 -l 3
-s 5 -xml uaus uaus.ncd -o uaus.twr uaus.pcf -ucf uaus.ucf

Design file:              uaus.ncd
Physical constraint file: uaus.pcf
Device,speed:             xc3s200,-5 (PRODUCTION 1.39 2006-06-02)
Report level:             error report

Environment Variable      Effect
--------------------      ------
NONE                      No environment variables were set
--------------------------------------------------------------------------------

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.



Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clka
------------+------------+------------+------------------+--------+
            |  Setup to  |  Hold to   |                  | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+------------------+--------+
dlp_dbus<0> |    2.623(R)|   -0.400(R)|clka_BUFGP        |   0.000|
dlp_rxf_l   |    2.622(R)|   -0.400(R)|clka_BUFGP        |   0.000|
dlp_txe_l   |    2.623(R)|   -0.400(R)|clka_BUFGP        |   0.000|
rst_l       |    3.687(R)|   -0.400(R)|clka_BUFGP        |   0.000|
vga_go_h    |    3.322(R)|   -1.382(R)|clka_BUFGP        |   0.000|
------------+------------+------------+------------------+--------+

Setup/Hold to clock clkb
------------+------------+------------+------------------+--------+
            |  Setup to  |  Hold to   |                  | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+------------------+--------+
rst_l       |    1.046(R)|    0.111(R)|clkb_BUFGP        |   0.000|
------------+------------+------------+------------------+--------+

Clock clka to Pad
------------+------------+------------------+--------+
            | clk (edge) |                  | Clock  |
Destination |   to PAD   |Internal Clock(s) | Phase  |
------------+------------+------------------+--------+
dlp_dbus<0> |    9.444(R)|clka_BUFGP        |   0.000|
dlp_dbus<1> |   10.652(R)|clka_BUFGP        |   0.000|
dlp_dbus<2> |   10.761(R)|clka_BUFGP        |   0.000|
dlp_dbus<3> |   10.476(R)|clka_BUFGP        |   0.000|
dlp_dbus<4> |   10.419(R)|clka_BUFGP        |   0.000|
dlp_dbus<5> |   10.243(R)|clka_BUFGP        |   0.000|
dlp_dbus<6> |   10.137(R)|clka_BUFGP        |   0.000|
dlp_dbus<7> |   10.023(R)|clka_BUFGP        |   0.000|
dlp_rd_l    |    6.404(R)|clka_BUFGP        |   0.000|
dlp_wr      |    6.404(R)|clka_BUFGP        |   0.000|
```

```
------------+------------+-----------------+--------+

Clock clkb to Pad
------------+------------+-----------------+--------+
            | clk (edge) |                 | Clock  |
Destination |   to PAD   |Internal Clock(s)| Phase  |
------------+------------+-----------------+--------+
b           |    8.201(R)|clkb_BUFGP       |   0.000|
g           |    8.222(R)|clkb_BUFGP       |   0.000|
hs          |    8.347(R)|clkb_BUFGP       |   0.000|
r           |    8.520(R)|clkb_BUFGP       |   0.000|
vs          |    8.584(R)|clkb_BUFGP       |   0.000|
------------+------------+-----------------+--------+

Clock to Setup on destination clock clka
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clka           |   14.446|         |         |         |
clkb           |    2.318|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock clkb
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clkb           |    9.189|         |         |         |
---------------+---------+---------+---------+---------+


Analysis completed Sat Apr 28 22:28:53 2007
--------------------------------------------------------------------------------

Trace Settings:
-------------------------
Trace Settings

Peak Memory Usage: 117 MB
```

Fig. 58. Xilinx ISE Timing Report *without the 50 MHz timing on clkb*: the Data Sheet shows the µAUS's maximum frequency is 69 MHz. However, the report (Fig. 59) with the 50 MHz timing constraint shows a maximum clock frequency of 85 MHz.

```
--------------------------------------------------------------------------------
Release 8.2.01i Trace
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.

C:\Xilinx\bin\nt\trce.exe -ise
C:/Documents and Settings/CSE462/Desktop/uaus/uaus.ise -intstyle ise -e 3 -l 3
-s 5 -xml uaus uaus.ncd -o uaus.twr uaus.pcf -ucf uaus.ucf

Design file:              uaus.ncd
Physical constraint file: uaus.pcf
Device,speed:             xc3s200,-5 (PRODUCTION 1.39 2006-06-02)
Report level:             error report

Environment Variable      Effect
--------------------      ------
NONE                      No environment variables were set
```

```
--------------------------------------------------------------------------------

INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
   a 50 Ohm transmission line loading model.  For the details of this model,
   and for more information on accounting for different loading conditions,
   please see the device datasheet.

================================================================================
Timing constraint: TS_clkb = PERIOD TIMEGRP "clkb" 15 ns HIGH 50%;

 11539 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is  11.772ns.
--------------------------------------------------------------------------------


All constraints were met.
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clkb
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clkb           |   11.772|         |         |         |
---------------+---------+---------+---------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 11565 paths, 0 nets, and 1481 connections

Design statistics:
   Minimum period:  11.772ns   (Maximum frequency:  84.947MHz)


Analysis completed Tue Apr 24 02:00:45 2007
--------------------------------------------------------------------------------
```

Fig. 59. Xilinx ISE Timing Report *with the 50MHz timing constraint on clkb*: the Data Sheet shows the µAUS's maximum frequency is 85 MHz.

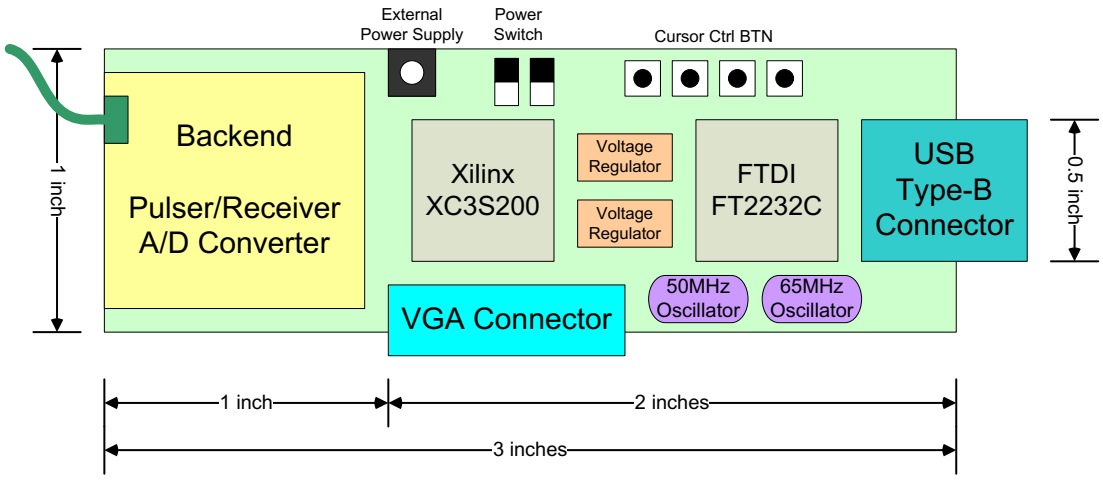

Fig. 60. The prototype layout of the µAUS circuit board with major components including a Xilinx XC3S200-4FTG256C FPGA, a FTDI FT2232C USB Dual USB UART/FIFO IC, a USB type-B connector, a VGA DB15 connector, two voltage regulators, a 50MHz oscillator, a 65MHz oscillator, and a backend which includes a pulser/receiver and an A/D converter.

HW Component	Unit Price	Retailer
Xilinx XC3S200-4FTG256C	$20.88	Digi-Key
FTDI FT2232C	$5.76	Mouser
USB Type-B Connector	$0.59	Mouser
VGA DB15 Connector	$2.23	Mouser
50MHz Oscillator	$2.33	Digi-Key
65MHz Oscillator	$2.33	Digi-Key
Voltage Regulator for FPGA	$0.25	Mouser
Voltage Regulator for USB IC	$0.25	Mouser
8-Bit 50MHz ADC	$1.40	Digi-Key
Total	**$36.04**	

Fig. 61. The estimated price list for each of the major hardware components in the µAUS system. The estimation is based on the retail prices from Digi-Key Corporation [http://www.digikey.com] and Mouser Electronics, Inc. considering the largest volume purchase discount.

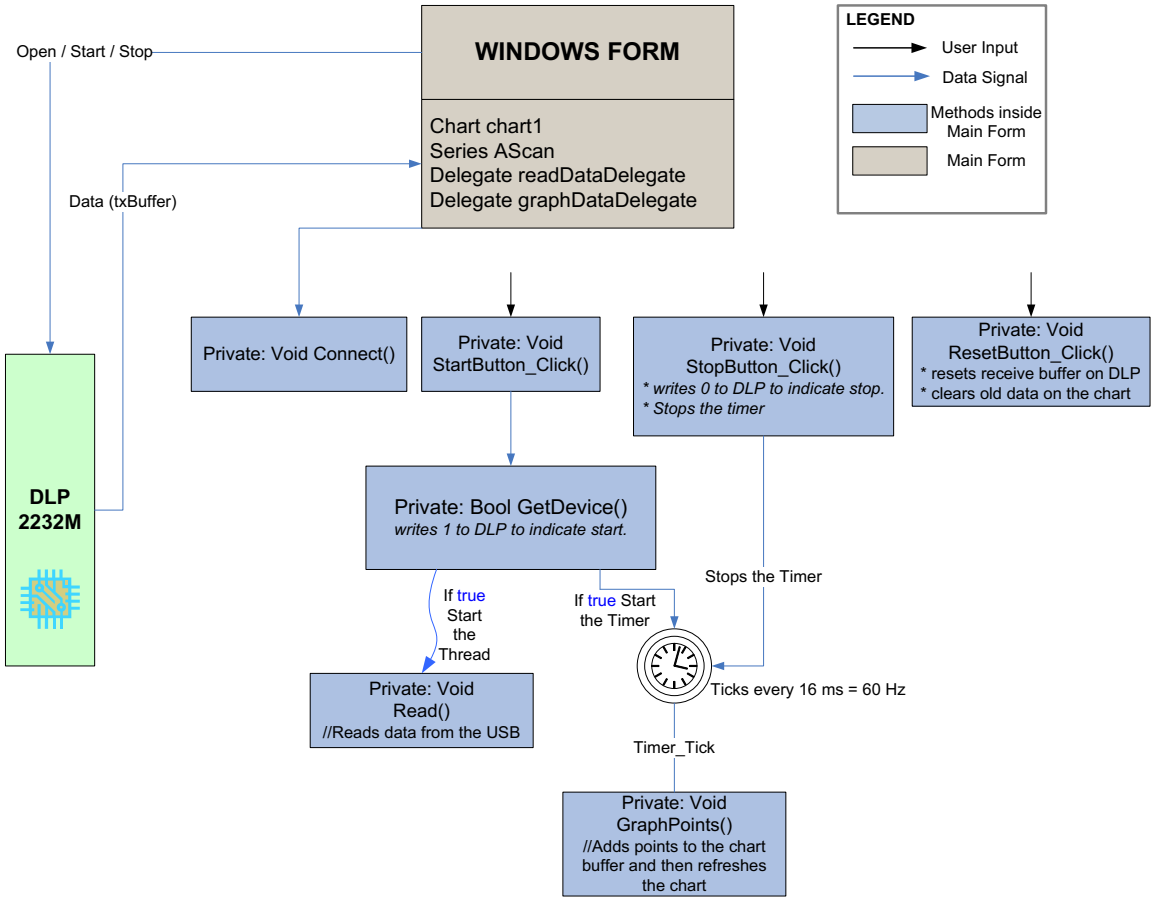

Fig. 62. Block Diagram of the Software Application.

```
//File: Form1.h
//Ascanner
//Purpose: Graphs the data received from the FTDI Chip.
#pragma once

namespace GraphinC {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Threading;
    using namespace System::Text;
    using namespace System::Globalization;

    const UInt32 FT_OPEN_BY_DESCRIPTION    = 2;

#define FT_PREFIX [DllImport("FTD2XX.dll")]

enum {
    FT_OK,
```

```
    FT_INVALID_HANDLE,
    FT_DEVICE_NOT_FOUND,
    FT_DEVICE_NOT_OPENED,
    FT_IO_ERROR,
    FT_INSUFFICIENT_RESOURCES,
    FT_INVALID_PARAMETER,
    FT_INVALID_BAUD_RATE,

    FT_DEVICE_NOT_OPENED_FOR_ERASE,
    FT_DEVICE_NOT_OPENED_FOR_WRITE,
    FT_FAILED_TO_WRITE_DEVICE,
    FT_EEPROM_READ_FAILED,
    FT_EEPROM_WRITE_FAILED,
    FT_EEPROM_ERASE_FAILED,
    FT_EEPROM_NOT_PRESENT,
    FT_EEPROM_NOT_PROGRAMMED,
    FT_INVALID_ARGS,
    FT_NOT_SUPPORTED,
    FT_OTHER_ERROR
};

    typedef void * FT_HANDLE;
    typedef unsigned long DWORD;
    typedef unsigned long FT_STATUS;
    typedef void * LPVOID;
    typedef void * PVOID;
    typedef DWORD * LPDWORD;
    typedef DWORD ULONG;
    typedef unsigned char UCHAR;
    typedef void *HANDLE;
    typedef int BOOL;
    typedef unsigned long ULONG;
    #define FALSE   0
    #define TRUE    1

    using namespace System::Runtime::InteropServices;
    FT_PREFIX FT_STATUS FT_OpenEx(PVOID pArg1, DWORD Flags, FT_HANDLE *pHandle);
    FT_PREFIX FT_STATUS FT_Read(FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD
       nBufferSize, LPDWORD lpBytesReturned);
    FT_PREFIX FT_STATUS FT_Write(FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD
       nBufferSize, LPDWORD lpBytesWritten);
    FT_PREFIX FT_STATUS FT_Purge(FT_HANDLE ftHandle, ULONG Mask);

    // need these kernel functions for the Event Handling stuff
    [DllImport("Kernel32.dll")] DWORD WaitForSingleObject(HANDLE hHandle,  DWORD
     dwMilliseconds);
    [DllImport("Kernel32.dll")] HANDLE CreateEvent(void * pNULL, BOOL
     bManualReset, BOOL bInitialState, char * pcNULL);
    [DllImport("Kernel32.dll")] BOOL SetEvent(HANDLE hEvent);

    FT_HANDLE ftHandle;
    HANDLE hEvent;
       UCHAR rxBuffer[2048];
    /// <summary>
    /// Graphs the data received from the FTDI Chip.
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
        Form1(void)
        {
            InitializeComponent();
            this->AScan = gcnew Dundas::Charting::WinControl::Series();
```

```
            this->chartArea2 = gcnew Dundas::Charting::WinControl::ChartArea();
            this->cont = true;
            this->t = gcnew Thread(gcnew ThreadStart(this,
               &GraphinC::Form1::Read));
            this->lastCursor = 0;
            chart1->ChartAreas->Clear();
            chart1->Series->Add(AScan);
            chart1->BackColor = System::Drawing::Color::WhiteSmoke;
            chartArea2->BackColor = System::Drawing::Color::WhiteSmoke;
            AScan->Color = System::Drawing::Color::Green;
            AScan->Type = Dundas::Charting::WinControl::SeriesChartType::Line;
            chartArea2->AxisX->Minimum = 0;
            chartArea2->AxisX->Maximum = 1024;
            chartArea2->AxisX->MajorGrid->LineColor =
                           System::Drawing::Color::Transparent;
            chartArea2->AxisX->MinorGrid->LineColor =
                           System::Drawing::Color::Transparent;
            chartArea2->AxisY->MajorGrid->LineColor =
                           System::Drawing::Color::Transparent;
            chartArea2->AxisY->MinorGrid->LineColor =
                           System::Drawing::Color::Transparent;
            chartArea2->AxisX->View->Zoomable = true;
            chartArea2->CursorX->UserEnabled = true;
            chartArea2->CursorX->UserSelection = true;
            chart1->ChartAreas->Add(chartArea2);
            this->pixelPanel->Text = "Difference: " + (Convert::ToInt32(this-
             >chartArea2->AxisX->Maximum) - Convert::ToInt32(this->chartArea2-
             >AxisX->Minimum)).ToString();
            this->Connect();
        }

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
            if (components)
            {
                delete components;
                    }
                    //Abort the Thread on Dispose
                    if(t->IsAlive)
                    {
                           t->Suspend();
                           t->Resume();
                           t->Abort();
                    }
        }

    private: Dundas::Charting::WinControl::Chart^  chart1;
    private: Dundas::Charting::WinControl::ChartArea^ chartArea2;
    private: Dundas::Charting::WinControl::Series^  AScan;

    private: System::Windows::Forms::ToolStrip^  toolStrip1;
    private: System::Windows::Forms::Button^  startButton;
    private: System::Windows::Forms::Button^  stopButton;
    private: System::Windows::Forms::Button^  resetButton;

    private: System::Windows::Forms::ToolStripDropDownButton^  toolStripDropDownButton1;
    private: System::Windows::Forms::ToolStripMenuItem^  lineToolStripMenuItem;
    private: System::Windows::Forms::ToolStripMenuItem^  splineToolStripMenuItem;
    private: System::Windows::Forms::ToolStripMenuItem^  rangeToolStripMenuItem;
```

```
    private: System::Windows::Forms::ToolStripMenuItem^  columnToolStripMenuItem;
    private: System::Windows::Forms::ToolStripMenuItem^  pointsToolStripMenuItem;
    private: System::Windows::Forms::StatusBar^  statusBar1;
    private: System::ComponentModel::IContainer^  components;
    private: System::Boolean cont;
    private: System::Boolean connected;
    private: System::Windows::Forms::Timer^  timer1;
    private: System::Windows::Forms::MainMenu^  mainMenu1;
    private: System::Windows::Forms::MenuItem^  menuItem1;
    private: System::Windows::Forms::MenuItem^  connectMenuItem;
    private: System::Windows::Forms::MenuItem^  exitMenuItem;
    private: System::Windows::Forms::StatusBarPanel^  statusPanel;
    private: delegate void graphDataDelegate(); //Delegate for graphing data to avoid
cross-threaded refrences.
       private: delegate void readDataDelegate();  //Delegate for graphing data to
thread safety (race condition).
    private: System::Windows::Forms::StatusBarPanel^  pixelPanel;
       private: Thread^ t;

       private: System::Double lastCursor;
    protected:

    private:

#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void)
        {

             //Auto Generated Code
             //P.S: Dr. Richard let me know if you want to view this portion and
             //I'll email it to you.
         }
#pragma endregion

private: System::Void Connect()
         {
             FT_STATUS ftStatus;
             this->statusPanel->Text = "Getting Device Status...";
             ftStatus = FT_OpenEx("DLP2232M B",FT_OPEN_BY_DESCRIPTION,&ftHandle);
             if (ftStatus != FT_OK)
             {
                 this->statusPanel->Text = "Unable to Connect";
                 this->connectMenuItem->Enabled = true;
             }
             else
             {
                 this->statusPanel->Text = "Connected";
                 this->startButton->Enabled = true;
                 this->stopButton->Enabled = true;
                 this->connectMenuItem->Enabled = false;
                 t->Start();      //Start the Thread
                t->Suspend();    //Suspend the Thread until needed
             }
          }

private: System::Boolean getDevice()
         {
             FT_STATUS ftStatus;
             DWORD bytesWritten;
```

```
            UCHAR    txBuffer[1]; // transmit buffer
            txBuffer[0] = 1;
            ftStatus = FT_Write(ftHandle, txBuffer, 1, &bytesWritten);
            if (ftStatus != FT_OK)
            {
                this->statusPanel->Text = "Error: Could not write byte.";
                return false;
            }
            else
            {
                 return true;
            }
        }
private: System::Void stopButton_Click(System::Object^  sender, System::EventArgs^  e)
        {
            t->Suspend();  //Suspend the Read Thread to Stop Reading
            this->timer1->Stop(); //Stop the Timer to Stop Plotting
            this->startButton->Enabled = true;
            DWORD bytesWritten;
            UCHAR txBuffer[1]; // transmit buffer
            txBuffer[0] = 0;
            FT_Write(ftHandle, txBuffer, 1, &bytesWritten);
            this->resetButton->Enabled = true;
        }
private: System::Void resetButton_Click(System::Object^  sender, System::EventArgs^  e)
        {
            this->AScan->Points->Clear();   //Resets Chart's Series
            this->chart1->Refresh();
            if(this->startButton->Enabled == false)
             {
                    this->startButton->Enabled = true;
             }
            this->resetButton->Enabled = false;
        }
private: System::Void startButton_Click(System::Object^  sender, System::EventArgs^  e)
        {
             if(this->getDevice())
            {
                    t->Resume();  //Resume the already started Read thread
                    this->timer1->Start();   //Start the Timer to Start Plotting
                    this->startButton->Enabled = false;
                    this->resetButton->Enabled = false;
             }
        }
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e)
       {
             graphDataDelegate^ gDelegate = gcnew graphDataDelegate(this,
               &GraphinC::Form1::GraphPoints);
             gDelegate->Invoke(); //Invoke the plotting funtction thread safely.
       }
private: System::Void Read ()
              {
                     DWORD bytesReceived;
                     DWORD dwRxBytes;
                     FT_STATUS ftStatus;
                     ftStatus = FT_Read(ftHandle, rxBuffer, 2048,
                       &bytesReceived);
                     if (ftStatus != FT_OK || bytesReceived == 0)
                     {
                       this->statusPanel->Text = "Error: could not read byte.\n";
                       return;
                     }
                     else
```

```
                    {
                      readDataDelegate^ rDelegate = gcnew readDataDelegate(this,
                        &GraphinC::Form1::Read);
                      rDelegate->Invoke(); //Loops back to Read again
                    }
                }
private: System::Void GraphPoints()
         {
               AScan->Points->Clear();
               for(int i =0; i < 1024; i++)
               {
                  // Add data to the Chart Buffer(Series)
                   AScan->Points->AddY(Convert::ToInt32(rxBuffer[i]));
}
                   chart1->Refresh(); //Paint the Chart Buffer.
          }

               //Behavioral Methods
private: System::Void Form1_Resize(System::Object^  sender, System::EventArgs^  e) {
             this->chart1->Width = this->Width - 30;
             this->chart1->Height = this->Height - this->startButton->Height - this-
>statusBar1->Height - 120;
    }

private: System::Void chart1_AxisViewChanged(System::Object^  sender,
Dundas::Charting::WinControl::ViewEventArgs^  e)
    {
        if(this->chartArea2->AxisX->View->Size > 0 && this->chartArea2->AxisX-
           >View->Size <=1024)
        {
            this->pixelPanel->Text = "Difference: " + this->chartArea2->AxisX-
             >View->Size.ToString();
        }
        else
        {
               this->pixelPanel->Text = "Difference: 1024";
        }
    }
private: System::Void exitMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
    {
             Application::Exit();
    }

              //On Click Methods
private: System::Void lineToolStripMenuItem_Click(System::Object^  sender,
System::EventArgs^  e)
        {
             AScan->Type = Dundas::Charting::WinControl::SeriesChartType::Line;
         }
private: System::
Void splineToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^  e)
         {
             AScan->Type = Dundas::Charting::WinControl::SeriesChartType::Spline;
         }
private: System::
Void rangeToolStripMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
         {
             AScan->Type = Dundas::Charting::WinControl::SeriesChartType::Range;
         }
private: System::
Void columnToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^  e)
        {
             AScan->Type = Dundas::Charting::WinControl::SeriesChartType::Column;
```

```
        }
private: System::
Void pointsToolStripMenuItem_Click(System::Object^ sender, System::EventArgs^  e)
        {
            AScan->Type = Dundas::Charting::WinControl::SeriesChartType::Point;
        }
private: System::
Void connectMenuItem_Click(System::Object^  sender, System::EventArgs^  e)
        {
            this->Connect();
        }
};
}
```

Fig. 63. C++ code for the Software application.

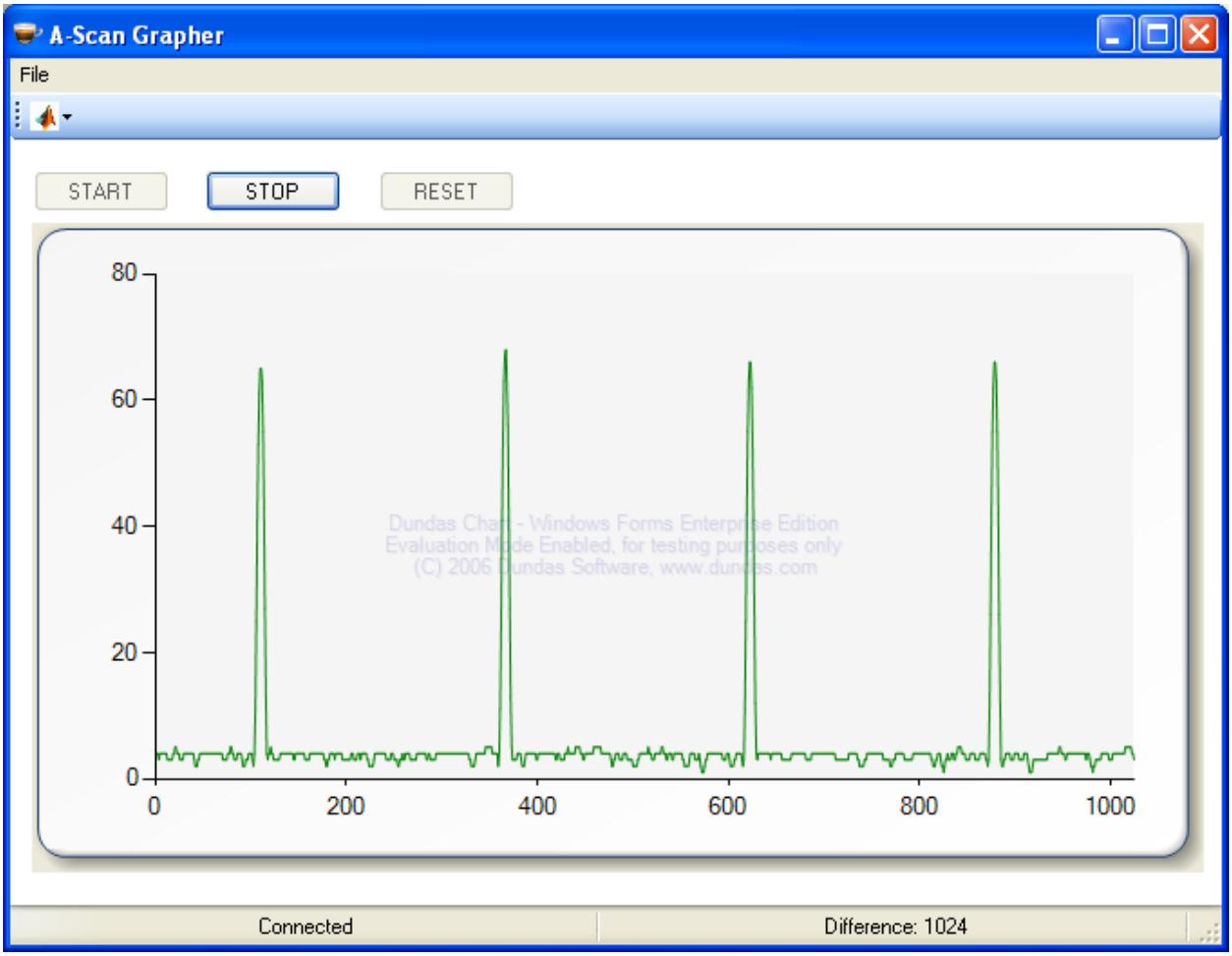

Fig. 64. Sreenshot of the application showing the A-mode display.