World Academy of Science, Engineering and Technology
International Journal of Electronics and Communication Engineering
Vol:4, No:11, 2010

Real-Time Digital Oscilloscope Implementation in 90nm CMOS Technology FPGA

Nasir Mehmood, Jens Ogniewski and Vinodh Ravinath

Abstract**—**This paper describes the design of a real-time audio-range digital oscilloscope and its implementation in 90nm CMOS FPGA platform. The design consists of sample and hold circuits, A/D conversion, audio and video processing, on-chip RAM, clock generation and control logic. The design of internal blocks and modules in 90nm devices in an FPGA is elaborated. Also the key features and their implementation algorithms are presented. Finally, the timing waveforms and simulation results are put forward.

Keywords**—**CMOS, VLSI, Oscilloscope, Field Programmable Gate Array (FPGA), VHDL, Video Graphics Array (VGA)

I. INTRODUCTION

A digital oscilloscope is required in modern laboratories to analyze the various parameters of any sort of input signal. It may consist of pre-processing circuits, analog to digital conversion, logic controller circuits, level generating circuits, video controlling circuits, clock generation circuits and memory circuits [1]. Digital oscilloscope is the basic instrument for testing and measurement. It may have the functions of waveform display in horizontal and vertical axes, timing measurement, voltage level adjustment, and zoom-in and out and other special functions.

FPGA stands for field programmable gate array and is a new technology to devise digital integrated circuits which are pre-fabricated using standard VLSI fabrication methods [1]. In FPGA-based design, the designer has the possibility to modify and re-design using a user-friendly programming language like Verilog or VHDL[1]. FPGAs are widely used in prototype circuit development for test and trial purpose. They contain a fabric structure in which programmable logic elements are connected together through programmable interconnects. A typical FPGA may contain thousands of logic gates and millions of transistors arranged in a specified manner. FPGA programming takes place in a user-friendly environment, in which a software code is transformed into hardware connections using FPGA built-in programmable fabric [2].

In this paper we will present the design of a real-time audio range digital oscilloscope and its implementation in an FPGA using 90nm CMOS transistors. This oscilloscope will not require any computer for its working. The oscilloscope operates in real-time to acquire the analog audio data stream generated by the PC sound card. The analog data is converted in digital format using on-board ADC chip. Further processing is performed upon digital data to reproduce the audio stereo signal, display its waveform on VGA screen to control the parameters of these signals. This oscilloscope is very cheap and its functions are controlled through keyboard.

Authors are with Air University, Islamabad, Pakistan
e-mail:mehnasir@gmail.com

Various keys are assigned for specific functions like volume up and down, x-axis and y-axis control, balance control, freezing; zooming etc. the design of this oscilloscope can be loaded into any FPGA and used in conjunction with keyboard and VGA monitor.

II. SYSTEM DESIGN AND OPERATION

The figure 1 shows the overall digital oscilloscope system block diagram.

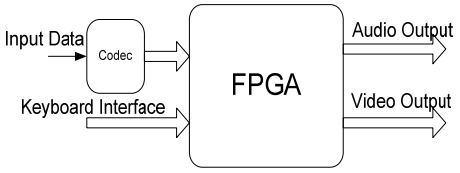

Fig. 1: Oscilloscope System Block Diagram

The system consists of audio input/output interface, keyboard interface and VGA interface. The input signal is the stereo audio signal in digital format. The keyboard acts as a controlling device for the hardware. The function of the system is to control the volume of the input signal, control the balance of both channels, process the input signal to display on VGA screen and generate the output signal to speakers [3].

The figure 2 comprehensively explains the various functional modules of the system.

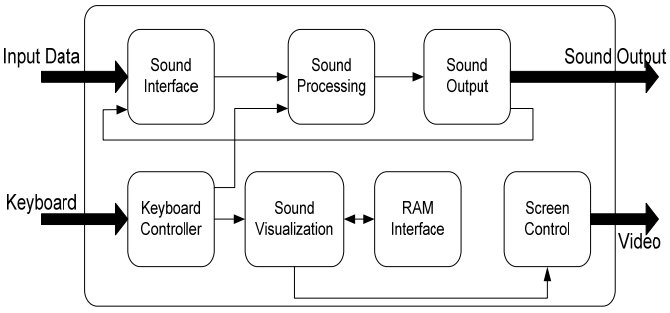

Fig. 2: Oscilloscope Functional Diagram

A. Sound Interface (SI)

This block takes the audio signal as input from the on-board codec. The input sound data consists of a serial row of 20 bits. The '*LRCK*' clock is used to split the data into 'left' and 'right' channels. Left channel is sampled on the rising edge of '*LRCK*' and the right channel is sampled on the falling edge of '*LRCK*'. The '*SCLK*' clock is used to receive and transmit the serial stream of sound data. The data is received by the system on the rising edge of '*SCLK*' and the processed data is transmitted to the codec at the falling edge of '*SCLK*'. A 'ch_select' signal is generated which toggles

whenever the sampling of 20 bit sound data is over. Figure 3 explains the above mentioned operation [3].

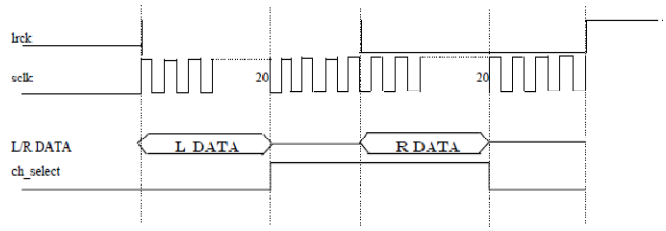

Fig. 3: Codec clock and data triggering

The following code is an extract from the SI block behavioral description.

```
process(clk,rst)
begin
if rst='0' then old_sc <= '0';
sel_tmp <= '0';data_out_tmp <= "00000000000000000000";
elsif(clk'event and clk='1' and clk'last_value='0') then
old_sc<= sclk; if(sclk='1' and old_sc ='0') then
if(bit_cntr=19 and lrck='1') then data_out_tmp <=reg;
sel_tmp <= '1'; elsif(bit_cntr=19 and lrck='0') then
data_out_tmp <= reg; sel_tmp <= '0'; else
data_out_tmp <= data_out_tmp; sel_tmp <= sel_tmp;
end if; else data_out_tmp <= data_out_tmp;
sel_tmp <= sel_tmp; end if; end if;
end process;
```

B. Sound Processing (SP)

This block processes the sound signal for volume and balance control upto 10 levels. The control signals for volume and balance are generated by the keyboard interface block '*VOL*' and '*BAL*' respectively. The volume is controlled by continuous right shift operations based on the current volume level. The volume of both the channels is independent of the '*ch_select*' signal. An extract form the code of volume control is shown below.

```
right_vol <= right_vol_tmp; left_vol <= left_vol_tmp;
proc_sgns : process (clk, rst)
begin
if (rst='0') then left_vol_tmp <= 10; right_vol_tmp <= 10;
    elsif (clk'event and clk = '1') then
      if (ch_vol = '1') then
        if (new_vol = '0') then
          if (left_vol_tmp > 0) then
            left_vol_tmp <= left_vol_tmp + 2;
          else
            left_vol_tmp <= 0;
      end if;
```

The data processed by the '*volume_control*' block is fed to the '*balance_control*' block. The balance of the sound channels is controlled by using continuous shift operations based on the status of '*ch_select*' signal. If '*ch_select*' is high and the balance level is less than 5 the right channel data is shifted right according to the balance level, otherwise the volume of both channels remains equal. The vice versa condition is applied for left channel if '*ch_select*' signal is low and balance is greater than 5 [3].

C. Sound Output (SO)

This block performs two basic functions:

1) Transmitting the processed sound data to codec. The data is first converted into a serial stream of 20 bits left and right channel data and then each bit is sent to the codec on every falling edge of '*SCLK*' clock. The channel selection is made by '*LRCK*' clock. When high the left channel data is sent out, otherwise right channel data. Only the first 20 bits are valid for both channels. Another control signal '*ch_select*' gives indication of when the data of one channel is ready to transmit.
2) Generating clocks required for the proper reception and transmission of sound data. These clocks include '*MCLK*', '*SCLK*' and '*LRCK*'.

The VHDL code for the generation of these clocks is shown below.

```
process (clk,rst)
begin
   if rst='0' then cntr <="00000000";
  elsif(clk'event and clk='1') then cntr <=cntr+1;
  end if;
  end process;
  sclk <=cntr(1);
  counter <=cntr(6 downto 2);
  lrck <=cntr(7);
  mclk <=clk;
```

D. Keyboard Control (KC)

The 'Keyboard Control' consists of a Keyboard Interface (KI) and Keyboard Decoder (KD).

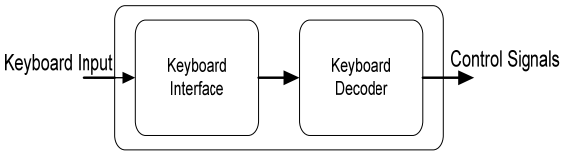

Fig. 4: Keyboard Controller

The function of this unit is to read the scan code generated by the keyboard keys and to set various control signals according to control block. The KI accepts the keyboard data serially along with keyboard clock and reads the scan code. This block will set the different control signals reference to the character scan code. The control signals for volume and balance control are '*vol*' and '*bal*' respectively. These are 4 bit control signals. When key 'M' is pressed the value of control signal is generated such that the volume increases and vice versa for 'N' key. A similar procedure exists for balance control. When key 'W' is pressed, the balance shifts towards left channel and when key 'Q' pressed the balance shifts towards right channel.

E. Signal Visualization (SV)

This block is responsible for the display of sound signal and the volume level indicator on the VGA screen. It consists of 4 sub-blocks named as (1) Volume Visualization, (2) Special Functions, (3) Waveform Function and (4) Image Creation. Figure 5 shows the interconnections between these sub-blocks. An extract from the waveform function VHDL code is shown below.

```
  if (clk'event and clk = '1') then
      if (wf_x_in < 793 and sound_left_en ='1') then
        wf_x_in <= wf_x_in + 1;
```

```
    wf_x <= conv_std_logic_vector(wf_x_in,10);
    wf_y <= sound_left(18 downto 10);
    wf_col <= "01";
    wf_en <= '1';
   elsif (wf_x_in <793 and sound_right_en ='1') then
    wf_x_in <= wf_x_in + 1;
    wf_x <= conv_std_logic_vector(wf_x_in,10);
    wf_y <= sound_right(18 downto 10);
    wf_col <= "10";
    wf_en <= '1';
   else
    wf_x <= (others => '0');
    wf_y <= (others => '0');
    wf_en <= '0';
    wf_col <= "11";
 end if;
```

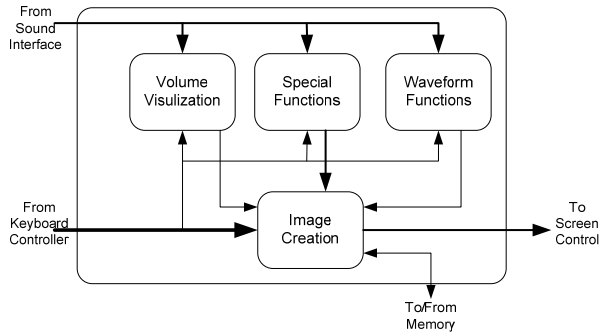

Fig. 5: Signal Visulization Block

The signal visualization is done in several steps. The first step is the sampling of the signal. The VHDL-block '*waveform_function*' takes care of that. Another block '*wf_ctrl*' creates the enable signals for that block according to that '*waveform_function*' will save the signal in a Xilinx block RAM. Later it loads the signal from the RAM and passes it to the block '*sig_alias*'. The function of 'sig_alias' is to create pixels representing the different samples. In parallel, '*vol_bal_gen*' creates the output for the visualization of the volume-level and the balance, and '*RAM-Interface*' loads the picture samples stored in the flash RAM.

The last block '*sel_col*' selects the parts which will be displayed. The highest priority goes to the display of the input audio signals, then the visualization of volume and balance, and finally the background picture.

To fully understand how the visualization of signals works, it is important to know how the sampling is done. In normal operation, 64 successive samples will be taken. Let's call this one Megasample. One Megasample will be taken every 40 ms, representing the 25 Hz which the screen's refresh rate. It is possible to adjust the time between 2 Megasamples as well as the time between two samples inside the Megasamples. By doing so it is possible to filter out the higher frequency signals.

The *Volume Visualization* (VV) block takes the 'volume level' and 'balance level' as input and transforms them into the VGA control signal values and then displays the volume bar accordingly on the screen.

The *Special Function* (SF) block implements the complex DSP functions like Fast Fourier Transform (FFT) and the Power Spectrum. The *Waveform Function* block transforms the input sound signal into pixel co-ordinates. It takes input from SI block and gives the (x, y) coordinates of the pixel value to be displayed on the screen.

The *Image Creation* block controls the communication of system with RAM Interface block. It takes the (x,y) coordinates from the other functional blocks and generates a set of control signals which includes address to be fed to the RAM Interface.

F. RAM Interface (RI)

This part regulates the access to the RAM for the signal visualization (SV) block. It takes address input from SV block as well as the pixel data to be written at that address. It generates a '*write_done*' signal whenever a byte of data has been written to the RAM. The RI code extract is shown below.

```
  get_pix : process(clk, rst)
  begin
    if (rst = '0') then
     pix_byte_tmp <= "00000000";
   elsif (clk'event and clk='1') then
    if (x_tmp(1 downto 0) = "01") then
    pix_byte_tmp <= data;
    else
  pix_byte_tmp(5 downto 0)<= pix_byte_tmp(7 downto
2);
     end if;
    end if;
 end process get_pix;
```

G. Screen Control (SC)

This block formats the processed audio data and displays it on the VGA screen. It takes the pixel values for each channel and displays them with different colors and at the proper locations on the screen. The process of screen control is facilitated by the generation of sync ('*hsync*' and '*vsync*') and blanking pulses [4]. The VGA signal timings are explained in figure 6.

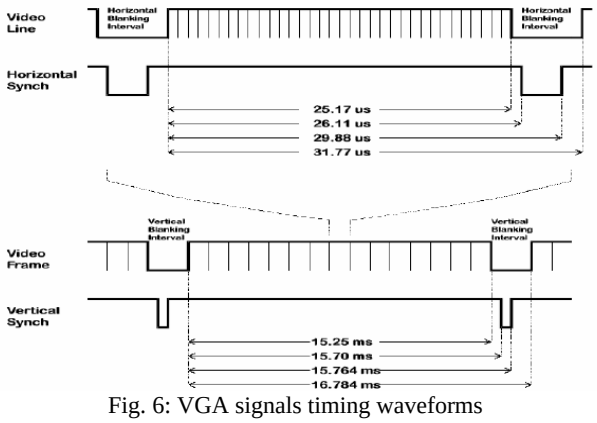

Fig. 6: VGA signals timing waveforms

The following code explains the creation of blanking and sync pulses.

```
crt_blank : process(clk)   begin
if (clk'event and clk='1') then
blank <= h_blank or v_blank;   end if;
end process crt_blank;
crt_hsync : process(clk, rst)   begin
if (rst = '0') then   hsync <= '1';   h_blank <= '0';
elsif (clk'event and clk='1') then
```

```
if (x > 593 and x < 689) then hsync <= '0';   else
hsync <= '1';    end if;
if (x > 512) then h_blank <= '1';    else
h_blank <= '0';   end if;  end if;
end process crt_hsync;
```

III. SYSTEM FEATURES AND SPECIFICATIONS

The features and specifications of the system are as mentioned in table 1.

TABLE I DESIGN FEATURES AND SPECIFICATIONS

S.No	Features	Specifications
1	Digital volume control	10 levels volume control
2	Digital balance control	10 levels balance control
3	Display of sound signal	Both channels simultaneously. Left or right channel separately
4	Display of volume	10 levels per volume value
5	Display of balance	10 levels per balance value
6	Zoom in and out	Upto 8X
7	Sampling frequency control	Increasing or decreasing X2
8	Color coding	Channels have distinct colors. Blue for Left and Green for right
9	Freezing of sound signal	The signal is freeze as per toggle key
10	Aliasing	Connects the adjacent pixels on the screen

IV. SIMULATION RESULTS

The oscilloscope project is performed in VHDL language using Mentor Graphics' FPGA Advantage software. The hardware design is split into various interconnected modules and sub-modules. Each module is then simulated and tested separately. A final simulation is performed after integration of all modules and sub-modules. Figure 7 shows the timing waveforms for some of the variables and outputs.

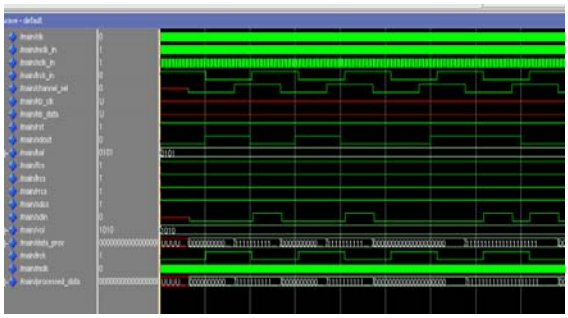

Fig. 7: Simulation waveforms

V. CONCLUSIONS

A real-time digital oscilloscope in audio range frequency has been designed and implemented in 90nm FPGA device. All the design work has been performed in VHDL and implementation is done on NEXYS2 board containing Spartan-III SC3S500E FPGA device. The oscilloscope needs FPGA board, keyboard and VGA screen to operate properly. The input data is sampled, digitized and processed and the audio signal is displayed on the screen. All the features mentioned in table 1 have been implemented. The synthesis summary shows that only 50 I/Os, 1321 function generators, 661 CLBs, 603 flip flops or latches and 02 block RAMs have been utilized.

ACKNOWLEDGMENTS

We are thankful to Dr. Kent Palmkvist and Kenny Johansson belonging to Division of Electronic Systems, Institute of Technology, Linköping University, Sweden, for their support and cooperation during the research work.

REFERENCES

[1] He Zhiqiang, Zeng Wenxian, Li Jianke, The Eighth International Conference on Electronic Measurement and Instruments ICEMI'2007 "*An Embedded Virtual Digital Storage Oscilloscope with 1GSPS*".
[2] Wyne Wolf " *FPGA Based System Design*" Pearson publisher Printice Hall.
[3] Nasir Mehmood, Jens Ogneewski, Vinodh Ravinath, Project Report "*Digital Oscilloscope*" Group 02, Year 2005/First Semester ISY/LiTH.
[4] Project Report by Amr Mohamed, Fady, Kareem, Gamal, Mazen and Sherief , "*VHDL implementation of oscilloscope usig FPGA*", Alexandria University.