FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Dep. Eng. Electrotécnica e de Computadores

Report

Digital Systems' Project

Final Project - Synthesizer

Authors:
Ana Cláudia Fernandes dos Reis
2011149543
Francisca Agra de Almeida Quadros
2011149841

Date:
03-07-2015

INDEX

1. INTRODUCTION

The purpose of this project is to implement a synthesizer using an FPGA and a touch panel. In order to do so, a digital audio signal with a specific volume and pitch is created and fed to the FPGA audio chip. These values (volume and frequency) will be controlled by the coordinates (x and y) of the points that the user touches on the LCD touch panel.

Therefore if the user wishes to increase the frequency, he should touch a point located more to the right (increasing the x coordinate). Furthermore, if we wish to increase the volume we should choose a lower point (increasing the y coordinate), this may seem non intuitive but the origin of the coordinates system of the touch panel is located on the top of the screen, therefore choosing a lower point increases the value of y.

To accomplish our goal we relied on some fundamental equipment. First we have the DE2 with an FPGA of Altera Cyclone II EP2C35 which is the base of our work. Then, connected to the DE2 is the LTM that consists of three major components: LCD Touch Panel module, AD converter, and 40-pin expansion header. So, the interfaces of the LTM are connected to the DE2 via the 40-pin expansion connector. Then, the AD converter will convert the coordinates of the touch point to its correspondent digital version that will finally be outputted to our sound components.

The implementation of the touch screen component was quite simple, because we could reuse a module that was present on the instruction CD of the LTM. Therefore we only needed to make the necessary changes in order to output the coordinates as we wished to.

Also, we edited an Audio Controller that takes the samples of sound we give him and send them to the speakers.

The most challenging part was producing and applying the actual changes to the sound, which we made through the development of a block called GeraSom.

2. SYSTEM OVERVIEW

The main focus of our system revolves around the manipulation of a pre-determined sound, generated by our synthesizer. Therefore one of the main obstacles we encountered whilst developing this project was the alteration of the sound wave's frequency. This was achieved after experimenting with several different techniques. On the other hand, the task of altering its volume was quite simple and it was implemented by multiplying the sound by a range of values that could either increase or decrease its intensity.

After some research we decided to take on a simple technique in order to alter the sound's sampling frequency. The sound wave will be composed only by two samples with symmetric intensities ($1x10^7$ and $-1x10^7$) and its pitch will be altered by delaying the switch between the two samples. This happens because the number of samples given for the same intensity are altered and therefore the sound's frequency changes.

These methods are the core of our synthesizer, forming a sound generating module which is connected to and audio controller. Therefore our system will be formed by three main components: a sound generating block, a sound controller and a block with functions regarding the touch screen. This last block is used to obtain the (x,y) coordinates of the touched point and to upload a picture onto the LTM screen.

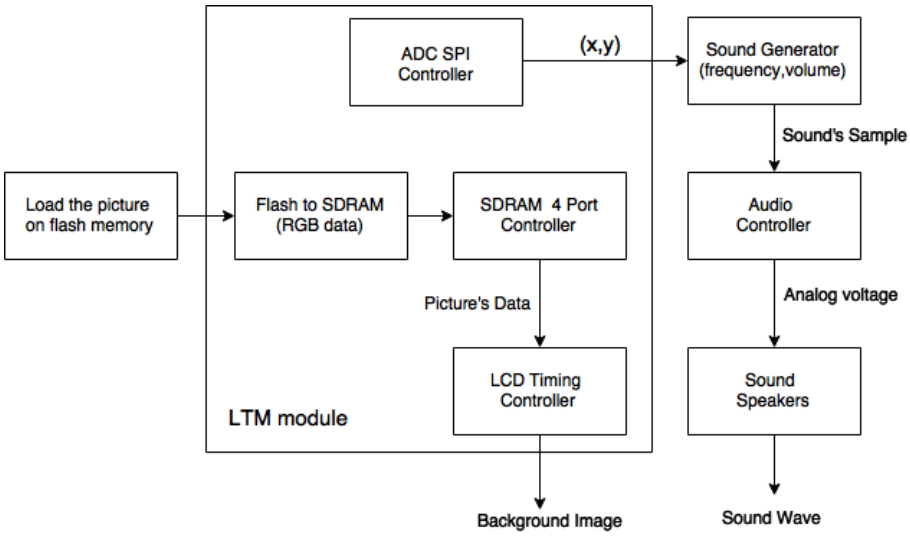

Diagram 1. *System overview*

3. HARDWARE DESIGN AND IMPLEMENTATION

3.1. Emission and manipulation of sound

In order to manipulate the pitch of a sound wave, we first tried to alter the sampling frequency of the audio chip. However we decided that this was not a good approach as the range of sampling frequencies was very limited for what we were trying to achieve.

Another technique that was experimented with was taking the Fast Fourier Transform (fast algorithm that calculates the Discrete Fourier Transform) of the audio signal, alter it in the frequency domain and then take the inverse FFT. This is called frequency domain pitch shifting, however it proved to be very mathematically intensive and therefore complicated to do in hardware.

However we know that a sound can be represented by a sequence of samples, which represent the intensity of the signal. Therefore we choose to take on a simpler approach, and create a sound wave composed entirely by two different samples/values of intensity.

This makes the manipulation of frequency much easier, as we can control the sampling frequency of the sound (number of samples per second). For example, if we wish to decrease the frequency, we simply feed more samples of the same value and delay the switch between value 1 and value 2. This is a very simple and effective way to alter the sound, which can be implemented with a few lines of coding.

As we previously mentioned before, the sound's sampling frequency will be controlled by delaying the switch between the two intensity values ($1x10^7$ and $-1x10^7$). Consequently, this will alter the number of samples given for the same intensity, and change the sound's frequency.

To implement this method, we created a counter, which increases on each CLOCK_50 ascending wave and controls the switch between the opposite values of intensity. If the user decides to decrease the sound frequency, he just needs to increase the value of the counter, which means that more samples of the same value will be fed before the switch to another value of intensity occurs.

The following example shows a sound generated with the counter set to 20, which means that each $(\frac{1}{50MHZ})$ $2x10^{-7}$ seconds, the value $1x10^{7}$ is fed to the audio chip and the counter is increased. When the counter hits 20, and $4x10^{-6}$ seconds have passed the sample value is switched to $-1x10^{7}$ and the counter starts again from zero.

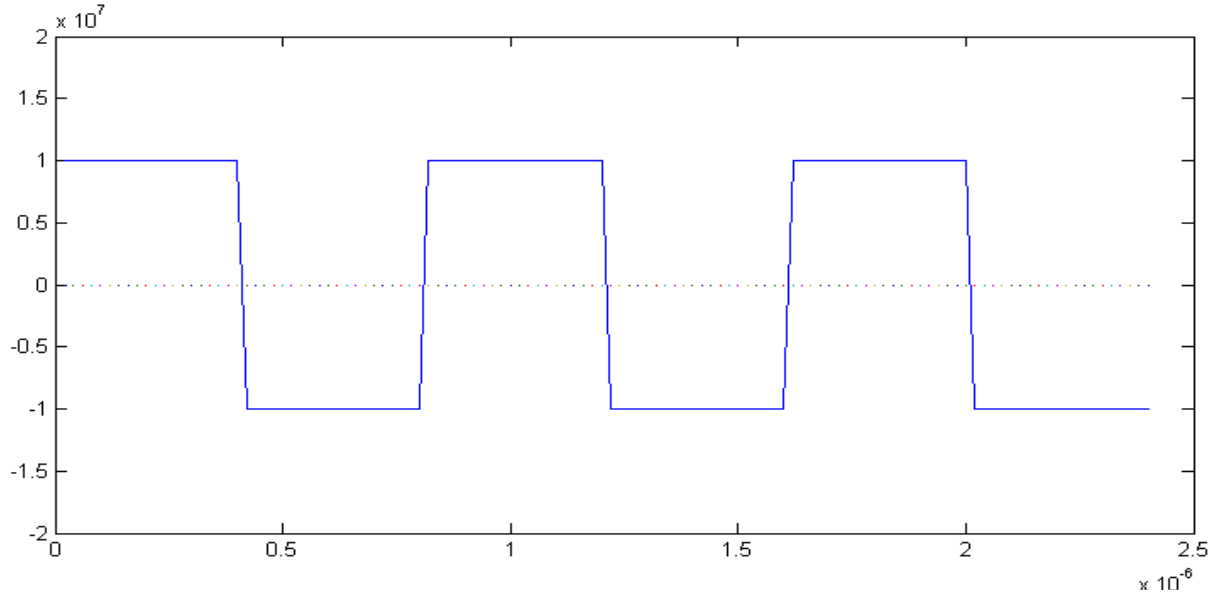

Figure 1. *Representation a possible sound wave.*

For this example we used a very low value for the counter because it made the process of generating the sound wave simpler, 20 was used for experimental purposes only.

In reality the value for the counter is given by the x coordinates that we receive from the touch panel, who can vary between 0 and 16. This is a very low range and would originate very high frequencies, therefore we add a random amount in front of the bits that correspond to the coordinates, in order to increase the counter's order of magnitude.

The following image shows how the samples are alternated according to the value "delay". This variable holds the maximum value for the counter, and as we can observe when "counter" reaches its maximum value it is set back to zero and the variable "invert" is changed to its opposite value.

```
/*****************************************************************************
 *                              Sequential Logic                             *
 *****************************************************************************/

always @(posedge CLOCK_27)
    if(counter == delay) begin
        counter <= 0;
        invert <= !invert;
    end
    else counter <= counter + 1;

wire [31:0] sound = (SW == 0) ? 0 : invert ? 32'd10000000 : -32'd10000000;
```

Figure 2. *Sequential logic of the "GeraSom" module.*

The variable "sound" corresponds to a sample which will be fed to the audio chip in order to generate the desired sound wave. If no values of volume or frequency are specified (SW==0) "sound" will be set to zero. On the other hand, if we receive a certain value for the volume and frequency, the variable "sound" will be equal to either $1x10^7$ or $-1x10^7$, depending on the current value of "invert".

This sequential logic is used in our "GeraSom" module, whose purpose is to receive the X and Y coordinates from the touch panel (that set the volume and pitch) and generate samples that will create the desired audio signal.

```
/*****************************************************************************
 *                              Port Declarations                            *
 *****************************************************************************/
// Inputs
input               CLOCK_27;                 // Clock signal
input       [7:0]   SW;                       // X and Y coordinates

// Outputs
output      [31:0]  left_channel_audio_out;   //Sound fed to the left speaker
output      [31:0]  right_channel_audio_out;  //Sound fed to the right speaker


/*****************************************************************************
 *                 Internal Wires and Registers Declarations                 *
 *****************************************************************************/

// Internal Wires
wire [19:0] delay;          // Value that sets the desired frequency
wire [7:0]  volume;         // Used to change the sounds volume

// Internal Registers
reg         invert;        // Used to alternates between sound samples
reg  [19:0] counter;       // Counter whose maximum value is set by delay
```

Figure 3. *Inputs and Outputs of the "GeraSom" module.*

As we mentioned before, the values that the module receives have a very small magnitude (they vary between 0 and 16) therefore we need to increase its order of magnitude or else the frequencies will be too high (the counter would hit its maximum too quickly and the alternation between samples would be too fast).

In order to do so, we use an internal register "delay" whose value result from the combination of the xx coordinates (given by the inputs SW[3:0]) and the decimal value 30000. On the contrary the register "volume" doesn't need this increase in magnitude, therefore its value results directly from the yy coordinates (given by the inputs SW[7:4]).

```
assign delay  = {SW[3:0],15'd30000};
assign volume = {SW[7:4]};
assign left_channel_audio_out   = sound*volume;
assign right_channel_audio_out  = sound*volume;
```

Figure 4. *Assignments of the "GeraSom" module.*

After the value for the sample “sound” is set, it is multiplied by the variable “volume” in order to create the final sample (with the desired pitch and volume) that will be fed to the right and left speakers.

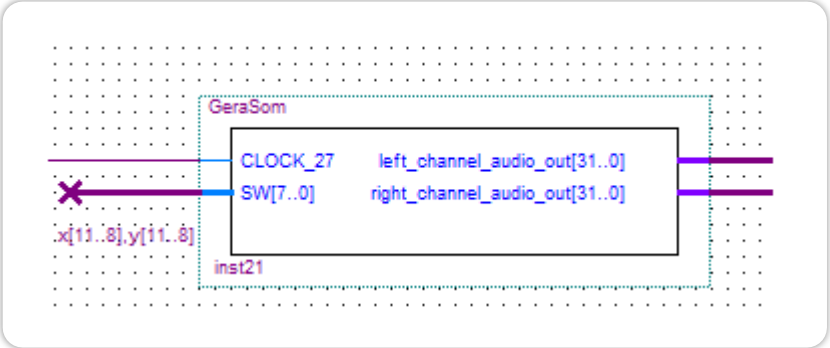

Figure 5. *“GeraSom” module.*

In the upper picture we can see the symbol for our module, whose inputs are a clock signal with a frequency of 27MHz and the most significant 4 bits of the x and y coordinates (which are outputs from the touch panel).

This concludes the description of the “GeraSom” module, which is responsible for the manipulation of sound in which the synthesizer is based on. After this module was completed, we simply connected it to an Audio Controller “Altera_UP_Avalon_Audio” which was already implemented and ready to use (even though it was a pre-made module we decided to edit it in order to adapt it to our project).

We will briefly explain the behavior of this pre-implemented module, which allows us to write data (sound wave samples) into the audio chip (WM8731) on Altera’s DE2 board.

The WM8731 is a low power audio CODEC designed for portable audio products. This codec includes line and microphone inputs to the ADC and headphone outputs from the DAC (which is located on the DE2 board). In this project we will resort to the digital to analogue converter (DAC) which uses a high quality multi-bit oversampling architecture in order to achieve an optimum performance.

The audio controller allows us to have a simple digital audio interface in order to write data into the WM8731. Its inputs and outputs are as follows:

```
module Audio_Controller(
    // Inputs
    CLOCK,                    // System clock input
    reset,                    // Active-high reset.

    clear_audio_in_memory,    // Clears the audio input buffer.
    clear_audio_out_memory,   // Clears the audio output buffer.

    read_audio_in,            // Read enable signal(microphone input)
    write_audio_out,          // Write enable signal(speakers)

    left_channel_audio_out,   //Audio data for playback on the left speaker
    right_channel_audio_out, //Audio data for playback on the right speaker
    AUD_ADCDAT,

    // Bidirectionals
    AUD_BCLK,
    AUD_ADCLRCK,
    AUD_DACLRCK,

    // Outputs

    AUD_XCK,
    AUD_DACDAT
);
```

Figure 6. *Audio_Controller's Inputs and Outputs*

In order to feed data to this controller, we need to set the write_audio_out enable signal to one. This is a level-sensitive input, data will be written on every clock edge while this signal is high. Furthermore, the other enable input signals (clears, reset, read_audio_in) will be set to zero, as we don't need to clear the audio buffers or feed audio into the audio chip, through the microphone.

The rest of the inputs/outputs are off-line chips that need to be connected to their corresponding pin names. The following table presents a brief description of the function of each signal:

Signal Name	FGPA Pin Nº	Description
AUD_ADCDAT	PIN_B5	Audio CIDEC ADC Data
AUD_BCLK	PIN_B4	Audio CODEC Bit-Stream Clock
AUD_ADCLRCK	PIN_C5	Audio CODEC ADC LR Clock
AUD_DACLRCK	PIN_C6	Audio CODEC DAC LR Clock
AUD_XCK	PIN_A5	Audio CODEC Chip Clock
AUD_DACDAT	PIN_A4	Audio CODEC DAC Data

This controller uses the method of PCM to digitally represent analog signals. Pulse code modulation is a method where the amplitude of an analog signal is sampled regularly at uniform intervals, and each sample is quantized to the nearest value within a range of digital sample.

This falls along what we previously mentioned before, our sound wave is represented by a sequence of samples, each sequence with a frequency associated with it. Therefore, the controller has data ports that are 32-bits wide (which will be the size of our sample) and these contain a signed integer that represents one audio sample (provided by the module “GeraSom”).

Therefore, in order to play the desired sound the controller uses the raw PCM data streams we provide him and feeds them directly to the DACs, which convert each value to an analog voltage. This voltage is connected to the Line-out jack on the DE2 board, which drives the speakers and generates sound.

This concludes our explanation about the sound generation of our synthesizer. The following pictures shows a high-level view of these components:

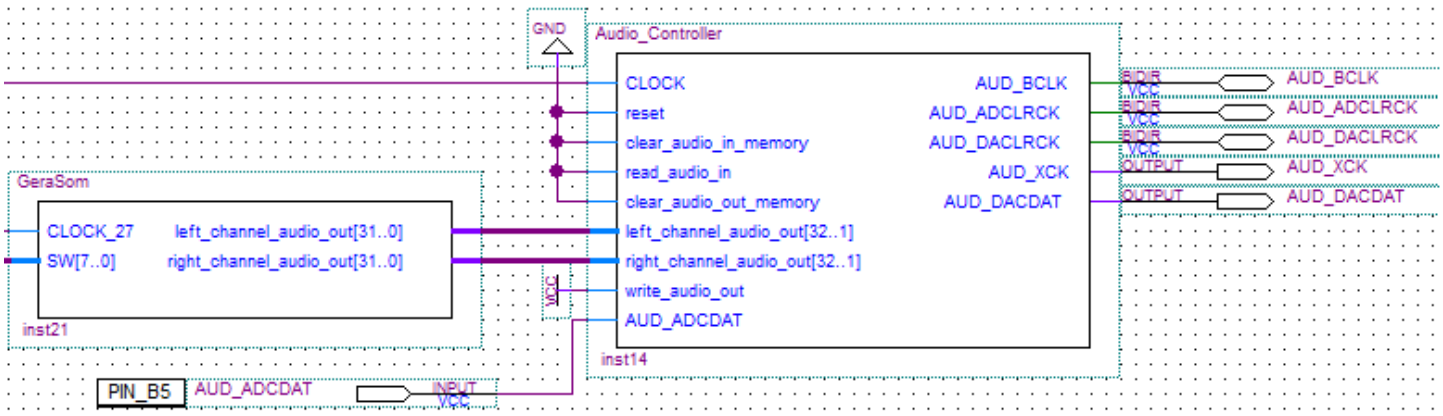

Figure 7. *Connection between "GeraSom" and the Audio_Controller*

3.2. Touch Screen

In this part of the report we will focus our attention on the touch screen, which is one of the main basis of our project. It is responsible for the detection of the coordinates that will later be used to control the volume and pitch of the sound. The touch screen must be connected to the DE2, which has a Cyclone II EP2C35 FPGA by Altera.

We will explore the components and their interconnections, so that it is possible to understand how we projected the work and developed the code.

To begin with, we relied on the following LCD Touch Panel block diagram. This system is divided into two fundamental parts: LCD Touch Panel and LCD Controllers. To better comprehend this, we will use a picture that shows how this system is organized considering these two sections.

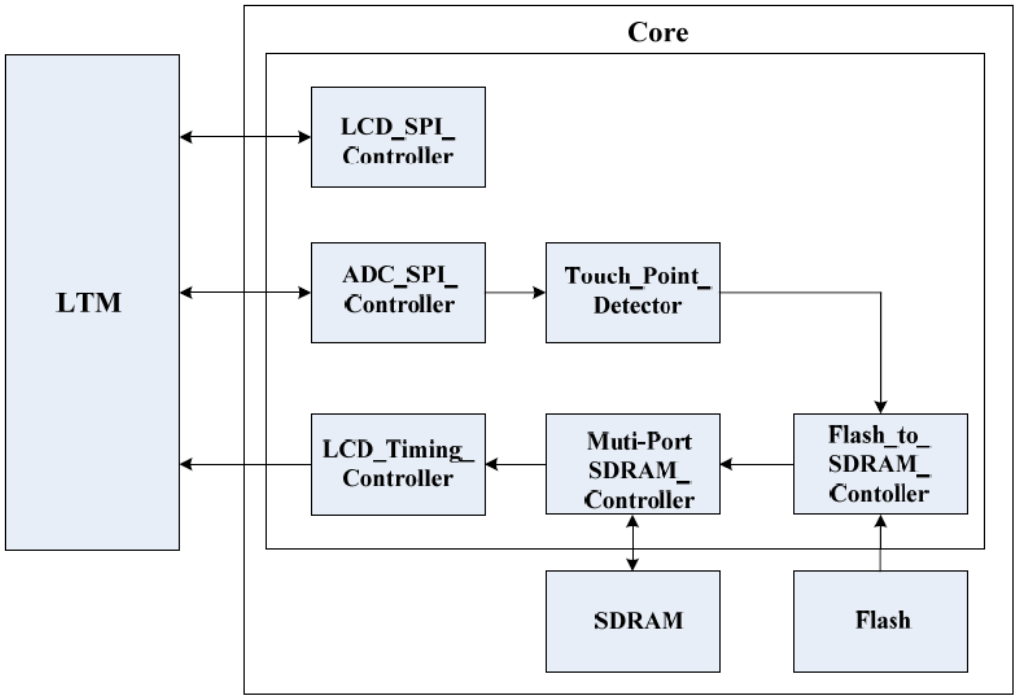

Figure 8. *Block diagram of the LCD Touch Panel Sub-System*

As we can see from the figure, we have on the left side the LCD Touch panel, and on the right side the LCD Controllers composed of the following components: ADC SPI (Serial Peripheral Interface) Controller, LCD Timing Controller, LCD SPI Controller, Touch Point Detector, Flash to SDRAM

controller and Multi-Port SDRAM. These are responsible for the control of the LCD Touch Panel and the data transfer from DE2's SDRAM and Flash memory.

We reused this structure, and changed it accordingly to what we wanted for our project. All the sub-components that we will mention are used. They were all designed in Verilog HDL. We will now discuss them in detail.

First of all, let's look at the LCD SPI Controller. The values of the control registers of the LCD Touch Panel, related with its function, are determined by the LCD SPI Controller. These values are specific and received from a look up table each time the system is activated. It also generates the appropriate signals for the timing of the LCD Touch Panel.

The ADC SPI controller is one of the most important components of our code. It is responsible for the reception of the digital signals from the ADC (Analog to Digital Converter) of the LCD Touch Panel each time an area of the panel is activated by touch. Then, it exports two 12-bit numbers that will represent our coordinates x and y. We will analyze this component later in detail to explain how we developed the code in order to ensure that this functionality is correctly implemented.

The flash memory is a special type of electrically erasable programmable read only memory that can be erased and reprogrammed in block instead of one byte at a time. Flash technology is non volatile, meaning that it does not lose its information once power is removed. This means that if we want to save the photos and do not expect them to be lost when we lose power, we must use this type of memory.

In order for the Flash memory to communicate with the SDRAM, we also have a component named Flash to SDRAM controller. On the flash side we have the address, a write enable, output enable, chip enable and reset. On the SDRAM side we have a write enable control signal and a write signal. This component is useful if one wishes to load different photos that are saved into the flash of the DE2. This block will read the RGB data of one picture stored in the Flash, and then write the data into SDRAM buffer.

The LCD Timing Controller forms the image that it's going to be displayed on the LCD Touch Panel. The process is based on the use of a counter with values from 0 to 800x480-1, which is the resolution of the Touch Panel. For each value of the counter, this controller sends a 24-bit number to be displayed on the correspondent position of the Touch Panel. In our final version, we pretend to use a background image that elucidates the user about the way he needs to proceed in order to do what he wishes to. To achieve this we will rely on this component.

Finally we have a SDRAM 4 port controller, which means that there are two ports: two for reading and two for writing. The information is read from the DE2's SDRAM. For example, through the LCD Timing Controller the 24-bit values that are stored in the SDRAM are displayed in the LCD Touch Panel. Basically, the SDRAM multi port controller enables the communication between the DE2's SDRAM and the LCD Touch Panel.

It is also important to mention that the LTM is equipped with Analog Devices AD7843 touch screen digitizer chip. This is a 12-bit analog to digital converter necessary for the digitizing of the coordinates of the points applied to the touch screen.

So let's now turn our attention into our implementation on Quartus. The following picture shows our main component that outputs the coordinates that we then use as inputs to control the volume and frequency of the sound and displays the background image, the DE2_LTM_Ephoto:

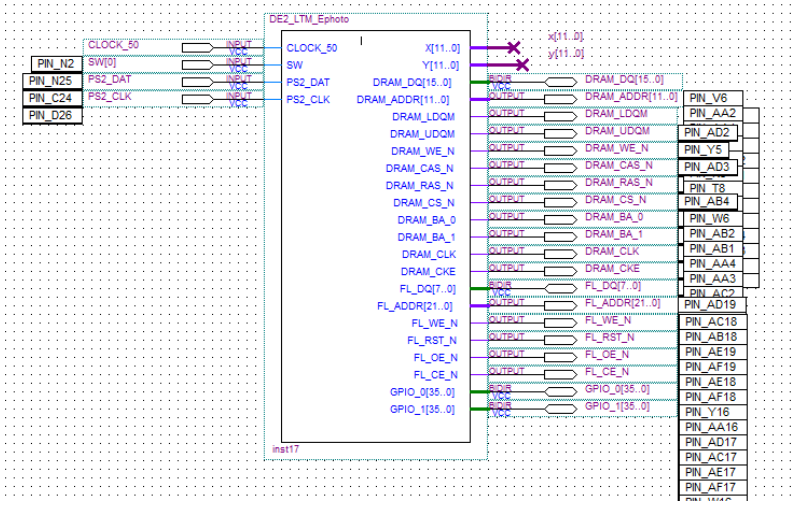

Figure 9. *The DE2_LTM_Ephoto component*

This component had to be changed in the final phase of our project in order to be possible to display the background image. We had previously eliminated outputs that turned out to be necessary to implement this part of the project.

The clocks are connected to their correspondent inputs. It is not visible in the picture, but the CLOCK_27 it's also connected with its correspondent, which is also used for the GeraSom and Audio_Controller.

3.2.1. Coordinates

As we can see, we have x[11..0] and y[11..0] outputs that are then fed into the GeraSom component, responsible for the changes on the desired characteristics of the sound:

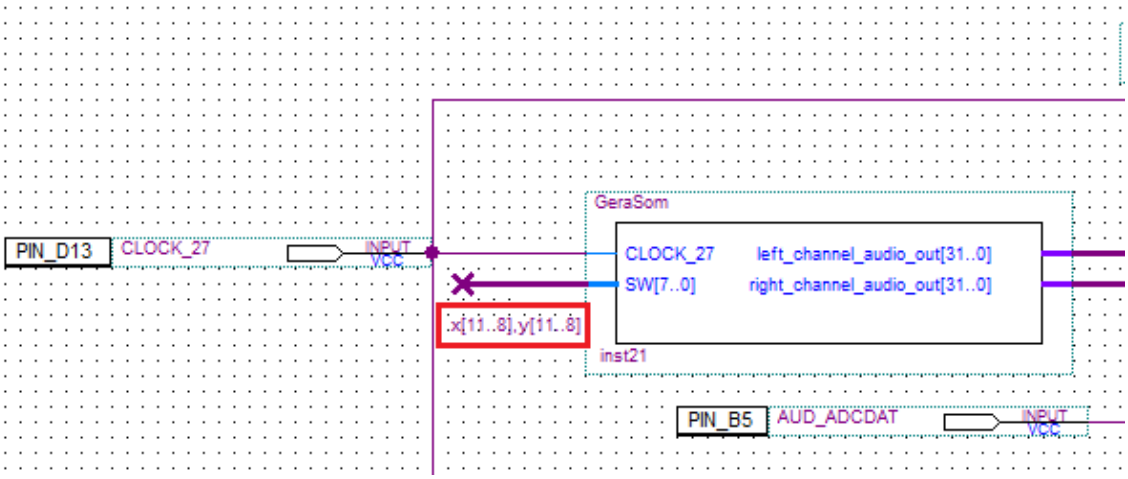

Figure 10. *The coordinates fed into the "GeraSom" component*

The fact that we use 4 bits means that we will have 2^4 different values. We considered this number sufficient and we came to the conclusion that if we used more bits, the frequency would change too fast, causing a misperception that looked like a distortion of the sound.

Also, we choose the bits of greater order of magnitude because if we used the ones of lower order, which would mean that we would have more bits, the changes would happen even faster as we are already considering small distance variations.

Those are the only outputs of this component that are actually being used in our project. They are indeed the bridge between two essential components: the DE2_LTM_EPhoto – responsible for the interconnection between the actions taken by the user on the Touch Panel and the resulting coordinates – and the component GeraSom – which will take these coordinates and apply them on changing the sound accordingly.

All the other outputs are necessary for the correct operation of the system but are not fed into any other component.

After this general approach of the DE2_LTM_Ephoto, we may now explore its interior constitution that involves the controllers mentioned above.

As we said before, the principal controller for what we want to do is the ADC SPI – Analog to Digital Converter Serial Peripheral Interface. The code is the following:

```
211
212   adc_spi_controller  u2       (
213                                .iCLK(CLOCK_50),
214                                .iRST_n(DLY0),
215                                .oADC_DIN(adc_din),
216                                .oADC_DCLK(adc_dclk),
217                                .oADC_CS(adc_cs),
218                                .iADC_DOUT(adc_dout),
219                                .iADC_BUSY(adc_busy),
220                                .iADC_PENIRQ_n(adc_penirq_n),
221                                .oX_COORD(x_coord),
222                                .oY_COORD(y_coord),
223                                .oNEW_COORD(new_coord),
224                                 );
```

Figure 11. *ADC SPI Controller*

The first two lines are with respect to the clock and the reset variables. There is also the ADC_DCLOCK that controls the transfer of information to and from the ADC.

The new_coord variable is a Boolean that, when set to one indicates the existence of new coordinates, and allows the update of the new values on the other components that rely on them.

The ADC outputs an interrupt signal, named ADC_PENIRQ_n which relates to “Pen Interrupt Signal”. When a pull high resistor is connected this output remains high, but when the touch screen is touched by a finger or pen, the value of this variable goes low. Then an interrupt to the FPGA is created, which allows the instructing of a control word to be written to the ADC, and finally obtain the coordinates through the ADC serial port interface.

The ADC_IN (ADC serial interface data in) may take a lot of different values – which form a word represented by the next figure – that correspond to the different phases of the cycle.

MSB							LSB
S	A2	A1	A0	MODE	SER / $\overline{\text{DEF}}$	PD1	PD0

These are:

• S – the start bit. The control word starts with this bit;

• A2-A0 – Channel Select Bits. These are address bit that control, along with SER/$\overline{DEF}$ bit, the setting of the multiplexer input, switches and reference inputs.

• MODE – Bit that controls the resolution of the conversion. When this bit is 0, the conversion has a 12-bit resolution. On the other hand, if its value is 1, the conversion has a 8-bit resolution.

• SER/$\overline{DEF}$ – Single Ended and Differential Reference elect bit – Already mentioned above.

• PD1-PD0 – These are power management bits that decode the power-down mode of the ADC.

The x_coord and y_coord are the actual coordinates detected which are assigned to the variables X and Y that are outputted and then fed into the other component, GeraSom.

Another important module for the output of the coordinates is the LCD SPI Controller. When the bit stream is downloaded into the FPGA, this block will configure the register values of the LCD driver IC using to control the LCD display function.

```
201   lcd_spi_cotroller    u1    (
202                               // Host Side
203                               .iCLK(CLOCK_50),
204                               .iRST_n(DLY0),
205                               // 3 wire Side
206                               .o3WIRE_SCLK(ltm_sclk),
207                               .io3WIRE_SDAT(ltm_sda),
208                               .o3WIRE_SCEN(ltm_scen),
209                               .o3WIRE_BUSY_n(ltm_3wirebusy_n)
210                               );
211
```

***Figure 12**. LCD SPI Controller*

To understand this part of the project it is important to know that the LCD and touch panel module on the LTM is equipped with a LCD driver IC to support three display resolutions. As we can see from the picture, we have a Host Side and a 3 Wire Side. If we open the interior of the module we see that we actually have 20 registers. These are configured by the user through the FPGA via serial port interface and control the following functions: source driver, serial port interface, timing controller and power supply circuits.

Because the SPI is full-duplex, it means that the communication works in both directions at once. As we can see from figure 1, data is received and sent to and from the LTM (LCD Touch Panel).

It is also important to note that, because the number of I/O on the expansion header is limited, the serial interfaces of the LCD driver IC and ADC need to share the same clock (ADC_DCLK) and chip enable (SCEN) signal I/O on the expansion header. So in order to avoid interferences, the chip enable signal (CS), which is inputted into the ADC, will come with a logic inverter – a NOT. This is shown in the following picture.

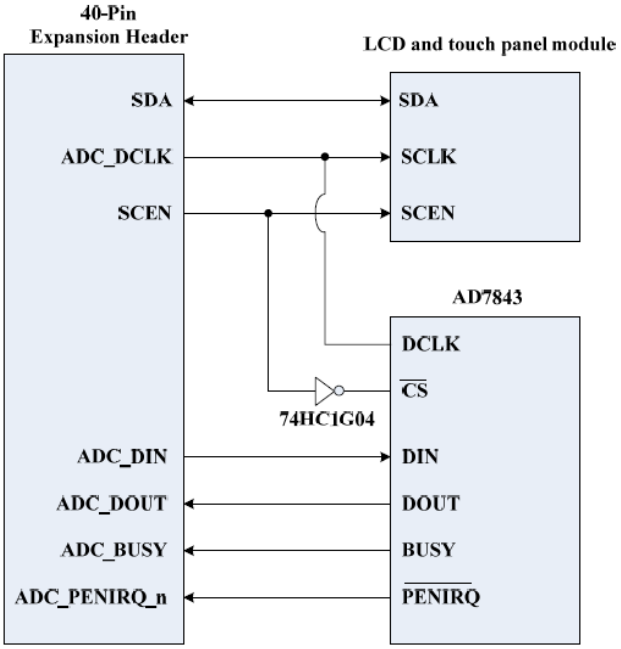

Figure 13. *The serial interface of the LCD touch panel module and AD7843*

3.2.2. Background Image

To improve the quality of our work, we decided to display on the screen a background image that elucidates the user about the influence of the coordinates on the sound. This is the image we used:

Figure 14. *Background Image*

As we can see, the pitch increases from the left to the right and the volume from the top to the bottom.

To do this, we first uploaded the image into the flash memory, and then, using the flash to SDRAM controller, the RGB data of the picture stored in the flash was written into the SDRAM buffer.

To load the photos into the flash we followed a set of steps:

- First, we connected the USB-Blaster download cable into our PC.
- Then we loaded the Control Panel bit stream (DE2_USB_API) into the FPGA.
- The next step was the execution of the Control Panel application software.

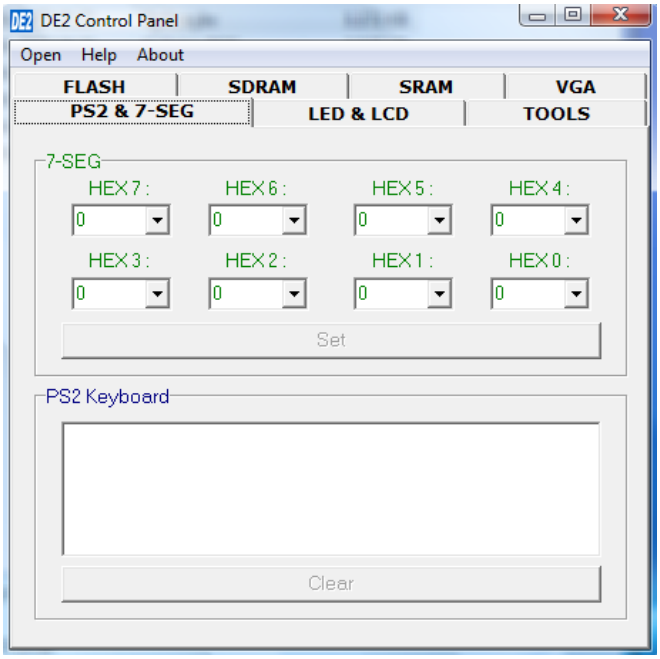

***Figure 15**. Control Panel application*

- We opened the USB port by clicking Open > Open USB Port 0 and all the USB ports that connect to the DE2 board were listed.
- Then we switched to the FLASH page and clicked on “Chip Erase” to erase any information that could remain there from previous utilizations.

- Finally we checked the “File Length” checkbox to indicate that we wanted to load the entire file and clicked on “Write a File to FLASH” and selected the correspondent photo.

Now that we have our image on the flash memory, we must write it on the SDRAM buffer. In order to do so, we have the flash to sdram controller, which has been already mentioned before:

```
217   flash_to_sdram_controller   u4     (
218                               .iRST_n(DLY1) ,
219                               .iF_CLK(F_CLK),
220                               .FL_DQ(FL_DQ)   ,
221                               .oFL_ADDR(FL_ADDR) ,
222                               .oFL_WE_N(FL_WE_N) ,
223                               .oFL_RST_n(FL_RST_N),
224                               .oFL_OE_N(FL_OE_N) ,
225                               .oFL_CE_N(FL_CE_N) ,
226                               .oSDRAM_WRITE_EN(sdram_write_en),
227                               .oSDRAM_WRITE(sdram_write),
228                               .oRED(sRED),
229                               .oGREEN(sGREEN),
230                               .oBLUE(sBLUE),
231                               );
```

***Figure 16**. Flash to SDRAM controller*

This component has an inout Flash data bus (FL_DQ) that data gives us the data from the flash. We read from the flash address until the maximum address is reached. The flash address bus with 22 bits is given by the variable FL_ADDR. In order for us to write the information into the SDRAM the variable SDRAM_WRITE_ENABLE must be one and the variable SDRAM_WRITE, when set to 1, indicates that the values of red, green and blue have been attributed. The values of RED, GREEN and BLUE are given by the information obtained from the data bus FL_DQ and are outputted by the variables sRED, sGREEN and sBLUE.

These are received by the component Sdram_Control_4Port. This component has two FIFOs that connect the Host side to the SDRAM side. It also has a read side and a write side, which allow sending and receiving information. In this case we use this structure to send and receive information about the RGB data of the image.

From the read side we obtain the values Read_DATA1 and READ_DATA2:

```
//  FIFO Read Side 1
.RD1_DATA(Read_DATA1),
.RD1(mRead),
.RD1_ADDR(0),
.RD1_MAX_ADDR(800*480),
.RD1_LENGTH(9'h80),
.RD1_LOAD(!DLY0),
.RD1_CLK(ltm_nclk),
//  FIFO Read Side 2

.RD2_DATA(Read_DATA2),
.RD2(mRead),
.RD2_ADDR(22'h100000),
.RD2_MAX_ADDR(22'h100000+800*480),
.RD2_LENGTH(9'h80),
.RD2_LOAD(!DLY0),
.RD2_CLK(ltm_nclk),
```

Figure 17. *Part of the code from the Sdram_Control_4Port*

These are going to be used by the lcd_timing controller, as can be seen by the next picture:

```
233  lcd_timing_controller   u6  (
234                              .iCLK(ltm_nclk),
235                              .iRST_n(DLY2),
236                              // sdram side
237                              .iREAD_DATA1(Read_DATA1),
238                              .iREAD_DATA2(Read_DATA2),
239                              .oREAD_SDRAM_EN(mRead),
240                              // lcd side
241                              .oLCD_R(ltm_r),
242                              .oLCD_G(ltm_g),
243                              .oLCD_B(ltm_b),
244                              .oHD(ltm_hd),
245                              .oVD(ltm_vd),
246                              .oDEN(ltm_den)
247                              );
```

Figure 18*. lcd_timing_controller component*

The first one gives us the Red and Green color data from the SDRAM and the second one the Blue color data from the SDRAM. This module will then output the LCD Red, Green and Blue color data.

Another thing that is new and wasn't explored before is the function Reset_Delay. This function is not an actual reset of the system because we will still have sound, but it blocks the functions of the LTM, meaning that there will be no new coordinates detected and that the background image will not be displayed. So, if the user wants to maintain the state of the LCD touch panel unaltered, he can turn off the switch. This action will activate the reset delay function that outputs reset variables which are used in all the controllers. When set to zero these variables will cause the controllers to stop their tasks and therefore there will be no outputted coordinates and no image displayed. The sound remains on the state created by the previously obtained coordinates (before the reset).

```
304   Reset_Delay          u8      (.iCLK(CLOCK_50),
305                                 .iRST(SW),
306                                 .oRST_0(DLY0),
307                                 .oRST_1(DLY1),
308                                 .oRST_2(DLY2)
309                                 );
```

***Figure 19**. Reset_Delay module*

So this is how we conclude our explanation of the touch screen. We have now explored some components a little bit further because they were used to display the background picture on the screen. We believe that this was a great improvement of the work that we have previously done.

4. RESULTS ANALYSIS

Looking back to the challenge we accepted some weeks ago, we had two main tasks to do: allow the user to control the frequency and the volume. At first, this sounded like a huge goal we had to achieve.

As the days went by, and we spent more time in the lab, we started to have some ideas and to collect more information that turned out to be very helpful for us. In the beginning every new discovery was just a piece of what we thought of as a big puzzle. But then, we started putting them together and as we did so, we finally begin to build something that, after some time, became a first version of the project and that is now our final version.

We evaluate the work we have done as very positive, because the control of the frequency and the volume are operational, and we manage to implement some more features to elevate the quality of the system. This means that we not only accomplished the milestones imposed, but also improve them by adding the extras that we considered more useful for the understanding of the behavior of the synthesizer. Those are the display of a background image and the reset function that allows the user to maintain the sound in a stable state.

5. CONCLUSION

In an overall perspective we can say that our synthesizer is completed, as the main components regarding the sound alteration and touch screen interaction are well implemented. We manage to go a little bit further than on the previous stage, and developed some more features.

We had to study in detail the functions of the touch screen in order to load a photo into the flash and then display it on the screen. But we believed that we succeed and manage to present an image that we created for this purpose and that demonstrates quite well the functioning of the system.

Definitely, the biggest hardships regarding the project were overcome and we believed that we made some significant improvements regarding the previous version.

6. REFERENCES

- http://www.asic-world.com/verilog/syntax2.html#Ports (verilog)
- http://www.cnblogs.com/oomusou/archive/2008/12/21/trdb_ltm_xy.html
- http://easeserver3.ece.ohiostate.edu:9443/SophExp_Labs/Shared%20Documents/Lab05/Synthesizer.pdf
- http://www.instructables.com/id/VHLD-Photosensitive-Synth-Machine/?ALLSTEPS
- http://www.alteraforum.com/forum/showthread.php?t=4039
- http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/s2013/esh64_er294_as885/esh64_er294_as885/