The Sound Disguiser™

09-10-19

Linus Tufvesson (dt06lt6@student.lth.se),
Nicklas Nidhögg (dt06nn8@student.lth.se) and
Nadir Khan Muhammad (sx08nk0@student.lth.se)

Table of contents

I. Abstract

The idea is to be able to get a sound via the input microphone, distort it and output it through a speaker. To be able to do that a system was created with a FPGA, microphone and an amplifier. When running the system the amplifier will output a distorted sound considering the chosen distortion effect. The different effects that were to be implemented were echo and robotic sound. The implementation of a rotary button would mean the addition in being able to modify the effects such as adjusting the pitch (high to low) and/or the interval of the echo as well as the selection of distortion effect.

II. Introduction

A. Background

The world in 2009, as technology advances mankind becomes more adapted by it. A government trying to oppose terrorism by terrorizing the human rights of their own citizens by eavesdropping on them. In order to antagonize the government the world is in great need of a voice disguiser. A voice disguiser that can make you a robot or achieve great confusion by echo effects.

A sound disguiser is a device that disguises your voice for someone or something else. A lot of effects were considered while planning the project. Effects such as echo, robotic sound, voice of another live person without having to be a bad improviser yourself. The user interacts with the system through the microphone and gets feedback through the headphones or speakers connected to an amplifier.

B. Design

The system was built with several peripherals (Pmods) and a FPGA. The Pmods used was a microphone (PmodMIC) and an amplifier (PmodAMP1). The FPGA was a Xilinx Spartan-3E.

At the planning stage the idea included implementing a rotary button (PmodENC) used for the interval between the echoes, high or low pitch and for choosing between the different modes. This wasn't implemented due to memory limits. To have the rotary button increase the delay of the echo, the memory would have to be significantly increased and the idea of the project was to create a small and resource efficient. Therefore the rotary button was not implemented according to plan.

The Xilinx Embedded Development Kit (EDK) is a suite containing tools and Intellectual Property (IP) and was used to design an embedded system for implementation in a Xilinx Field Programmable Array (FPGA) device.

Xilinx Platform Studio (XPS) is a development environment which was used in designing the hardware system. XPS has the specification of the microprocessor, peripherals and the interconnection between the components. XPS was also used in simulating the system via the Hardware Description Language (HDL) simulator.

This report will describe the development of the above defined system.

III. Sound Effects

In the project two sound effects are implemented which are Delay and Robot sound.

A. Delay

In this effect a previous sample is added with the current sample and than average become new sample. The effect can be made better by taking more contribution from current sample than the previous sample like we did in our project. We are taking 66% contribution from current sample and 33% contribution from previous sample. Delay is shown in figure below.

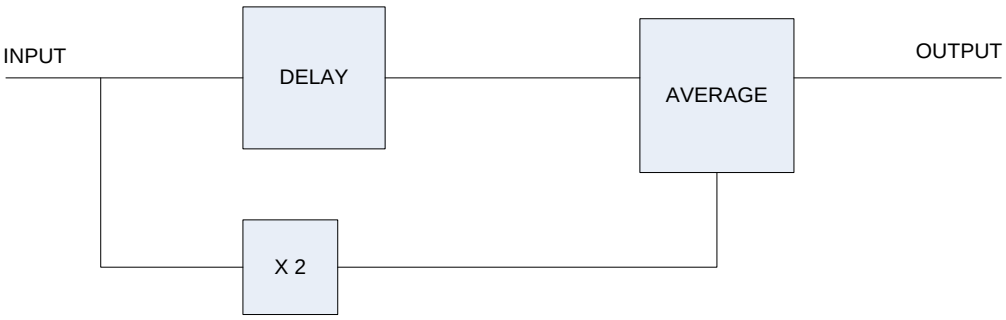

Figure 1. Block diagram for delay effect

B. Robot sound

Making robot sound is similar to delay. The only difference is that we have to add a previous sample in a dynamic way. First we created a Matlab model for this and basically what the Matlab model does is that it adds a previous value with current value. How big the delay of the previous sample is will increases from 0 to fs/20 and then decreases to 0. This increase and decrease spans fs/2 samples.

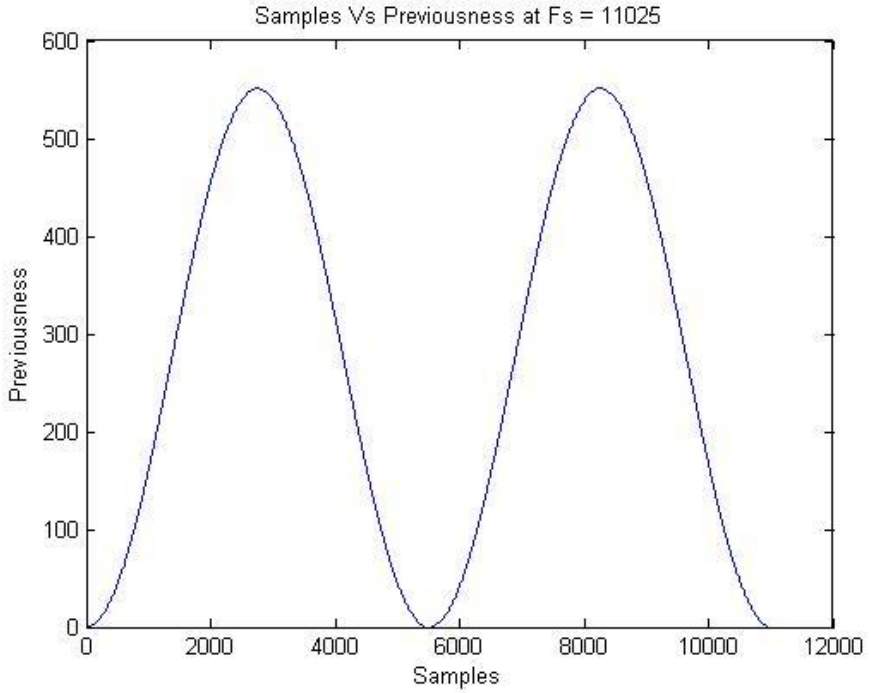

Figure 2. Graph of sample Vs previousness (cos function)

The curve shown in above figure is shows how robot effect works. In the graph x-axis represents current sample we are dealing with and y-axis shown how much previous sample we are combing with the current one. When the sampling frequency is 11025, how old the sample has to be added to current sample increases from 0 until 2756 and then it decreases until it reaches 5512 sample.

From hardware point of view curve which is shown in above figure is more resource consuming so we changes the curves to the curve shown in figure below.

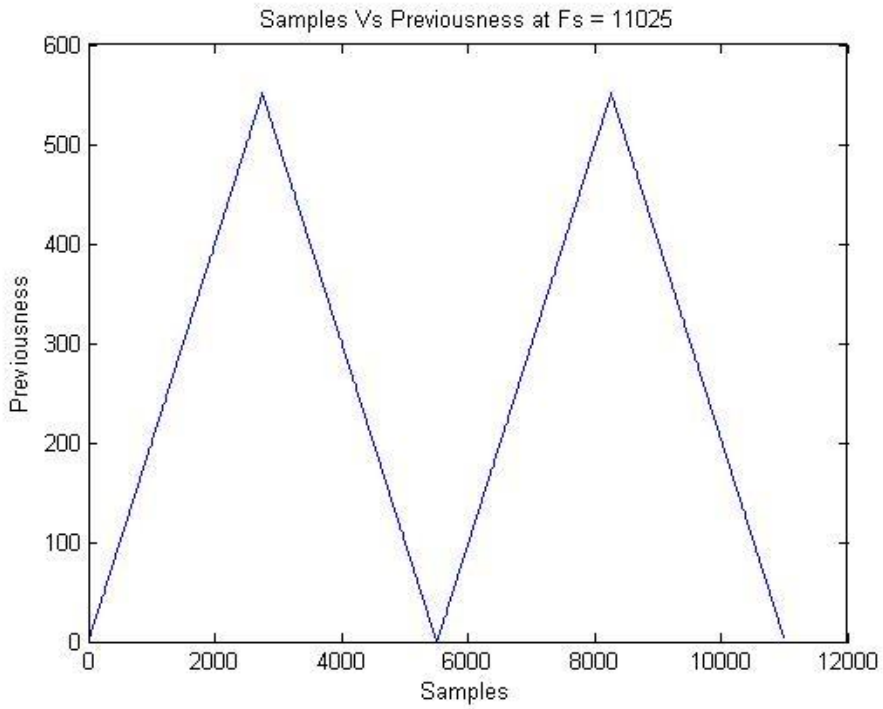

Figure 3. . Graph of sample Vs previousness (linear function)

With this new curve the resulting robot sound is similar to original one and this curve is very simple to implement and there is not much difference in sounds.

IV. Software

On the software side only one thread is running. The thread is sampling the microphone continuously. Once a sample is obtained it is down sampled to 8-bits. The samples are sent to the hardware through an fsl-bus. The buffer in the fsl-bus is what's keeping the sampling rate synchronized with the speed that the amplifier outputs the sound. When the buffer in the bus is full, the function call that puts more data on the bus blocks the thread until the hardware is done with processing the old samples.

The communication with the PmodMIC is done with xps-spi module. The reason that the microphone is sampled using software instead of hardware is that we wanted to learn how to communicate with Pmods both using software and hardware. In the beginning of the project the plan was to do the sound effects in software and when it was decided that they were to be done in hardware the microphone sampling was already running with the software solution.

The PmodMIC was connected to the board via MOSI (master out slave in), SS (Slave Select) and Clk. The xps-spi peripheral took care of the data management from the connected pins of the PmodMIC and made them available to the software via the interface.

When the SS goes high the microphone starts to sample and send the data on the MOSI-pin.

V. Hardware

The hard part starts after the FSL link. Sound samples are taken from microphone and then those samples are put on the FSL link. The hardware part takes those samples and does the processing. We have the FSL link which give sample to the main FSL block and FSL block is the main architecture.

A. FSL link

When data is written on FSL it makes the EXIST signal high and put the data on DATA_IN, and data stays there until it gets acknowledgement from the component.

The hardware architecture is shown in figure below.

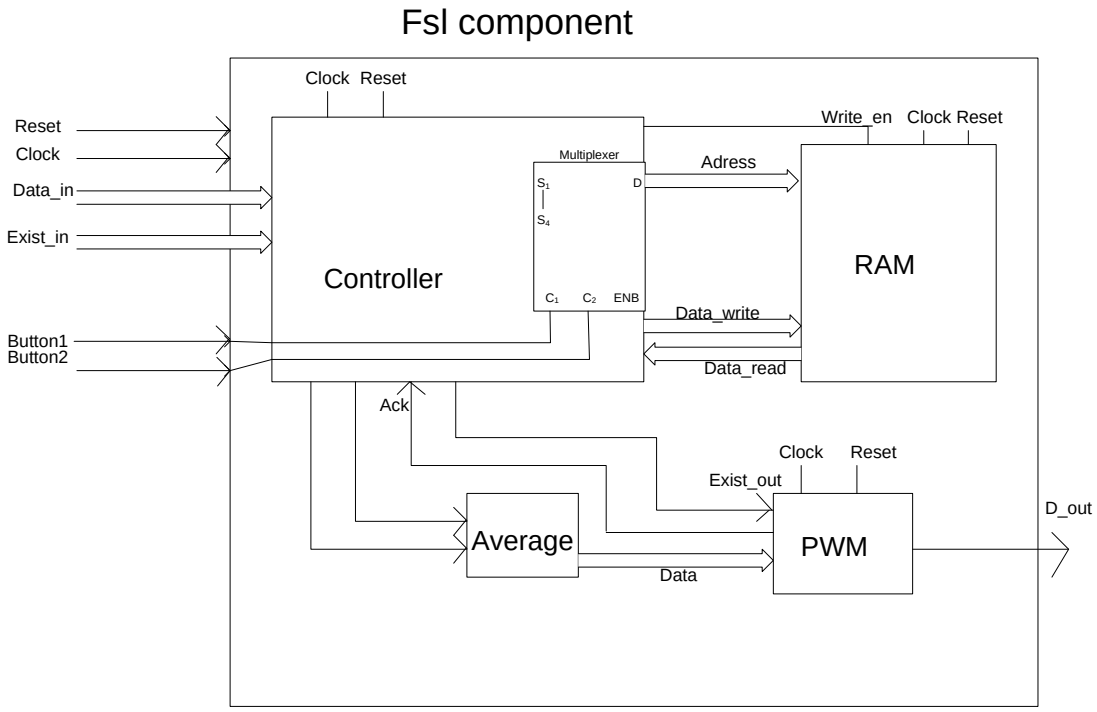

Figure 4. Hardware Architecture

The FSL component can be seen in the block diagram which shows the different components namely Controller, RAM, Average and PWM.

B. Controller

Controller is basically a state machine which can be shown in figure below.

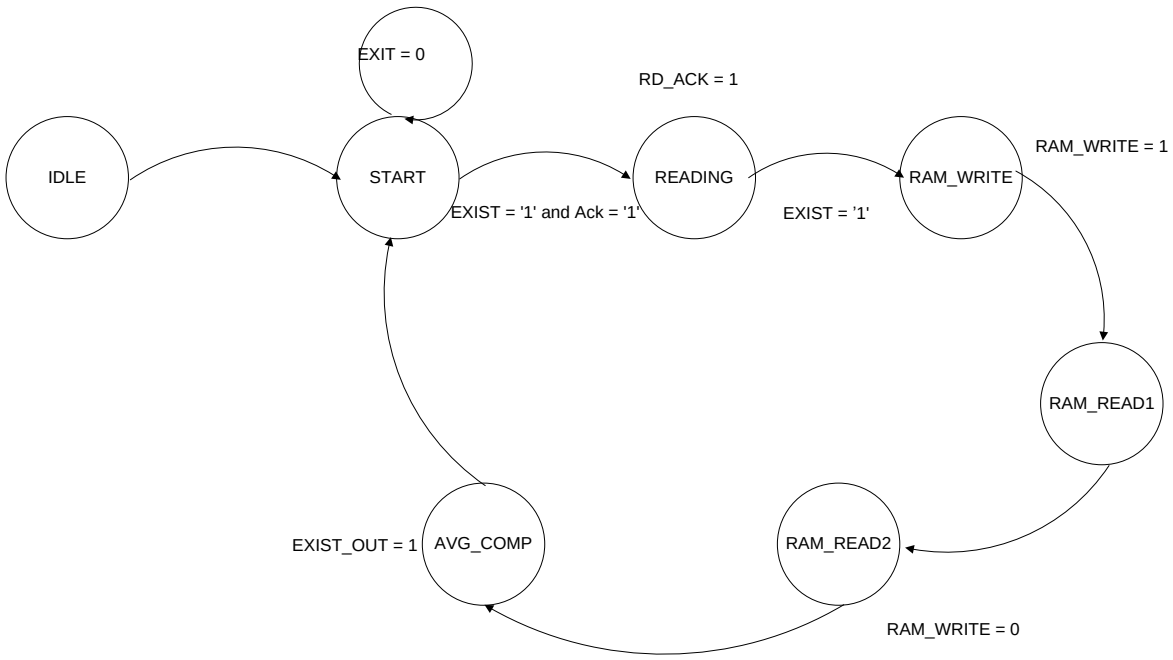

Figure 5. State machine of controller

Above state machine shows how controller works, it starts from IDLE state where all the signals are initialized and it goes to START state. It stays there until EXIST and ACK signals are not high. As it is already mentioned when FSL link has data, EXIST signal is made high. ACK signal is basically between PWM and Controller. Controller stays in start state until PWM component is not free. Once both the data is on the FSL link and PWM is free, controller moves to the READING state where it reads data from the link. Then it goes to RAM_WRITE state where it stored that sample value in the RAM component by making the RAM_WRITE signal high. In the next two states controller just reads previous data from RAM. Then it goes to the next state which is AVG_COMP. in this state controller give current and previous sample to the AVERAGE component. And make the EXIST_OUT signal high which is connected to PWM component.

C. AVERAGE

This component takes current and previous samples from the controller and multiplies current sample with 2 and adds it with previous sample and then divides the sum with 2. The output is given to PWM.

D. PWM

PWM is also a state machine which can be seen in figure below

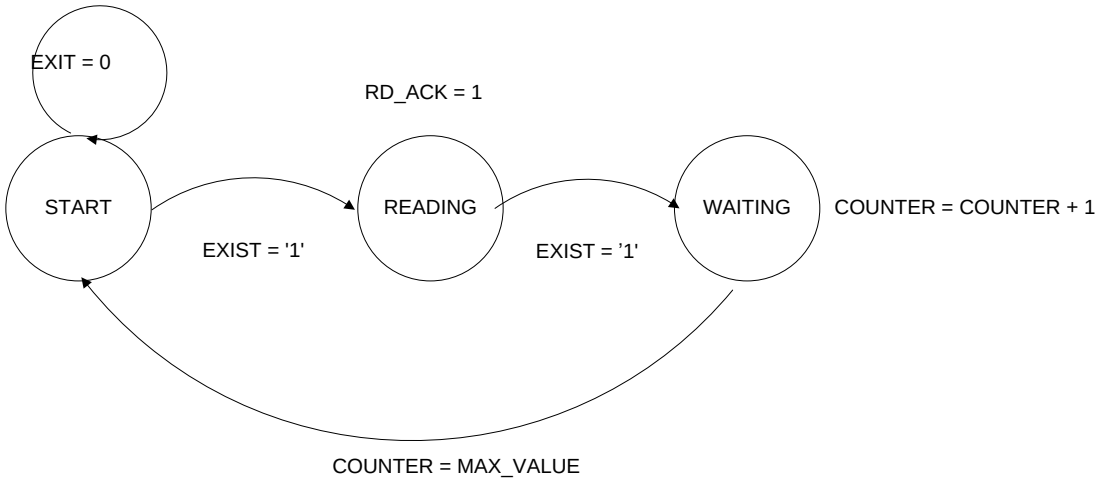

Figure 6. state machine of PWM

State machine starts from START state where signals are initialized, and it stays in this state until EXIST is high. EXIST signal tells PWM that data is available, then it goes to READING state where it reads data from AVERAGE and then it goes to WAITING state. In WAITING state it starts COUNTER and stays in this state until COUNTER reaches it maximum value.

PWM component is basically generating pulse with different widths of level 1 and 0 depending upon the value it gets from AVERAGE component. COUNTER's maximum value is the number of cycles occupied by one square pulse and by adjusting the width of this counter, sampling frequency of the output is controlled. PWM component is working on 50MHz, and suppose the width of signal Y is 10bits than maximum it can have is 1024. It means one sample will occupy 1024 cycles, which means only 50MHz/1024 = 48828 samples can be given to speaker. If we increase the width of Y more the sampling frequency will decrease

Suppose COUNTER signal's maximum value is Y and we get a value X from AVERAGE component, the PWM will make D_OUT signal high for X clocks and then low for Y – X clocks. This happens in WAITING state.

Normally sample value is scaled up or value Y is scaled down if Y has a high width.

E. RAM

In this architecture FPGA's RAM is not used rather RAM is created in VHDL. RAM is an important component because it occupies a lot of space. For our design we don't need much RAM. For delay effect we only need fs/3 previous values, because we want to hear 0.33 sec delay. For Robot effect, we even need less RAM. If you see Figure 1 it is clear for fs = 11025 we just need 521 samples which means for Robot effect we only need fs/20 samples.

As our fs = 48,000, we are using 16,384 byte RAM whose address bus has a width of 14 bits and data bus has a width of 8 bits.

F. Logic Utilization

Total Number Slice Registers:	1,684 out of 17,344	9%
Number used as Flip Flops:	1,665	
Number used as Latches:	19	
Number of 4 input LUTs:	2,491 out of 17,344	14%

G. Logic Distribution:

Number of occupied Slices:	1,923 out of 8,672	22%
Total Number of 4 input LUTs:	2,577 out of 17,344	14%
Number used as logic:	2,057	
Number used as a route-thru:	86	
Number used for Dual Port RAMs:	256	
Number used as Shift registers:	178	
Number of RAMB16s:	15 out of 28	53%
Number of BUFGMUXs:	1 out of 24	4%
Number of DCMs:	1 out of 8	12%
Number of MULT18X18SIOs:	3 out of 28	10%

VI. Problems encountered

- Lack of knowledge on how to communicate with the Pmods.
    - After looking through several different example codes we finally figured out what and how we needed to implement things. When we started the course we had zero knowledge on how the communication with the Pmods worked. It took us the first couple of weeks to get them running at all. The example codes that finally got our microphone running were http://bears.ece.ucsb.edu/class/ece253/ECE253F09_LAB2.zip (see spi.c, spi.h and xil_xspi_l.h) and we got the amplifier to work after we looked at **<NADIR LINK HERE>.**
- To low sampling frequency gave a ringing noise
    - The reference pwm implementation that we used as a base for our own pwm implementation was generating 12 kHz samples. This was to slow and caused the signal to generate a high pitch sound.
- Not synchronizing the pwm and controller.
    - The Software part was running at an unknown speed sampling, 'as fast as possible', putting data on the fsl bus. The controller was constantly taking the data from the fsl bus, saving it in memory and making it available to the pwm. The pwm was running at a constant speed of 24 kHz and thus not using all the samples from the microphone. We added a synchronization part so the controller wouldn't get a new sample from the fsl-bus until the pwm had done a read on the memory and since the pwm was reading at a rate of 24 kHz the controller was now asking the fsl-buffer for 24 k samples per second, wich in turn led to the software side only being able to put 24 k samples per second on the bus. Once we did this we noticed an increase in the echo effect.
- Version control
    - We had some trouble keeping everything synchronized. We tried to use svn for the project but it would not synthesize when we moved the project between different computers.

VII. Installation Manual

The project files can be downloaded at pix.sonhult.se/Soundthing.zip

The default setup is:

- Xilinx Spartan 3E board,
- PmodMIC connected to the upper peripheral (Pmod) connector JA1,
- PmodAMP1 connected to the upper Pmod connector JB1.

The default setup is the recommended setup; Sound Disguiser™ takes no responsibility whatsoever in our implementation and therefore all downloaded code is used at one's own risk.

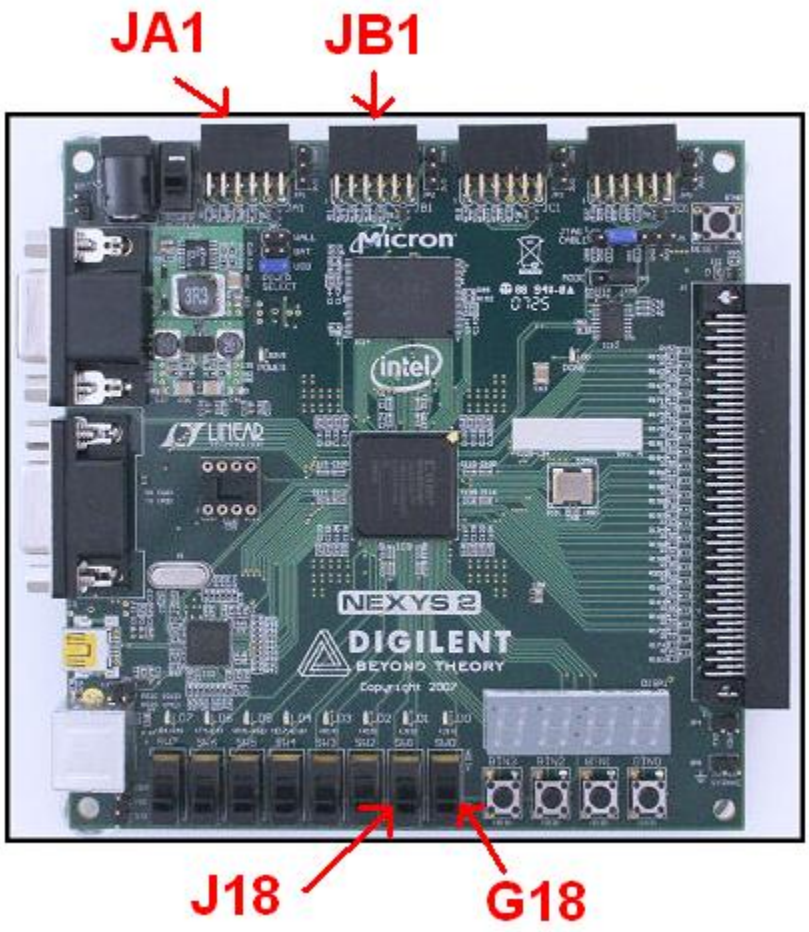

Figure 7 – Showing PINs and Pmod connectors

If any changes needs to be made to the hardware in the project due to a different setup etc. the project must be re-synthesized. If it was not, skip this part. To synthesize the project press “Update Bitstream” found under the “Device Configuration” menu.

A. After synthesizing

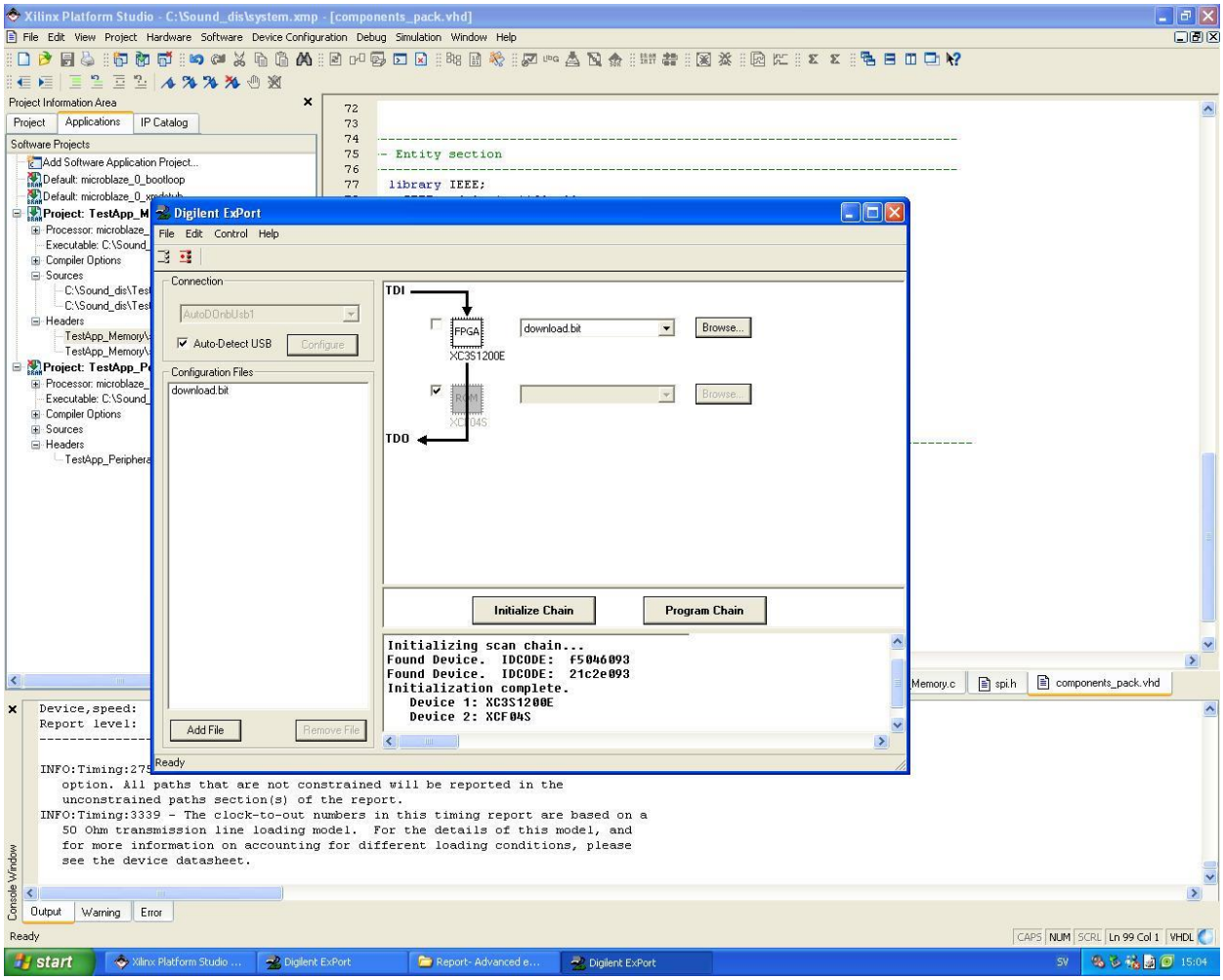

Figure 8 - Showing ExPort settings

The board must be turned on, connected through the usb port and the VGA port.

Start the “ExPort” found under “Start menu → Xilinx ISE design Suite 10.1 → Adept(Diligent)”.

Check “Auto-Detect USB” under connection, press “Initialize chain”, check to bypass the “PROMs”, press “Browse...” and choose the download.bit found in the implementation folder in the project folder.

B. Device running

Changing distortion mode can be made combining the switches named (G18, H18).

Effect	**G18**	**H18**
Robotic sound	On	On
Delay	On	Off
Normal	Off	On
Normal	Off	Off

VIII. Lesson Learned

The major lesson learned was how to communicate between the peripherals and the FPGA. We realize that using the xps-spi peripheral and a processor was unnecessary since only using hardware for everything would decrease the power consumption, the utilization of the board and cost if the system was to be mass produced since a processor would not be needed.

Matlab was a great tool for testing and modulating our sound distortion algorithms. We were able to input a sound file and get direct feedback via Matlab as mentioned.

Developing hardware is vastly different from developing software in the sense that you can't hit and miss as much as you can with software. Since every recompiling will take 10-20 min trying to tune a variable empirically is not viable.

If we were to redo the project we would probably do a stricter partition into small hardware components and design and test every component individually.

IX. Contributions

Nadir Khan has been in charge of the hardware design of the project, mainly assisted by Linus Tufvesson. Software design and implementation was done by Linus and Nicklas. The soundeffects modeling and implementation was done by Nadir Kahn. The different parts of the report are written by the person in charge of that area.

X. References

1. http://www.digilentinc.com/Support/Support.cfm?NavTop=85 (Reference implementations)
2. http://www.cs.lth.se/EDA385/
3. The Built in help in xlinks