ECE532 Digital Hardware

Real-Time Audio Processing

Group Report

28th March, 2005

Ajmal Khan 990843735
Mohammad Tariq Rafique 991016734

Table of Contents

TABLE OF CONTENTS **2**

1 OVERVIEW **3**

1.1 PROJECT GOALS 3
1.2 SYSTEM OVERVIEW 3
1.3 BLOCKS OVERVIEW 4

2 OUTCOME **5**

2.1 FURTHER WORK & IMPROVEMENTS 5

3 BLOCK DESCRIPTIONS **5**

3.1 AC97 CONTROLLER 5
3.2 FFT/IFFT 5
3.3 EQUALIZER 8
3.4 FFT CONTROLLER 10
3.5 ECHO 11
3.6 MICROBLAZE [5] 11

4 DESIGN TREE **12**

5 REFERENCES **13**

1 Overview

1.1 Project Goals

The objective of this project was to implement Real-Time Audio Processing capability in the frequency and time domain using the Xilinx Multimedia Board. For frequency domain processing, we would provide the ability to manipulate the gains of select frequencies (an equalizer) and for time domain, the ability to generate an echo.

1.2 System Overview

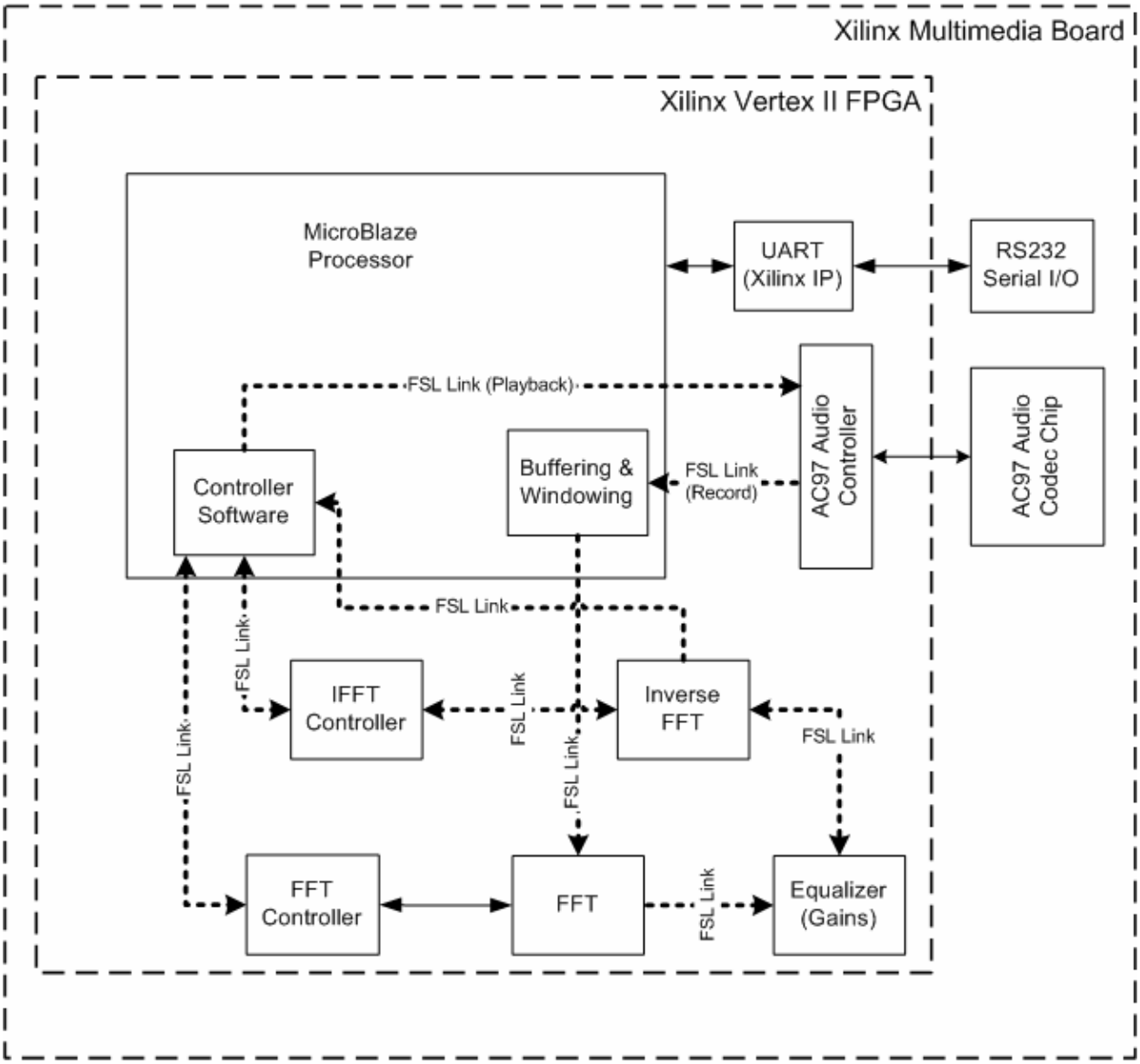

Figure 1 – System Block Diagram, excluding Echo Block

The AC97 is used to capture audio data, which is then buffered on the MicroBlaze until sufficient samples have been accumulated to complete a frame. The data is then sent to

the FFT and the FFT Controller is used to initiate the Fourier transform from the MicroBlaze. The equalizer multiplies the frequency coefficients from the FFT block and then forwards the data to the IFFT input buffer. Once this is complete, the MicroBlaze signals the IFFT Controller to begin the inverse operation. Upon completion, the MicroBlaze reads back the data and sends it to the AC97 controller for playback.

Note that Figure 1 does not include the echo capability, since we were unable to incorporate the echo block into the complete system due to lack of time. The diagram below shows how the echo block was implemented on a separate system.

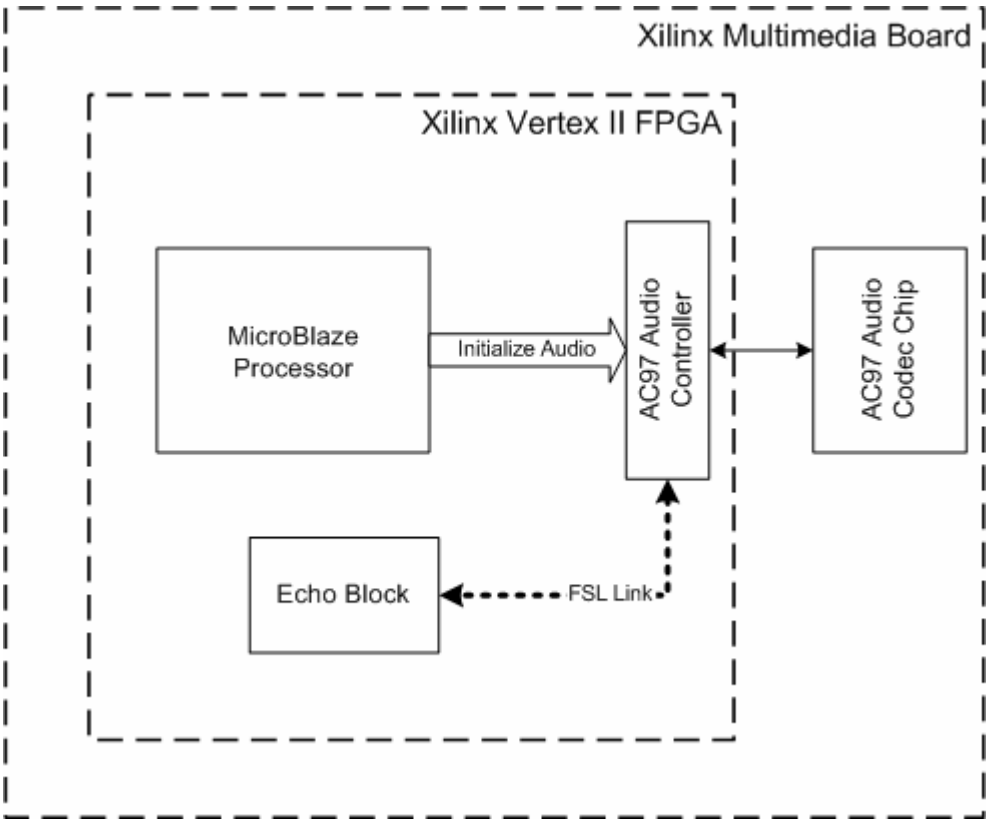

Figure 2 – Block diagram of echo block implementation

Here the echo block was connected to the AC97 directly using the FSL bus. The MicroBlaze was simply used to initialize the AC97 Controller through the OPB.

1.3 Blocks Overview

The AC97 Audio controller was code modified by students taking ECE532 in 2004. It allows for playback and recording using the FSL bus.
The FFT & IFFT blocks provide FSL interfaces to the Xilinx implementation of the Fourier Transform and the inverse. These blocks ensure that there are enough samples to the do the transforms and provide a mechanism to extract the data once the transforms are complete.
FFT & IFFT controllers simply send initiation commands to the FFT & IFFT blocks through the MicroBlaze and read back scale factor information.

The Equalizer applies gains to the frequency components generated by the FFT.
The Echo Block provides applies an echo effect to the sound currently being played back.

2 Outcome

We are able to play back sound using by taking the Fourier transform of the incoming data and then applying the inverse transform. The sound is sometimes distorted as if the AC97 playback was saturating. This is because the FFT and IFFT apply scaling factors to utilize the full dynamic range available from 16 bits.
We have been able to extract the scaling factors from the FFT and IFFT, but are unable to utilize them to scale the audio.
The equalizer has been implemented, but can only apply fixed gains that are set at compile time through the block ram initialization file in ISE.
The Echo Block has been implemented separately but has not been integrated with the whole system due to Block RAM size restrictions on the FPGA.
The User Interface to the project has not been implemented.

2.1 Further Work & Improvements

We need to effectively scale the data being received from the FFT and IFFT blocks so that the AC97 output does not saturate and that the sound plays at a constant volume.
Currently the MicroBlaze is being used for transferring data and working with mono audio. In the future we can connect all the components directly through the FSL bus. This would free up the MicroBlaze for other tasks such as managing the Graphical User Interface. This user interface can be implemented through a monitor and a mouse connected to the PS/2 port on the multimedia board.
To overcome the Block RAM restrictions on the FPGA we can use external ZBT memory to access the echo data.

3 Block Descriptions

Our project used several IP blocks, which are detailed below.

3.1 AC97 Controller

The AC97 controller was used to provide an FSL interface to the onboard audio codec. It was configured to record and playback using the FSL bus. The block was written by students taking ECE532 in 2004 and can be found on the ugsparc system at /nfs/ugsparcs/cad/cad/cad2/ece532s/2004projects/ac97controller-driver/design_fsl.zip.

3.2 FFT/IFFT

The FFT and IFFT block were used to transform the audio stream to/from the frequency domain. These blocks were written using Xilinx's FFT core v3.1 [1]. The blocks act as an interface to Xilinx's FFT core and their functionality is summarized in Figure 3 and Figure 4.

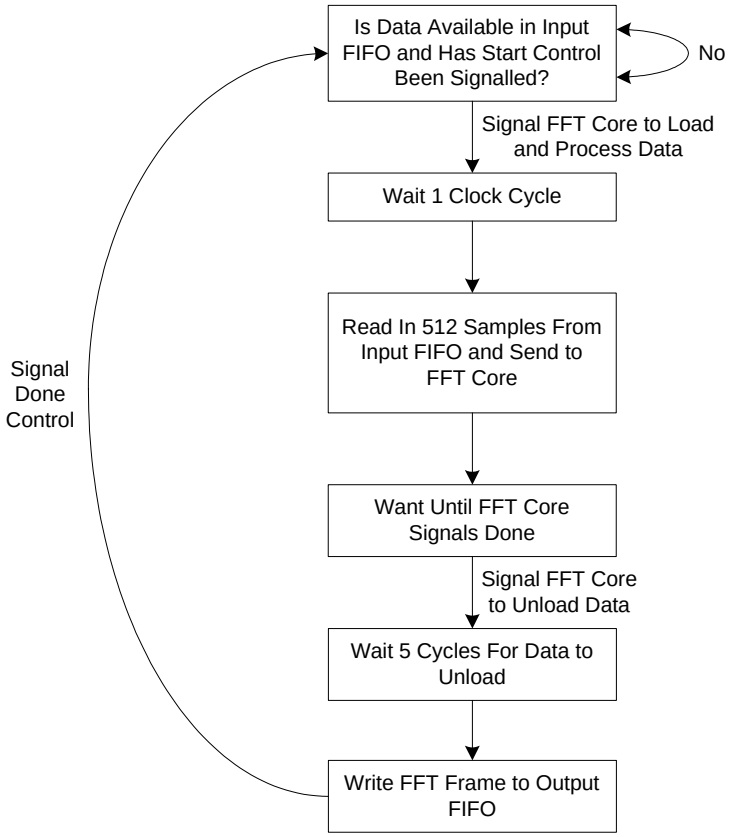

Figure 3 – Functionality of FFT block.

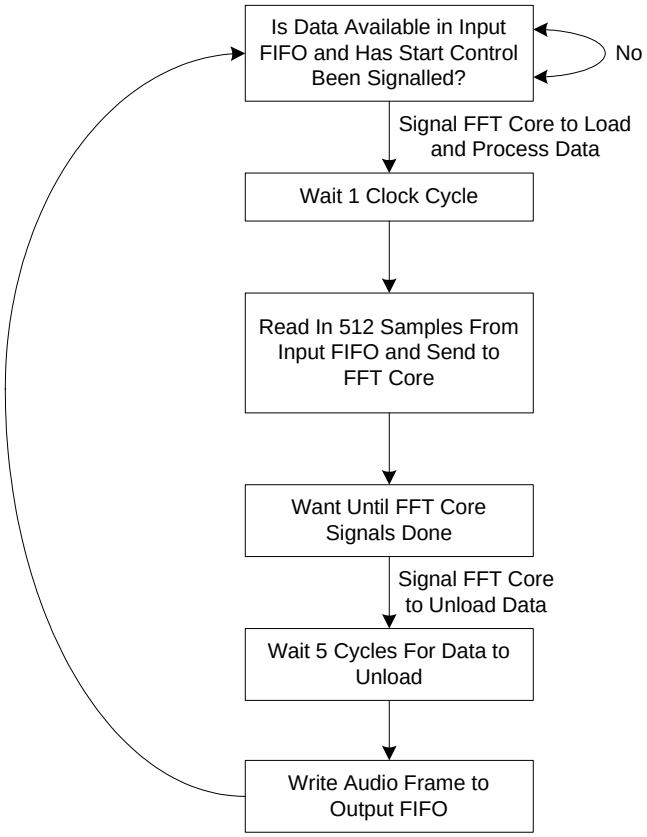

Figure 4 - Functionality of IFFT block.

As can be seen, the only difference between the FFT and IFFT is the external control signaling done. Since the FFT core requires a 16-bit real and imaginary component every clock cycle, it was decided to use the 32 bits of the FSL bus to store the real and imaginary components of the FFT. The real component was stored in the upper 16, and the imaginary in the lower 16. That way, each element in the FSL contained both the real and imaginary component of the current sample and both could be read out in 1 clock cycle.

The FFT core provided by Xilinx comes in 3 flavours: pipelined (the fastest and most resource consuming) and radix-4 and radix-2 (the slowest and least resource consuming), both of which are meant for burst mode processing. We used the radix-2 FFT and generated it using the parameters in Table 1.

Parameter	Value
Transform Length	512
Input Data Width	16
Phase Factor Width	16
Optional Pins	None
Scaling Options	Block Floating Point
Rounding Modes	Convergent Rounding
Memory Options - Data	Block RAM
Memory Options – Phase Factors	Block RAM

Table 1 - Parameters used to generate the FFT core.

In order to use this block in XPS, minor changes had to be made to the block data files. The MPD file had to be changed to include the STYLE = MIX option, which instructs XPS that this block uses both netlists and HDL. Also, a BBD file was created which listed the two netlist files used by the block that were generated using ISE.

Before implementation, the block was tested using an HDL test bench in ModelSim. The FFT and IFFT block were tested separately. We tested using a constant value as input to the FFT and compared the output with the expected value of only DC. The resultant DC output was used as input to the IFFT and compared with the original constant input. Also, alternating values of 32767 and –32767 were FFT'ed to make sure no overflow issues occurred and the result was compared to MATLAB. The results verified the successful operation of the block, however it was noticed that the output was scaled incorrectly. From the simulation, it was realized that the blk_exp output from the FFT core could be used to scale the resulting output correctly. The blk_exp reports how many bits the input to the current frame was shifted right. We know each input frame must be divided by 512, or shifted right 9 bits. Thus, we were able to use the amount the FFT and IFFT shifted by and compensate the output accordingly.

Upon implementation of the block in hardware, we realized that it was not working correctly. We were able to take the output of the FFT and used MATLAB to invert it to determine what signal the FFT block transformed. We found out that the FFT block had transformed an input that was delayed by 1 sample, which indicated that the block was sending input data to the FFT core 1 cycle too late. However, this contradicted the results of our simulation, which indicated correct timing. This may be a bug in the FFT core simulation files or related to some other issue that we are not aware of. We were able to fix the problem by sending data to the FFT core 1 cycle sooner.

3.3 Equalizer

The equalizer block performs the processing on the FFT data. Each frequency bin of the input is multiplied by a gain that is variable. In order to allow fractional multiplication, the gain used was a fraction of 2. For example, if the gain is 4, the input is multiplied by 4 and then shifted right 1 bit, to provide a total gain of 2. This provides gain increments of 0.5, which was deemed suitable for our purposes. The block uses Xilinx's Dual Port Block Memory v6.1 [2] and Multiplier v7.0 [3](note, referenced documentation is for older version of core, since newer version is not available online) cores. The dual port memory is used to store the gains to apply to the FFT frame. Dual port memory was used so that the gains could be updated without interfering with the multiplication process. Two multiplier cores were used to perform the multiplication for both the real and imaginary components of the input frame. The functionality of the block is summarized in Figure 5.

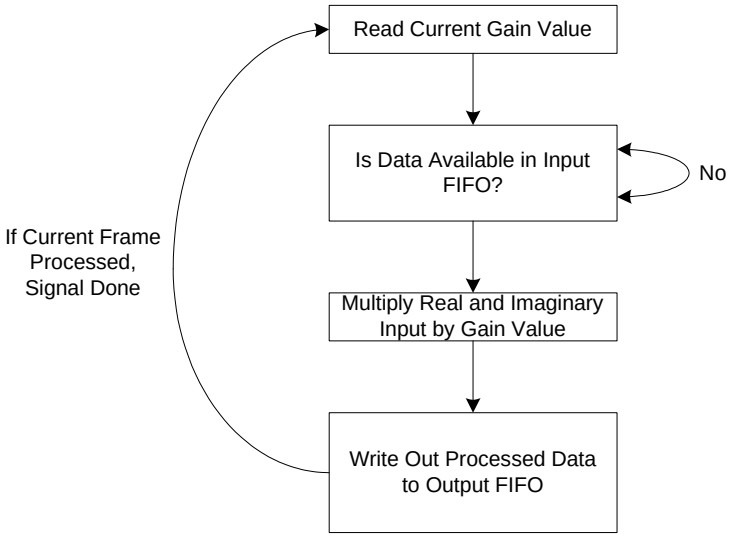

Figure 5 - Functionality of equalizer block.

The dual port memory was configured with the parameters in Table 2.

Parameter	Value
Memory Size – Width A	16
Memory Size – Depth A	512
Memory Size – Width B	16
Port A Options	Read Only
Port B Options	Write Only, No Read On Write
Port A Design Options	Handshaking Pins
Port A Design Options – Pin Polarity	Rising Edge Triggered, Active High
Port B Design Options	Handshaking Pins
Port B Design Options – Pin Polarity	Rising Edge Triggered, Active High
Primitive Selection	Optimize For Area
Simulation Model Options	Disable Warning Messages
Initial Contents – Global Init Value	2

Table 2 – Parameters used to generate the dual port block memory core.

The multipliers were configured with the parameters in Table 3.

Parameter	Value
Multiplier Type	Parallel
Multiplier Type – Multiplier Construction	Use 18x18 Multiplier Blocks
Multiplier Type – Virtex II Multiplier Optimization	Speed
Layout	Create RPM
Layout - Shape	Rectangular Shape
Input Options	Register Inputs
Input Options – Port A - Width	16
Input Options – Port A – Data Type	Signed
Input Options – Port B – Width	16
Input Options – Port B – Data Type	Unsigned

Output Options - Width	32
Output Options	Registered
Handshaking Signals	ND, RFD, RDY
Pipeline	Minimum Pipelining

Table 3 - Parameters used to generate the multiplier core.

Since we are multiplying the input, it is necessary to check for overflow after applying the gains. Checking for overflow was done in two cases: for negative numbers, if bits 31 to 16 are not all 1's, overflow has occurred, in which case the output is set to the largest 16-bit negative number, -32768. For positive numbers, if bits 31 to 16 are not all 0's, overflow has occurred, in which case the output is set to the largest 16-bit positive number, 32767. If overflow has not occurred, then the 32-bit result of the multiplication is shifted right 1 bit and the lower 16-bits are taken as the result.

As with the FFT cores, the same modifications had to be made to the relevant data files in order for the block to use the Xilinx cores. Unfortunately, there was not enough time to fully simulate the block before implementation. Only a single case of non-overflow input was tested, which worked correctly. Therefore, it is not entirely unexpected that the equalizer core did not function correctly in hardware. Also, the second FSL interface to the block that would allow for updates to the gains was not implemented either. The block as it currently is, simply takes input samples and multiplies by 2 and then shifts right 1 bit, effectively applying a gain of 1 to all frequencies.

3.4 FFT Controller

The FFT controller is used to communicate with the FFT blocks via the MicroBlaze. The controller performs two functions: it signals the FFT to start processing and it reads back the blk_exp value of the FFT blocks. The functionality of the controller is summarized in Figure 6.

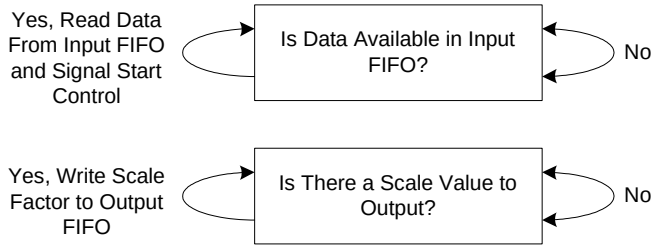

Figure 6 - Functionality of FFT controller.

Two controllers were used in our design: one for the forward FFT and another for the inverse FFT. The controller for the FFT block is used to begin processing on the current audio frame and to read back the scale amount for the forward FFT. The IFFT controller is only used to read back the scale factor used, since it is signaled to start by the Equalizer block.

The controller does not use any other IP, since it is a fairly simple block. Also, simple simulations were carried to make sure the block was responding correctly to external requests.

3.5 Echo

The echo block adds an echo effect to incoming audio data. The block uses Xilinx's Single Port Block Memory v6.1 [4] core as a buffer. The block implements an FSL Interface to read and write audio data. The buffer is used as a circular buffer. Reads begin from the start of the memory, while writes start at the end of the memory. After every write into the echo block, a read is performed while the read and write pointers are incremented. Since the size of block memory is a power of 2, the appropriate bit size can be chosen for the address registers such that they wrap around upon overflow. The block memory was generated using the parameters in Table 4.

Parameter	Value
Memory Size – Width	16
Memory Size – Depth	16384
Write Mode	No Read on Write
Port Design Options	None Selected
Output Register Options	None
Design Options – Pin Polarity	Rising Edge Triggered, Active High
Primitive Selection	Optimize For Area
Simulation Model Options	Disable Warning Messages
Initial Contents – Global Init Value	0

Table 4 – Parameters used to generate the single port block memory

3.6 MicroBlaze [5]

This block was taken from the library. We used version 3.00a with default parameters, except for the parameter C_FSL_LINKS, which was set to 4 in order to have 4 FSL links connected to the Microblaze. The reference clock frequency and bus frequency were both set at 27 MHz and 64 KB of local data and instruction memory were used. Also, XMD with software debug stub was used for debugging purposes. The MicroBlaze was used to buffer and window the incoming audio stream to send to the FFT core and to display and process the user interface. The functionality of the MicroBlaze is summarized in Figure 7. Testing the software code and hardware block functionality was done using the XMD stub and GDB.

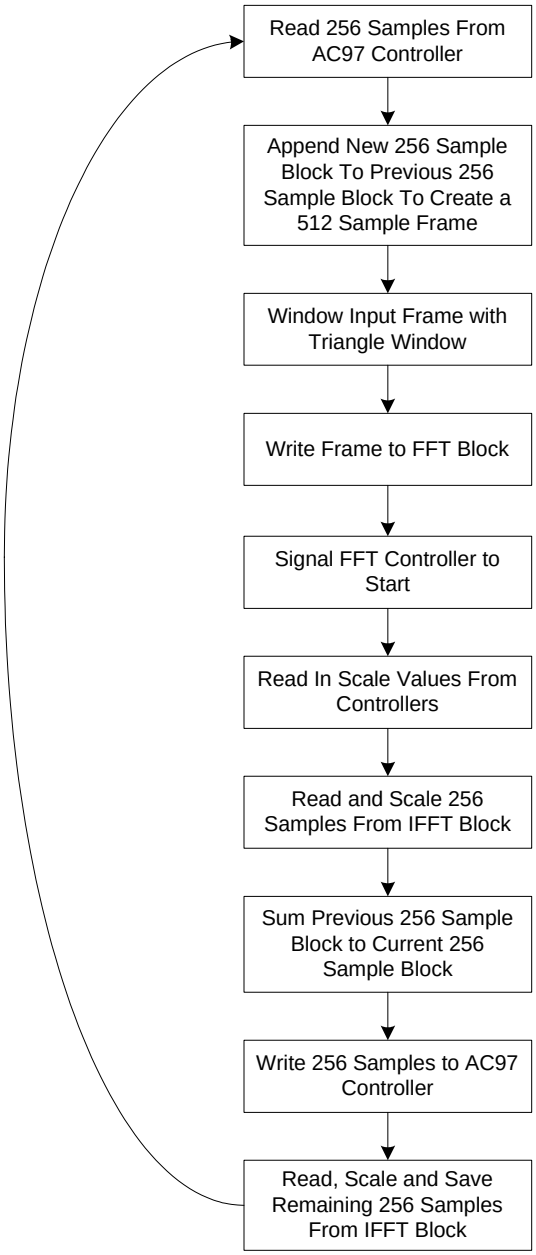

Figure 7 - Functionality of MicroBlaze.

4 Design Tree

Our design tree is organized as follows. Each hardware block has a directory "*name_of_block*_core". Each block has two subdirectories: one directory for the ISE project of the block that contains all design files and test benches and a second directory that is to be copied to the pcores directory of an XPS project directory. The XPS project directory is under the directory "project" and contains the final hardware setup. That is, the AC97 controller, FFT/IFFT blocks, FFT controllers, equalizer and the MicroBlaze. The software is setup to read 512 samples of audio from the AC97 controller and send the frame off to the FFT block. The FFT block passes the frame onto the equalizer block which then finally passes the frame to the IFFT block. The

audio data is then read back and sent to the AC97 controller to output. The following is a list of the hardware blocks included in our design tree:

- echo_block – the echo block
- eq_core – the equalizer block
- fft_core – the FFT block
- fft_ctrl_core – the FFT controller
- ifft_core – the IFFT block
- inbuf_core – block from initial design that was to be used to buffer input data
- led_core – simple block used for debugging purposes to control the LEDs
- m2s_core – block from initial design that converted the stereo audio stream to mono
- s2m_core – block from initial design that converted the processed mono audio stream to stereo

The testEchoBlock directory contains the XPS implementation and demonstrates the echo block. It also contains a preliminary user interface in testEchoBlock/code/testEchoBlock.c

The "doc" directory contains our project documentation, namely, this report, and our individual reports as well, in PDF format.

5 References

[1] Xilinx Intellectual Property: Fast Fourier Transform, "Fast Fourier Transform v3.1", November, 2004, http://www.xilinx.com/ipcenter/catalog/logicore/docs/xfft.pdf

[2] Xilinx Intellectual Property: Dual-Port Block Memory, "Dual-Port Block Memory v6.1", May, 2004, http://www.xilinx.com/ipcenter/catalog/logicore/docs/dp_block_mem.pdf

[3] Xilinx Intellectual Property: Multiplier Generator, "Multiplier Generator v6.0", Novemeber, 2002, http://www.xilinx.com/ipcenter/catalog/logicore/docs/mult_gen.pdf

[4] Xilinx Intellectual Property: Single-Port Block Memory, "Single-Port Block Memory v6.1", May, 2004, http://www.xilinx.com/ipcenter/catalog/logicore/docs/sp_block_mem.pdf

[5] Xilinx: MicroBlaze Resource Page, "MicroBlaze Processor Reference Guide", August, 2004, http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf