

# **ECE 492 Final Report:**

## **Voice Manipulator**

Device that transforms an input voice signal and sends it to an audio output device

Sean Hunter  
Michael Wong  
Thomas Zylstra

## **ABSTRACT:**

The Voice Manipulator takes input from a microphone and changes it based on input from the user before sending the modified audio to an output device. It is built on the Altera DE2 platform, with a Cyclone II FPGA. DSP functionality is implemented in hardware in the form of two modules: a Frequency Shifter and an Echo Generator. Frequency shifting is performed using single sideband modulation. The user uses the buttons on the DE2 to control the parameters for the transformations, with the LCD providing the display of the user interface. A switch is used to select between Direct Mode and Bluetooth Mode. In Direct Mode, audio from the microphone is transformed and played through speakers. In Bluetooth Mode, audio from the microphone is transformed and sent over Bluetooth, while incoming audio over Bluetooth is played through speakers. A Nios II processor implemented on the FPGA is responsible for controlling audio data flow and managing the user interface.

We were able to successfully implement audio input and output through the Audio Codec, as well as all intended manipulations: volume adjustment, frequency shift, and echo generation. We were also able to interface with the Bluetooth transceiver, allowing the Voice Manipulator to transmit audio data over a Bluetooth link. Potential avenues for future expansion are also presented in this report.

# **TABLE OF CONTENTS**

FUNCTIONAL REQUIREMENTS .....	3
DESIGN AND DESCRIPTION OF OPERATION .....	4
Overall Operation .....	4
User Interface .....	5
Communication with Bluetooth Transceiver .....	5
Frequency Shifting .....	6
Echo Generation .....	7
BILL OF MATERIALS .....	9
REUSABLE DESIGN UNITS .....	10
Audio Core for Altera DE-Series Boards .....	10
Hilbert Transformer .....	10
DATASHEET .....	11
User-Perspective Block Diagram .....	11
Performance .....	11
IO Signals .....	11
Power Supplies .....	14
Power Consumption .....	14
Operating Conditions .....	14
BACKGROUND READING .....	15
SOFTWARE DESIGN .....	16
RESULTS OF EXPERIMENTATION AND CHARACTERIZATION .....	18
TEST PLAN AND RESULTS .....	20
Integration .....	20
Software .....	20
Hardware .....	21
REFERENCES .....	24
APPENDIX A: QUICK START MANUAL .....	25
APPENDIX B: HARDWARE DOCUMENTATION .....	26
APPENDIX C: SOURCE CODE .....	27
APPENDIX D: FUTURE WORK .....	28

## **FUNCTIONAL REQUIREMENTS**

The core functionality of the Voice Manipulator is to apply audio transformations to an input voice signal, captured through a microphone, and send the modified audio to an output device, which will either be speakers or a Bluetooth transceiver. Whether the output is sent to Bluetooth or speakers is chosen by the user using one of the board's switches. In the case of Bluetooth, the microcontroller would be responsible for sending audio data to an attached Bluetooth transceiver, which would present itself as a typical Bluetooth microphone. This would allow the Voice Manipulator to be used with a cellphone, for example.

The transformations occur as audio streams, so that the manipulation begins as soon as any sound is captured, and the modified sound is played as soon as the transformations are complete. The basic transformations proposed are volume adjustment, frequency shifting, and echoing. The user will specify the degree of volume change, the degree of frequency shift, the time delay before the echo, and whether to reduce the volume of the echoed sound. Digital manipulation of audio will necessarily incur some delay between input and output, but effort is made to optimize performance by minimizing the time required by the Voice Manipulator to finish its computations, in order to reduce as much as possible the delay between capture and playback. If the device is to function without annoying the user, this delay should be less than 0.1s.

Every manipulation was successfully implemented, although downsampling from 32 kHz to 8 kHz was required for the Bluetooth module, since the Bluetooth transceiver chosen for this project can only process audio at 8kHz. However, this downsampling does not noticeably diminish audio quality. Bluetooth pairing allows the user to pair their phone with the Voice Manipulator and to use it to transform their voice during a phone call. Although the frequency shifting initially presented several issues due to its relatively complex mechanism, this functionality has been greatly improved since initial implementation. In the project's final state, a voice signal can be modified and played back noticeably transformed, but still satisfactorily understandable. The delay between input and output is imperceptible, and estimated to be on the order of one millisecond, well within the originally proposed constraint of 0.1s.

The original project proposal called for directing audio input to the Bluetooth transceiver to be sent to a phone, but it makes no mention of outputting the audio coming from the phone over speakers. We were able to implement this additional functionality, so that when the Voice Manipulator is directing audio to the Bluetooth, it is also playing the received audio, unmodified, over connected speakers. In this way, the Voice Manipulator functions as a complete Bluetooth headset with bidirectional audio.

However, we did not see success in all proposed additional features. We attempted to interface a keypad to enable a hands-free dialing mode, in which the user would be able to dial a number from the board without needing to handle the phone. We were able to implement this feature in an isolated project, but integration efforts resulted in a significant drop in audio quality in the core functionality of the Voice Manipulator. Unfortunately, due to time constraints, we were ultimately unable to solve this problem before the end of term.

# DESIGN AND DESCRIPTION OF OPERATION

## Overall Operation

The fundamental functionality of the Voice Manipulator is to capture a voice signal through the microphone, and then apply modifications to the input audio signal as specified by the user through the onboard buttons. The output is then either played through speakers, or sent through Bluetooth to be used by a paired device.

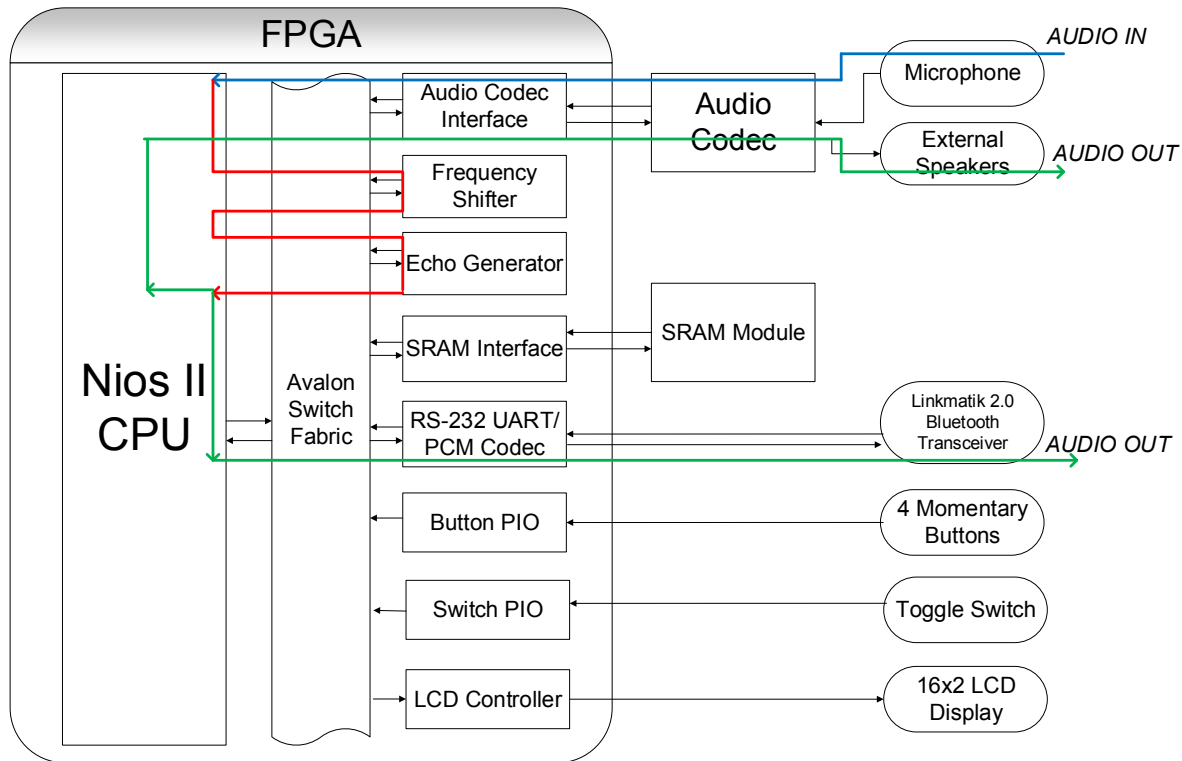


Figure 1 – Hardware Diagram Showing Audio Data Flow

Figure 1 shows a hardware block diagram for the Voice Manipulator, as well as the path through the system that the audio data stream takes: the blue path represents audio input, the red path represents signal processing, and the green path represents audio output. A Nios II processor is implemented on the FPGA, and is responsible for directing the audio data to the appropriate DSP module. The analog audio signal is captured through a microphone, and is fed into the audio codec interface where it is converted into a digital stream of 16-bit audio sampled at 32kHz. It is then sent through the switch fabric to the processor, where it is redirected into the Frequency Shifter. The output of the Frequency Shifter returns to the processor and is then passed to the Echo Generator. The final audio stream is passed to the appropriate output channel. See Appendix B for a hardware block diagram of the DSP modules.

## User Interface

The toggle switches present on the DE2 are used to allow the user to configure audio flow on the Voice Manipulator. One toggle switch is used to select between Direct Mode and Bluetooth Mode. In Direct Mode, the audio captured through the connected microphone is transformed and then routed to the external speakers. In Bluetooth mode, it is instead routed to the PCM Codec to be transmitted over Bluetooth, while the incoming Bluetooth audio (ie- the other direction of the call) is routed to the external speakers. It is in this mode that the Voice Manipulator functions as a Bluetooth headset.

The four momentary buttons are used to set the transformation parameters and output volume. The Voice Manipulator offers the ability to modify four properties of the outgoing audio stream: volume, frequency, echo delay, and echo decay. Two buttons are used to select a parameter to modify by cycling through these four options. The other two buttons are used to increment or decrement the value of the selected parameter. The currently selected parameter and its current value are displayed on the 16x2 LCD Display.

## Communication with Bluetooth Transceiver

Flexipanel's LinkMatik 2.0 Bluetooth Transceiver provides two interfaces that are useful for operation of the Voice Manipulator: a serial RS-232 interface, and a PCM audio interface.

The LinkMatik 2.0 is a feature-rich Bluetooth transceiver with many options for configuration. The RS-232 interface is therefore used to configure the transceiver's pairing behaviour and mode of operation. The LinkMatik implements Bluegiga's iWRAP Bluetooth stack, which is proprietary software that abstracts away many of the complexities of data transfer over Bluetooth. The presence of iWRAP makes configuration simple. By sending ASCII commands over the RS-232 interface, the Voice Manipulator can set the Bluetooth device name, authentication PIN code, and device class, among other settings. It can also enable to needed services for headset operation over this interface. Furthermore, the LinkMatik uses this serial interface to communicate status messages from the connected phone (call incoming, call hangup, etc.) to the system, although this functionality is not used in standard operation of the Voice Manipulator.

Audio data is communicated using a separate set of pins. The LinkMatik transceiver uses four pins to transfer audio data with the appropriate timing:

- PCMC – bit clock
- PCMS – synchronization signal
- PCMI – audio from Voice Manipulator to transceiver
- PCMO – audio from transceiver to Voice Manipulator

PCMC is the bit clock signal that provides the timing needed to transfer the serial data. Individual bits are read on the falling edges of the bit clock. PCMS is a signal used to indicate the start of an audio sample. When PCMS goes high, audio data is expected on both input and output lines by the following falling edge of the bit clock. The LinkMatik transceiver processes 16-bit audio at a rate of 8kHz. This imposes the requirement that the input audio be downsampled from 32kHz to 8kHz if being sent over the Bluetooth.

It was determined via oscilloscope probing that when a call is ongoing, the LinkMatik transceiver generates a 256kHz clock signal as the bit clock. It also generates a synchronization signal PCMS that

risers with a rising edge of the bit clock, stays high for 8 cycles, and then goes low for 24 cycles. All in all, there are 32 bit clock cycles between rising edges of PCMS.

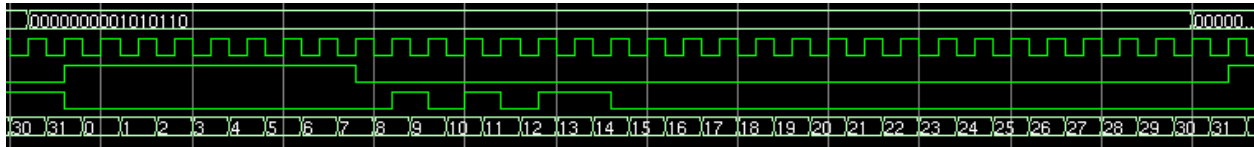


Figure 2 – Sending Audio Over PCM – Parallel to Serial

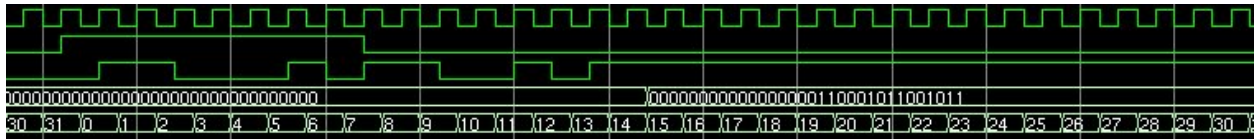


Figure 3 – Reading Audio Over PCM – Serial to Parallel

The design challenge presented is therefore to implement a PCM codec that, after the rising edge of PCMS, performs the following:

- takes 16-bit integers from a FIFO fed by the processor and transmits them serially over PCMI
- reads 16 bits serially from PCMO to obtain an integer value that can be passed to the processor

These requirements are illustrated as timing diagrams in Figures 2 and 3. In Figure 2, the top line is the 16-bit value to be transmitted. This is followed by the bit clock and synchronization signals, and then by the desired PCMI waveform. In Figure 3, the bit clock and synchronization signals are shown at the top, followed by the incoming serial data over PCMO. The codec must read this serial data and output a 16-bit integer, shown on the fourth line, when transmission is complete. In both figures, a counter is included at the bottom that illustrates the 32 cycles between rising edges of PCMS.

Due to the downsampling requirement (32kHz to 8kHz), the FIFO that provides the values to be serialized by this codec must be designed so that it accepts data even if it is full, and simply discards the value at the front of the queue if there is no room for the incoming sample. This allows the PCM codec to simply read the front of the queue for a value according to the 8kHz sync to achieve the downsampling.

### Frequency Shifting

The Frequency Shifter implements single sideband modulation in order to shift a given voice signal. The Voice Manipulator will implement the phase shifting method, which uses the Hilbert transform, an operation that shifts the phase of all positive frequencies by  $-90^\circ$ , and of all negative frequencies by  $+90^\circ$ . It can be shown that for an input signal  $x(t)$ , the following expression[1]:

$$x(t) \cos(2\pi f_c t) + H\{x(t)\} \sin(2\pi f_c t)$$

...where  $H\{x(t)\}$  is the Hilbert transform of  $x(t)$ , is a real-valued signal where all frequency components are shifted away from baseband by a frequency  $f_c$ . The Hilbert transform can be approximated using a finite impulse response filter. Figure 4 shows DSP operation of the Frequency Shifter. The presence of a

delay block is due to the delay introduced by the FIR filter, and ensures that the two branches of the datapath remain in phase.

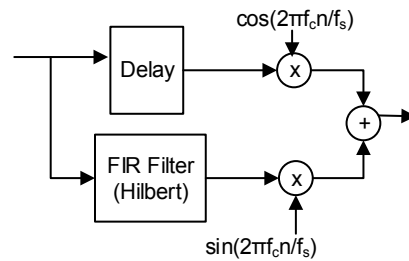


Figure 4 – Operation of the Frequency Shifter

Four levels of frequency shifting, including the zero-shift level, are possible on the Voice Manipulator:

- Upwards shift of 1kHz, upper sideband
- Upwards shift of 600Hz, upper sideband
- Zero shift
- Upwards shift of 200Hz, lower sideband

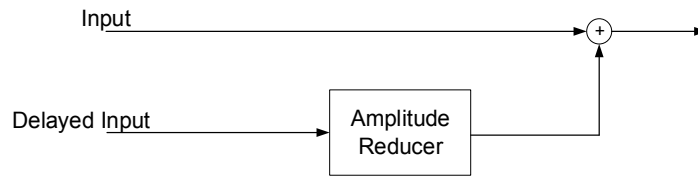
These values are chosen qualitatively and experimentally, with the aim of obtaining shifts that are noticeably different but still understandable. The upwards shift of 1kHz is an extreme shift, intended to push to border of intelligibility, while the shift of 600Hz results in a voice signal that is noticeably shifted, but still understandable. When zero shift is selected, the input audio should be unmodified by the Frequency Shifter. Finally, note that single sideband modulation is only able to shift an audio spectrum linearly *away* from baseband, which means that it is impossible to achieve a downwards shift using this technique. However, it was experimentally determined that by selecting the lower sideband instead of the upper sideband with a relatively small shift of 200Hz, the output tends to sound lower for the frequencies present in a typical human voice sample. Using the lower sideband instead of the upper sideband can be done with a similar expression, but by subtracting the two terms instead of adding them.

### Echo Generation

When the audio data stream returns from the Frequency Shifter, the Nios II processor stores a history of 0.5s in a circular buffer (16000 samples for 32kHz audio). Two audio data streams are then passed to the Echo Generator module: the current audio stream, and the delayed audio stream.

With a history stored in this manner, the Voice Manipulator is able to control the delay of the echo by changing the position of the pointer used to read from this history buffer, thus affecting the amount of time between the current and delayed streams. Since this is done in software, the Echo Generator is left with the simple task of adding the signals together, and with making the delayed signal by reducing the signal amplitude if Echo Decay is turned on by the user. This amplitude reduction is implemented as a bit-shift to the right by two places. Operation of the Echo Generator is shown in Figure 5.





*Figure 5 – Operation of the Echo Generator*

The effect of the history buffer and amplitude reduction combined ultimately results in implementation of the following expression:

$$y[n] = x[n] + (1 - 0.75R)x[n - 32000d]$$

...where  $R \in \{0,1\}$  is a boolean value representing whether Echo Decay is turned on (with 0 denoting “off” and 1 denoting “on”), and  $d \in \{0,0.1,0.2,0.3,0.4,0.5\}$  is the amount of time in seconds between the current audio signal and the echo. Note that specifying no echo is done by setting delay  $d$  to 0.

As the Echo Generator is the final DSP module before output, it is also responsible for implementing the base value shift necessary for proper output (see Results of Experimentation and Characterization). Thus, each sample is summed with 0x7fff before being passed back to the processor.

## **BILL OF MATERIALS**

<b>Part</b>	<b>Description</b>	<b>Unit Cost</b>	<b>Quantity</b>	<b>Total Cost</b>
Altera/Terasic DE2* ( <a href="http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html">http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html</a> )	FPGA development board -24-bit Audio Codec -16x2 LCD display -512kB SRAM -4MB Flash memory -4 push button keys -18 toggle switches	\$495.00	1	\$495.00
FlexiPanel LM20 ( <a href="http://www.flexipanel.com/Docs/LinkMatik%202.0%20DS379.pdf">http://www.flexipanel.com/Docs/LinkMatik%202.0%20DS379.pdf</a> ) [supplier: <a href="http://www.digikey.ca">http://www.digikey.ca</a> ]	Class 1 Bluetooth Transceiver -Serial data interface -PCM audio interface	\$89.85	1	\$89.85
Logitech G330 [supplier: <a href="http://www.bestbuy.ca">http://www.bestbuy.ca</a> ]	Headset microphone/ speakers	\$49.99	1	\$49.99
Through-hole Vector Board*		\$8.00	1	\$8.00
Vector Board feet*		\$0.05	4	\$0.20
Vector Board screws*		\$0.05	4	\$0.20
40-pin Ribbon Cable*		\$5.00	1	\$5.00
40-pin Header*		\$1.00	1	\$1.00
30 AWG wire wrap wire*		\$0.25/ft	2 ft	\$0.50
			<b>Total</b>	\$658.84

\*Indicates component supplied by lab instructor

## **REUSABLE DESIGN UNITS**

### **Audio Core for Altera DE-Series Boards**

The Altera Audio Core provides an interface for the WM8731 Audio CODEC chip on the DE2 board which can be instantiated inside the Quartus 2 SOPC Builder. Altera also provides the Audio/Video Configuration Core for DE-Series Boards, which can be used to configure the audio CODEC settings, as well as the Clock Signals for Altera DE-Series Boards cores which provide the necessary input clock signal for the Audio CODEC. These cores will be highly useful in implementing the Voice Manipulator, as they enable a simple interface to the audio codec via the Altera Hardware Abstraction Layer (HAL).

- [[http://www.altera.com/education/univ/materials/embedded\\_systems/ip-cores/unv-ip-cores.html](http://www.altera.com/education/univ/materials/embedded_systems/ip-cores/unv-ip-cores.html)]
- Source Size: 507kB
- Source Size including HAL C files: 715kB
- Resource Requirements:
  - 235 Logic Elements on Cyclone II FPGA
  - Clock Signals Core requires one PLL
- Performance:
  - These cores performed well, which is expected since they were developed by the actual supplier of the platform. No major issues were encountered when working with these cores.

### **Hilbert Transformer**

This is a core that implements the Hilbert Transform, which is needed in the Voice Manipulator for the frequency shifting operation. Development is complete, and the core implements a fully pipelined design to maximize throughput. This core actually provides several implementations of an analytic filter, another DSP operation of which the Hilbert transform is a part. It therefore also offers a Hilbert transformer as a submodule. We reused this Hilbert transformer design in the Voice Manipulator, and expanded it to create the Frequency Shifter.

- [[http://opencores.org/project,hilbert\\_transformer](http://opencores.org/project,hilbert_transformer)]
- Source Size: 5.4kB
- Resource Requirements:
  - 75 Logic Elements on Cyclone II FPGA
- Performance:
  - Due to the fully pipelined design, we expect that this module would perform very well. The Frequency Shifter based around this core had no trouble handling the data throughput needed for successful operation of the Voice Manipulator

# DATASHEET

## User-Perspective Block Diagram

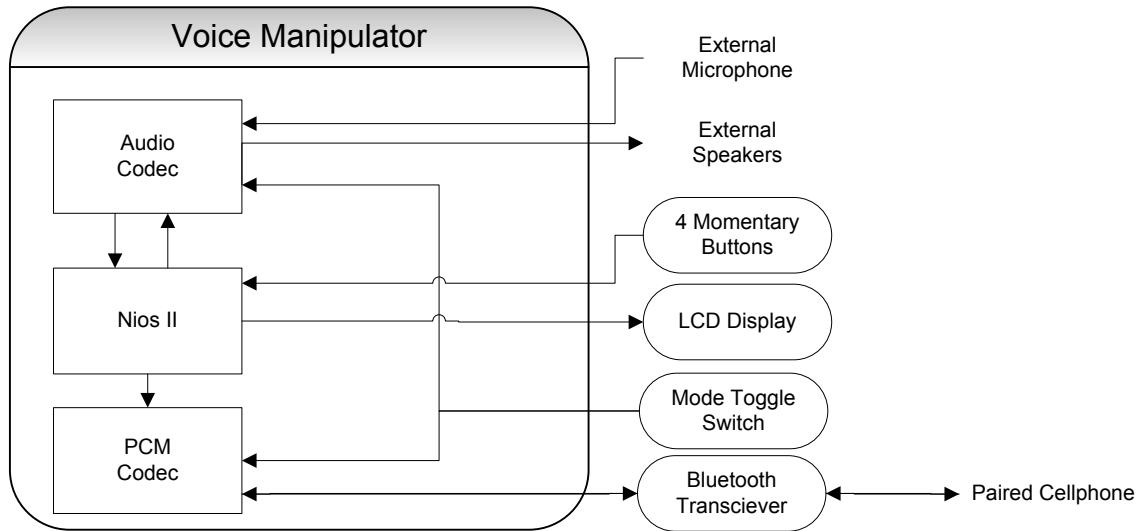


Figure 6 – User-perspective Block Diagram

## Performance

- Input Audio: 16-bit, 32kHz sampling rate; 3.5mm external MIC IN
- Output Audio: 16-bit, 32kHz sampling rate; 3.5mm external LINE OUT
- Output Audio: 16-bit, 8kHz sampling rate; PCM over Bluetooth link
- Echo delay of 0.1s to 0.5s in 0.1s intervals, at full amplitude or  $\frac{1}{4}$  amplitude
- Single sideband modulation frequency shifts of 200Hz LSB, 600Hz USB, 1kHz USB

## IO Signals

IO Signal	Located (FPGA Pin if applicable)	Description
<b>RAM Signals</b>		
RAM_DATA	FPGA to board (SRAM_D)	16-bit bus. Carries data between processor and off-chip RAM module.
RAM_ADDR	FPGA to board (SRAM_A)	18-bit bus. Communicates address between processor and off-chip RAM module.

RAM_UPPER_STROBE	FPGA to board (SRAM_UB)	Communicates upper strobe between processor and off-chip RAM module.
RAM_LOWER_STROBE	FPGA to board (SRAM_LB)	Communicates lower strobe between processor and off-chip RAM module.
RAM_WRITE_EN	FPGA to board (SRAM_WE)	Used to carry the signal that enables writes in the off-chip RAM module.
RAM_CHIP_EN	FPGA to board (SRAM_CE)	Used to carry the signal that selects the chip in the off-chip RAM module.
RAM_OUTPUT_EN	FPGA to board (SRAM_OE)	Used to carry the signal that enables output in the off-chip RAM module.
<b>Audio Codec Signals</b>		
AUDIO_IN	Off-board	Input analog audio signal. Captured via microphone and passed into the board via the 3.5mm MIC IN. $1.0V_{rms}$ .
AUDIO_OUT	Off-board	Output analog audio signal. Passed to output speakers via the 3.5mm LINE OUT. $1.0V_{rms}$ .
AUD_DACLCK	FPGA to board (AUD_DACLCK)	Alignment clock which determines whether left or right channel data is present on ADCDAT.
AUD_ADCDAT	FPGA to board (AUD_ADCDAT)	Formatted digital audio stream input from the ADC digital filters with left and right channels multiplexed together.
AUD_ADCLK	FPGA to board (AUD_ADCLK)	ADC sample rate left/right clock. 12.288 MHz.
AUD_DACDAT	FPGA to board (AUD_DACDAT)	Formatted digital audio stream output to the DAC digital filters with left and right channels multiplexed together.
AUD_BCLK	FPGA to board (AUD_BCLK)	Alignment clock which determines whether left or right channel data is present on DACDAT.
<b>Board Peripheral Signals</b>		
BUTTON_0	FPGA to board (KEY0)	Button 0. Used to increment current DSP parameter.
BUTTON_1	FPGA to board (KEY1)	Button 1. Used to decrement current DSP parameter.
BUTTON_2	FPGA to board (KEY2)	Button 2. Used to select next DSP parameter.
BUTTON_3	FPGA to board (KEY3)	Button 3. Used to select previous DSP parameter.

SWITCH_0	FPGA to board (SW0)	Switch 0. Used to select output device: Speakers through Line Out or Bluetooth link to paired device.
<b>LCD Peripheral Signals</b>		
LCD_ENABLE	FPGA to board (LCD_EN)	Enable Signal. Signal to LCD controller designating that a read or write should occur.
LCD_RS	FPGA to board (LCD_RS)	Register Select. Signal to LCD controller designating which register is to be written to or read from.
LCD_R/W	FPGA to board (LCD_WR)	Read/Write Select. Signals to LCD controller whether selected register is to be written to or read from.
LCD_DB	FPGA to board (LCD_D)	8-bit bus. Carries data between LCD controller and processor.
<b>Bluetooth Module</b>		
BT_CTS	FPGA to Off-board (GPIO_1[9])	Bluetooth Serial Link - Clear to Send: When low, LM20 outputs data to TxD.
BT_PCMC	FPGA to Off-board (GPIO_1[4])	Bluetooth Pulse Code Modulation - audio bit clock generated by LM20. 256kHz.
BT_PCMI	FPGA to Off-board (GPIO_1[8])	Bluetooth Pulse Code Modulation - serial audio input signal. Sent over Bluetooth link to paired device.
BT_PCMO	FPGA to Off-board (GPIO_1[6])	Bluetooth Pulse Code Modulation - serial audio output signal. Received over Bluetooth link from paired device.
BT_PCMS	FPGA to Off-board (GPIO_1[2])	Bluetooth Pulse Code Modulation - audio frame sync generated by LM20. 8kHz.
BT_RTS	FPGA to Off-board (GPIO_1[7])	Bluetooth Serial Link - Ready to Send: When high, LM20 is not ready to receive data.
BT_RxD	FPGA to Off-board (GPIO_1[5])	Bluetooth Serial Link - data input from FPGA to LM20
BT_TxD	FPGA to Off-board (GPIO_1[3])	Bluetooth Serial Link - data output from LM20 to FPGA

## Power Supplies

Component	Typical Voltage	Min, Max Operating Voltage
LCD - Logic Supply Voltage	5.0 V	4.5 V, 5.5 V
LCD - LCD Drive Voltage	4.7 V	4.2 V, 5.3 V
BT Module - Supply Voltage	5.0 V	3.2 V, 5V
SRAM - Supply Voltage	3.3 V	3.0 V, 3.6 V
CODEC - Core Voltage	1.5 V	1.42 V, 3.6 V
CODEC - Buffer Voltage	3.3 V	2.7 V, 3.6 V

## Power Consumption

The LinkMatik 2.0 Datasheet gives the following values for average current draw of the Bluetooth transceiver:

- Idle – 3mA
- Connected as master – 21.5mA
- Receiving – 33.4mA

Allowing 5mA of draw for the DE2 itself, we can calculate power consumption as (9V)(total current).

Condition	Measured Voltage	Measured Current	Measured Power	Calculated Power
Powerup	9.03V	24mA	217mW	-
Idle	9.03V	8mA	72mW	72mW
Pairing Bluetooth	9.03V	24mA	217mW	-
Connected Bluetooth	9.03V	17mA	154mW	239mW
During Call	9.03V	31mA	280mW	346mW

Note that the measured power during operation seems to be less than the values obtained from calculation.

## Operating Conditions

- Storage Temperature: -40°C to 105°C
- Operating Temperature: -40°C to 85°C
- Operating Voltage: 9V (standard DE2 power supply)

## **BACKGROUND READING**

1. A. Shahina and B. Yegnanarayana. (2006, Oct 4). “Mapping Speech Spectra from Throat Microphone to Close-Speaking Microphone: A Neural Network Approach” in EURASIP Journal on Advances in Signal Processing. doi: 10.1155/2007/87219
2. MathWorks. *Single Sideband Modulation via the Hilbert Transform*. [Online]. Available: <http://www.mathworks.com/products/signal/examples.html?file=/products/demos/shipping/signal/hilberttransformdemo.html>
3. W. Verhelst, M. Roelands, “An Overlap-add Technique Based on Waveform Similarity (WSOLA) for High Quality Time-scale Modification of Speech” in IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 554 - 557 vol. 2, 1993.
4. The DSP Dimension. (1999 Sept 21). *Pitch Shifting Using The Fourier Transform*. [Online]. Available: <http://www.dspdimension.com/admin/pitch-shifting-using-the-ft/>

The first paper was found while researching properties of a voice signal captured by a throat microphone. The Voice Manipulator was originally proposed as a Throat Microphone Enhancer. However, upon discovering that a throat microphone did not capture enough information to easily reconstruct a conventional voice signal, this idea was abandoned. This paper outlines one method of performing the conversion, which involves using a database of speech patterns to perform a mapping. Such an approach was deemed to be impossible for the timeframe allotted in this course. Nonetheless, it is an interesting read, and offers insight into acoustic linguistics, which is still relevant to the Voice Manipulator.

The second is a web resource that details how the Hilbert Transform is used to achieve single sideband modulation. It provides more details into the implementation and mathematics behind the Hilbert Transform, including waveforms and Matlab examples.

The latter two on this list provide a detailed look at alternative techniques to pitch shifting, which use more complicated processing involving the short-time Fourier Transform and summation of overlapping windows. These algorithms offer an advantage to single sideband modulation in that they implement a “true pitch shift” that preserves harmonics, which would be important if working with music rather than voice. As we are dealing only with a voice signal, we chose to use single sideband modulation for ease of implementation, but the more complex algorithms are likely implementable on an FPGA, offering a potential avenue of expansion for this project.



## SOFTWARE DESIGN

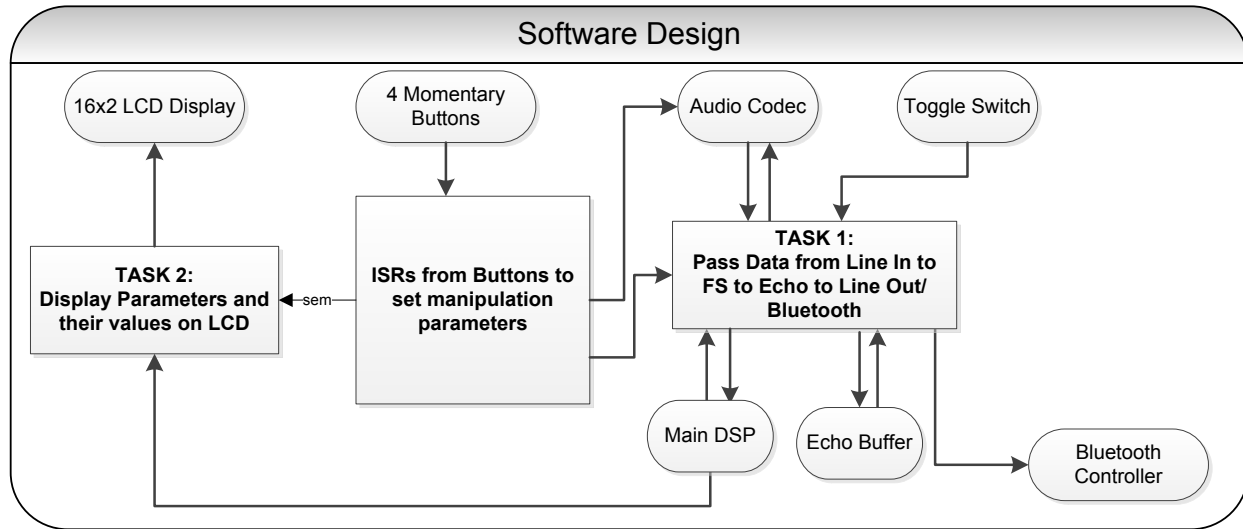


Figure 7 – Software Design of Voice Manipulator

The software for the Voice Manipulator will employ the MicroC RTOS, and will be comprised of two tasks and four interrupts to handle control of the manipulation parameters, as shown in Figure 7.

The Nios II will process interrupt routines triggered by the momentary buttons, and, depending on which button was pressed, modify the manipulation parameters. Two of the interrupts (corresponding to the first two buttons) will cycle through the different parameters to be modified. The other two interrupts (corresponding to the last two buttons) will be responsible for incrementing or decrementing the value of the parameter. In the case of volume adjustment, the value will be written straight to a register in the WM8731 to effect the change the volume, instead of modifying software flow. Each of the four interrupts will also post to a semaphore on which the LCD task pend. When a change is made, posting to this semaphore indicates to the LCD task that a change has been made, so that the LCD task can refresh the screen with the appropriate content.

In order to achieve the necessary communication of these parameters between interrupt service routines and tasks, an array of integers is declared in the main function whose base pointer is passed to the interrupt service routines and tasks as a context pointer. This array holds values indicating the parameter currently being modified, the volume, the echo delay, the presence of echo decay, and the degree of frequency shift. The potential for synchronization issues has been acknowledged in choosing this unprotected method of communication. Care was taken to ensure that only the interrupt service routines modify these values. As interrupts are triggered by buttons, there is little risk of one routine interrupting another. Furthermore, the tasks will read these values, but never write to them. Also, due to the nature of operation of this project, it is not critical that the audio task read the correct value as soon as it is written. The correct value will simply be read when processing the next audio sample, which still provides the desired behaviour.

The first task is the most important function of the software aspect of the Voice Manipulator. It is responsible for moving data between the Audio Codec interface or PCM codec and the DSP modules. It continually reads in audio samples from the FIFOs of the Audio Core Interface using the appropriate HAL API calls. The read data is then moved to the Frequency Shifter's input FIFO. However, as described in Design and Description of Operation, the Frequency Shifter also requires sine and cosine input signals, whose frequency corresponds to the degree of frequency shift. This is accomplished by having an array in software that holds 320 integers that, when read in order, produce one period of a sine wave. Then, by continually cycling through this array and sending one value to the Frequency Shifter each time we send an audio sample, we can effectively send a  $(32000 \text{ samples/s}) / (320 \text{ samples}) = 100\text{Hz}$  sine wave. The cosine wave is produced by keeping the cosine pointer 80 samples ahead of the sine pointer, as this corresponds to a  $90^\circ$  phase shift. Then, to emulate sinusoids of higher frequencies, this array is traversed in step sizes of greater than one sample. For instance, taking every third sample from this array will produce sinusoids with frequency 300Hz. The frequency shifter takes these three input streams (the audio, the sine wave, and the cosine wave) and outputs a frequency-shifted modified stream.

This task then takes this preliminary output from the Frequency Shifter's output FIFO and routes it to the Echo Generator, while also storing 0.5 seconds of sample history, or 16000 samples at 32kHz. This is done in a circular buffer. As audio data is received from the output of the Frequency Shifter, the pointer to the location to be written to is constantly updated so that 0.5 seconds of the most recent audio data is always present in the buffer. Then, by reading from the buffer using a read pointer that is constantly updated to be some number of samples behind the most current, we produce an audio stream that is a delayed copy of the current stream. Furthermore, the amount of delay can be increased by increasing the difference in index position between the two pointers. The difference will be 16000 samples for a delay of 0.5 seconds, 800 samples for a delay of 0.1 seconds, etc. Both the current audio stream and the delayed stream are passed to the Echo Generator, which takes these two inputs and returns the final output.

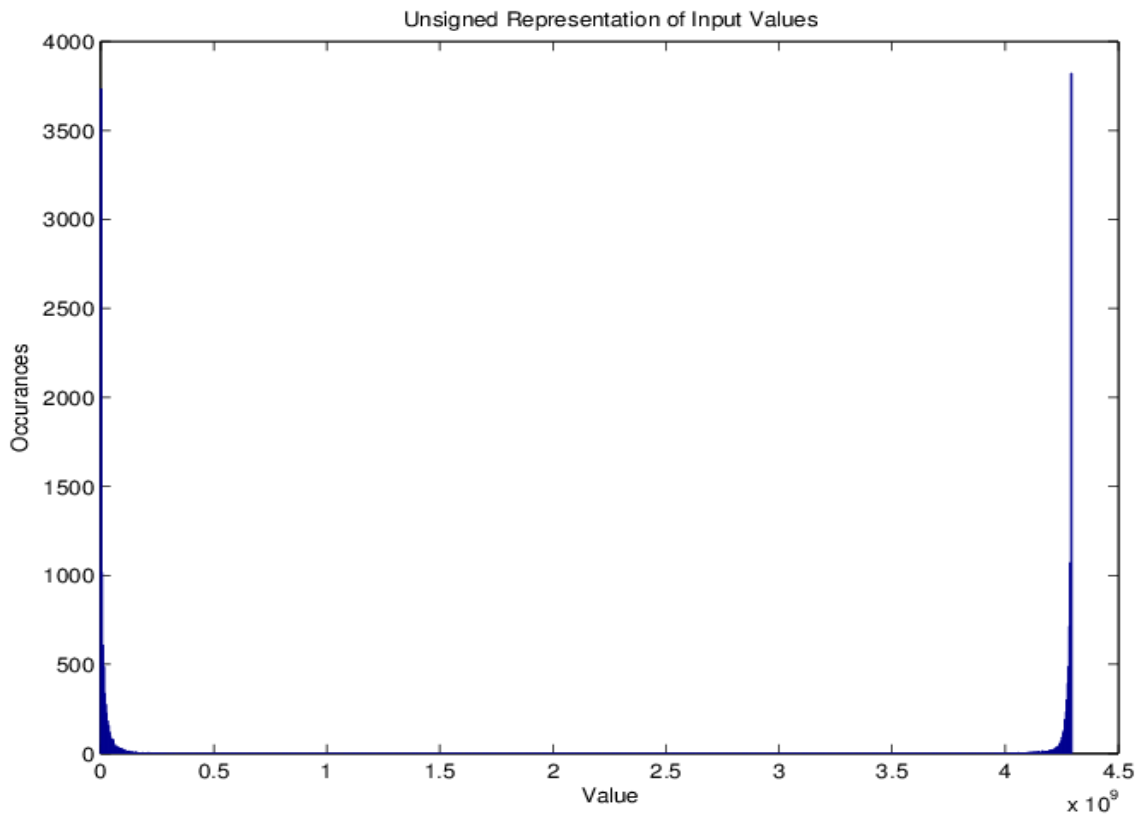
The audio data task then takes this output from the output FIFO of the Echo Generator and moves the data to either the Audio Codec outgoing FIFO or the PCM codec, determined by checking the position of a toggle switch on the DE2. If the audio stream is destined for the Audio Codec, the processed data is simply written to the Audio Codec Interface using the appropriate HAL API call. If the audio stream is destined for the Bluetooth link, it is written to the FIFO feeding the PCM codec to be serialized. Recall that the LinkMatik is only capable of handling audio data at a sampling rate of 8kHz. Although audio data is moving through the system with a sampling rate of 32kHz, the software component of the Voice Manipulator is not concerned with downsampling. The FIFO implemented on the PCM codec does not block incoming data if it is full, but rather accepts the sample and discards the sample at the front of the queue. Thus, by pushing data into the FIFO at 32kHz, but allowing the PCM codec to sample at 8kHz, we achieve the needed downsampling without any additional checks or processing.

The second task is responsible for LCD display. It displays the parameter currently being modified on the LCD display, as well as its current value, which is stored in software context. This task will be implemented in a loop that displayed the parameter being modified and its value, and then pends on a semaphore which is posted when a button interrupt fires. Because the LCD screen only needs to be updated when a button is pressed, this ensures that we do not waste processor resources on refreshing the LCD screen when it is not necessary.

## RESULTS OF EXPERIMENTATION AND CHARACTERIZATION

### Characterization of Audio Data

When attempting to implement a simple application that routes the audio from MIC IN to LINEOUT, it was observed that faithfully copying the digital values resulted in loud, high-frequency static, even when the microphone was only capturing ambient noise. The analog-to-digital conversion performed by the WM8731 appears to produce digital values above and below zero. This causes an issue when writing these values to a digital-to-analog converter, as the small negatives values result in overflow to a large positive value. Shown below in Figure 8 is a histogram representation (using 32-bit values rather than 16-bit) of approximately half a second of a typical voice signal.



*Figure 8 – Histogram Showing Digital Samples as Unsigned Integers*

From the sampling, we also extracted the following information (again for 32-bit values):

- Number of values: 16128
- Maximum positive value: 314041344 = 0x12B7E400
- Unsigned representation of minimum negative value: 3794451200 = 0xE22ABB00

Note that there appears to be a mirroring between the positive and negative values. We therefore hypothesize that the codec is using 0 as the baseline value, with the waveform swinging above and below zero as sound is captured. We can mitigate this overflow by shifting the baseline value to the median unsigned value, 0x7FFFFFFF for 32-bit values, or 0x7FFF for 16-bit. Indeed, implementing this constant shift results in understandable output.

### FIR Filter Design

The number of taps to use for our filters was another decision that had to be made. This decision is characterized by a trade-off between lowest frequency captured and delay. If we let the number of taps be  $n$ , then the delay of a FIR filter will be

$$delay = \frac{(n - 1)}{2f} = \frac{(n - 1)}{16000} \text{ seconds}$$

... for an output audio rate of 8kHz. However, with  $n$  taps, the lowest frequency we can capture for an audio rate of 8kHz is

$$f_{min} = \frac{8000}{n}$$

We therefore have a trade-off, where increasing  $n$  allows us to capture lower frequencies with our filter, while decreasing  $n$  reduces the delay. We chose  $n=102$  for this project. This allows us to capture frequencies down to 78.4Hz, which is lower than the fundamental frequency of a typical male voice[2], while giving a delay of 6.3ms, which is acceptable for this project.

### Numerical Simulation of Frequency Shifting Algorithm

Before implementation in hardware, the frequency shifting algorithm was first simulated using Matlab in order to verify proper functionality. A Simulink model was developed, shown in Figure 9, and the simulation was run with a test input .wav file. The resulting .wav file was then qualitatively examined, and it was verified that a frequency shift had indeed been performed.

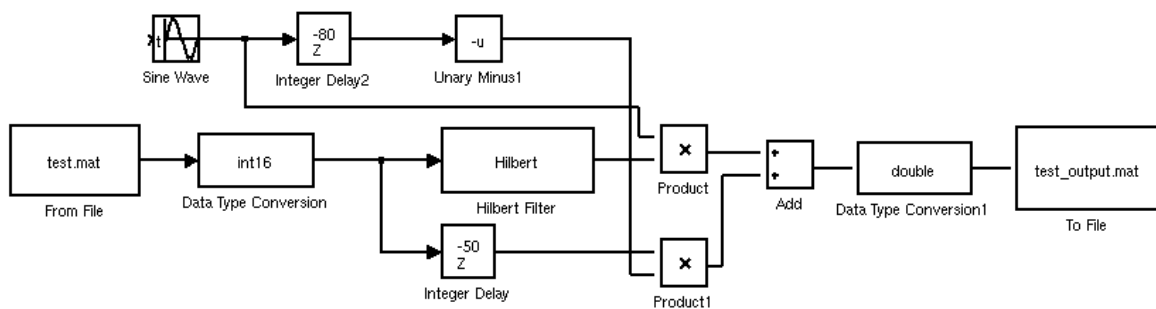


Figure 9 – Simulink model of Frequency Shifting Algorithm

## **TEST PLAN AND RESULTS**

### **Integration**

The primary metric for our system is the time delay from input to output. The delay incurred by routing the audio through the Voice Manipulator was closely monitored during development, and system design is optimized in order to obtain minimum delay. In order to avoid annoying the user, delay should be no greater than 0.1s.

In the final implementation we found the total delay in hardware to be less than 2ms, with some additional delay due to our PCM codec and Bluetooth transmission latency. It is not feasible to obtain an accurate quantitative measure of the total delay from input to output, but from a qualitative analysis, the delay is imperceptible, and certainly falls within our 0.1s ideal delay metric.

The integrity of the output signal was also qualitatively evaluated. The quality of the unaltered sound over speakers is clear and free of easily recognizable noise. With the echo effect applied, this signal is also clear and easy to understand. The frequency shift distorts the voice signal and introduces some undesirable noise, which makes it increasingly difficult to understand at higher levels of frequency shift. When the audio signal is sent over Bluetooth, there is a noticeable degradation in sound quality, but it is still easy to understand without effects applied. When a frequency shift greater than 1kHz is applied to the voice signal over Bluetooth it becomes very difficult to understand unless the speaker is very careful to speak slowly and clearly.

### **Software**

Button Interrupt Functionality: A basic routine was written to print out the status to the LCD for each button interrupt that was triggered, to ensure that the button interrupts were being dispatched and forwarded to an interrupt handler properly. This functionality was then integrated into our final software to control the DSP parameters.

I/O Functionality: Originally, a test routine was devised to ensure that the software could read to parameter registers in the DSP modules. As the DSP parameters were moved into arrays in software, testing of this functionality has changed. Updating of parameters was rigorously tested, including bounds checking to ensure that each parameter could not be incremented or decremented past its intended range.

Reliability Testing: The system was allowed to sit idle for 30 minutes continuously and the functional tests run, followed by a qualitative sound check, listening for overall responsiveness of the system and sound quality. We found no noticeable degradation in sound quality or change in overall system responsive time as a result of this idle time. We did not feel it necessary to check for memory leaks, as our software does not dynamically allocate any memory, and works instead with fixed size static buffers.

## Hardware

Audio Codec LINE OUT - Basic Functionality: Using the Altera audio test program as a sample, a basic routine to output a single tone to the audio codec was written. Speakers were connected to the 3.5mm audio output and the presence/absence of an output tone was observed to determine basic functionality of the audio codec interface. Once this test was successfully passed, this functionality was integrated into our final hardware design.

Audio Codec MIC IN- Basic Functionality: Once the functionality of the LINE OUT was verified it was used to determine the functionality of the MIC IN. The Altera audio test program was modified into a basic routine which routes the signal from the MIC IN through to LINE OUT. A microphone was connected to the 3.5mm line in, and speakers to the 3.5mm line out. An audio signal was captured by the microphone and compared against the sound output by the speakers to determine the audio codec line in/out functionality. Once this test was successfully passed, this functionality was integrated into our final hardware design.

Echo Generator - Basic Functionality: The echo generator implementation is fairly trivial, so extensive design simulation and characterization were deemed unnecessary. Testing of this functionality was done qualitatively; the implemented echo generator was tested by integrating it with the audio system and comparing the echoed sound with the original sample - and measuring the time between original signal and echoed signal.

Frequency Shifter - Basic Functionality: The VHDL implemented frequency shifter output was simulated for 500 samples of a typical voice signal. The single sideband modulation process was also simulated in Matlab's Simulink and applied to the same dataset. A comparison of the waveforms is shown in Figure 10. Note the close resemblance, indicating proper operation of the Frequency Shifter.

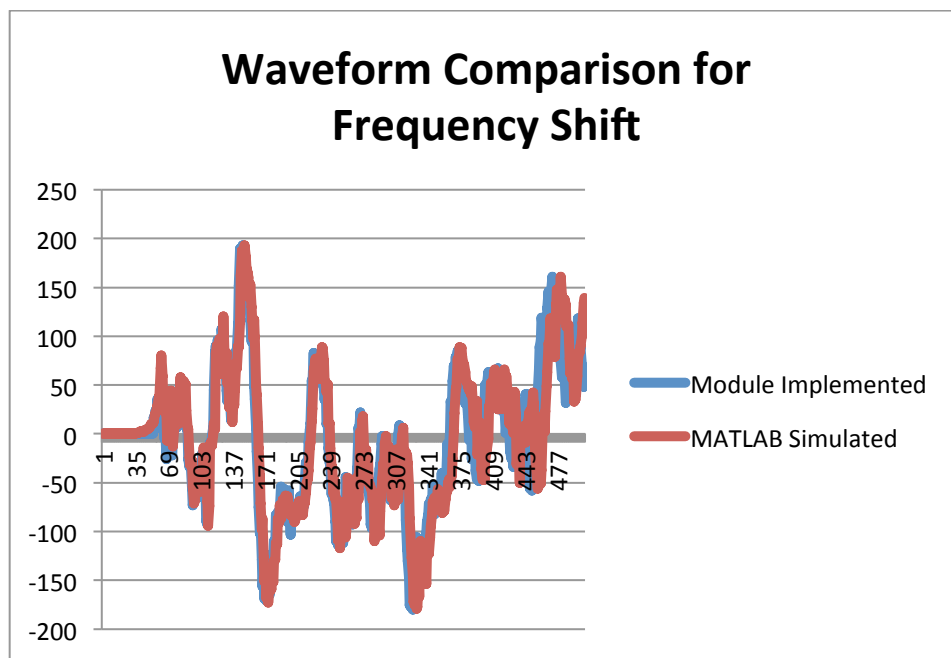


Figure 10 – Implemented and Simulated Waveforms for Frequency Shift

Bluetooth Transceiver - Basic Functionality: The first criterion for successful Bluetooth operation was to pair the Bluetooth module to another device - preferably a smartphone. After successful pairing confirmed basic functionality for the Bluetooth module, the second test was to send data via the serial interface to the paired device, compare the received data against the sent data, and send back data in the opposite direction to ensure proper transmission of data between the two devices.

```
FILE* lm20;
FILE* lcd;
char input = '\0';

lm20 = fopen(LM20_UART_NAME, "w+");
lcd = fopen("/dev/lcd_0", "w");

/* Configure Bluetooth module in serial mode */
fprintf(lm20, "SET CONTROL ECHO 2\n");
fprintf(lm20, "SET BT NAME ECE492_SM\n");
fprintf(lm20, "SET CONTROL AUTOCALL\n");
fprintf(lm20, "SET CONTROL CD 4 0\n");
fprintf(lm20, "SET BT PAGEMODE 4 2000 1\n");
fprintf(lm20, "SET BT CLASS ff0000\n");
fprintf(lm20, "SET BT ROLE 0 f 7d00\n");
fprintf(lm20, "SET BT AUTH * 492\n");
fprintf(lm20, "SET PROFILE SPP ON\n");
fprintf(lm20, "SET\n");

while (1)
{
    do {
        input = getc(lm20);
        fprintf(lcd, "%c", input);
        fprintf(lm20, "%c", input+1);
    } while (input != EOF);
    OSTimeDlyHMSM(0, 0, 0, 100);
}
```

*Figure 11 – Code Snippet for Testing Bluetooth Serial Data Transmission*

The above test routine sets up the LM20 module in serial port mode, prints data received through the Bluetooth link to the LCD module, and echoes the data back through the link to the original device, with each character value shifted up by 1. Executing the test using a Bluetooth-paired smartphone with a serial port application yielded the expected results, with the string “Hello World” sent from the phone being displayed on the LCD, and the phone then receiving the string “Ifmmp Xpsme”.

PCM Codec - Basic Functionality: Audio transmission to the phone requires the audio be adapted to “Pulse Code Modulation” (PCM) serial frame format. A codec was implemented in VHDL to generate the required audio input signal, and capture the audio output signal from the Bluetooth transceiver.

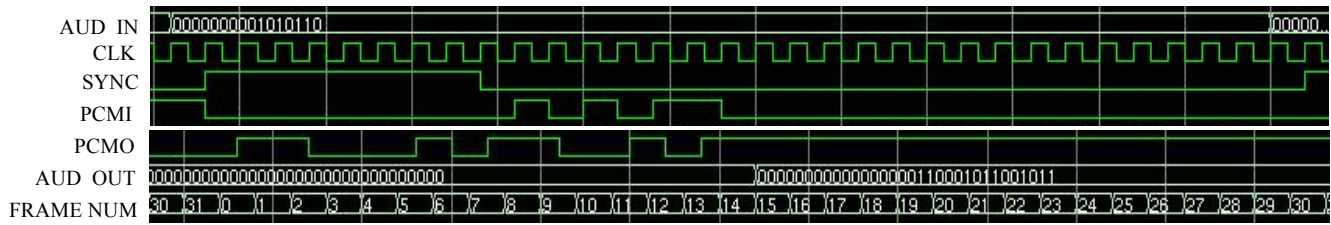


Figure 12 – Simulation Waveforms for Implemented PCM Codec

Figure 12 shows the result of a simulation of our PCM codec implementation. As can be seen, the AUD\_IN signal is latched from memory, and then each bit is sent out in serial along PCMI on the rising edge of SYNC. At the same time, each input bit coming in on PCMO is latched, and then sent out along AUD\_OUT to memory once the entire frame is received.

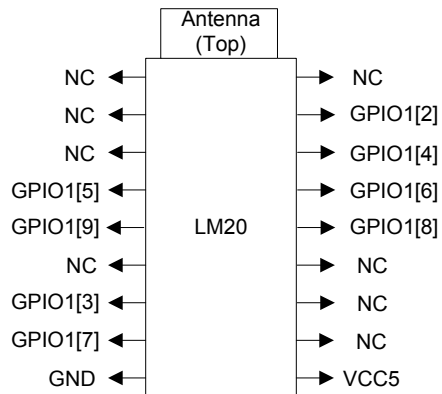
Bluetooth Transceiver - Audio Functionality: In order to stream audio through the Bluetooth module, we had to configure the module into Hands Free mode. This was tested by pairing a phone with the module configured in HFP and examining the status messages that are returned by the Bluetooth module. Once we were satisfied with the status messages, the final integration tests were run by making a call with the paired device and qualitatively evaluating the sound quality, both from the sound observed from the caller and the listener, to observe the quality of PCMI and PCMO. As mentioned above, we were satisfied that the sound quality met the metrics we had determined for this project.



## REFERENCES

1. MathWorks. *Single Sideband Modulation via the Hilbert Transform*. [Online]. Available: <http://www.mathworks.com/products/signal/examples.html?file=/products/demos/shipping/signal/hilberttransformdemo.html>
2. J. Eulenberg and A. Farhad. *Fundamental Frequency and the Glottal Pulse*. [Online]. Available: [https://www.msu.edu/course/asc/232/study\\_guides/F0\\_and\\_Glottal\\_Pulse\\_Period.html](https://www.msu.edu/course/asc/232/study_guides/F0_and_Glottal_Pulse_Period.html)
3. Altera. (2012). *Audio Core for Altera DE-Series Boards*. [Online]. Available FTP: [ftp.altera.com](ftp://ftp.altera.com)  
Directory: /up/pub/Altera\_Material/12.0/University\_Program\_IP\_Cores/Audio\_Video/  
File: Audio.pdf
4. Altera. (2012). *Audio/Video Configuration Core for DE-Series Boards*. [Online].  
Available FTP: [ftp.altera.com](ftp://ftp.altera.com)  
Directory: /up/pub/Altera\_Material/12.0/University\_Program\_IP\_Cores/Audio\_Video/  
File: Audio\_and\_Video\_Config.pdf
5. B.P Lathi and Z. Ding, *Modern Digital and Analog Communication Systems*, Fourth Edition. Oxford University Press, 2009.
6. B.P. Lathi, *Signal Processing and Linear Systems*, Oxford University Press, 2001.
7. University of Alberta. *ECE 492 Lab 2 Manual*. [Online].  
Available: <https://eclass.srv.ualberta.ca/mod/resource/view.php?id=542998>
8. Optrex Corporation. *DMC16207 : LCD Module Specification*. [Online].  
Available: <http://pdf1.alldatasheet.com/datasheet-pdf/view/92672/OPTREX/DMC16207.html>
9. Flexipanel. *LinkMatik 2.0 Bluetooth 2.0 Transceiver*. [Online].  
Available: <http://www.flexipanel.com/Docs/LinkMatik%202.0%20DS379.pdf>
10. Flexipanel. *LinkMatik 2.0 Command Reference*. [Online].  
Available: <http://www.flexipanel.com/Docs/LinkMatik%202.0%20Reference%20DS389.pdf>
11. Wolfson Microelectronics. *Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates Datasheet*. [Online].  
Available: [http://www.wolfsonmicro.com/documents/uploads/data\\_sheets/en/WM8731.pdf](http://www.wolfsonmicro.com/documents/uploads/data_sheets/en/WM8731.pdf)

## **APPENDIX A: QUICK START MANUAL**



*Figure 13 – Schematic for Connecting LinkMatik 2.0 Bluetooth Transceiver to the Voice Manipulator*

### **Setup:**

1. Connect all peripheral components to the DE2 board
  - a. Wire up all the LM20 module to a 40-pin header on a vector board (see schematic)
  - b. Connect 40-pin ribbon cable from header to GPIO\_1 on the DE2
2. Program the FPGA with ECE492\_VM.sof, (Or load ECE492\_VM.pof into flash)
3. Load the ECE492\_VM program into Flash
4. Connect Microphone and Speakers to DE2 MIC IN and LINE OUT jacks, respectively
5. On a cell phone, search for available Bluetooth devices
  - a. Pair with “ECE492\_VM”, using pin code “0492”
  - b. Configure phone to use ECE492\_VM for audio during calls (phone specific)

### **Demonstration:**

1. Direct Mode
  - a. Set switch 0 to UP position
  - b. Talk into the microphone, the audio will be sent through the speakers
  - c. The four push buttons on the DE2 can be used to add effects to the output sound
    - i. Buttons 3 and 2 select the parameter (Volume, Echo delay, Echo decay, and Frequency shift)
    - ii. Buttons 1 and 0 select the value of the parameter
2. Bluetooth Mode
  - a. Set switch 0 to DOWN position
  - b. Initiate a call and ensure audio over Bluetooth device is enabled
  - c. The microphone will be used as the audio input to the paired phone, and the speakers will be used as the audio output from the paired phone, during the call.

Note: We implemented switch 1 as a mute switch to aid with demonstration. Switch 1 must be UP in order to have any audio output

## APPENDIX B: HARDWARE DOCUMENTATION

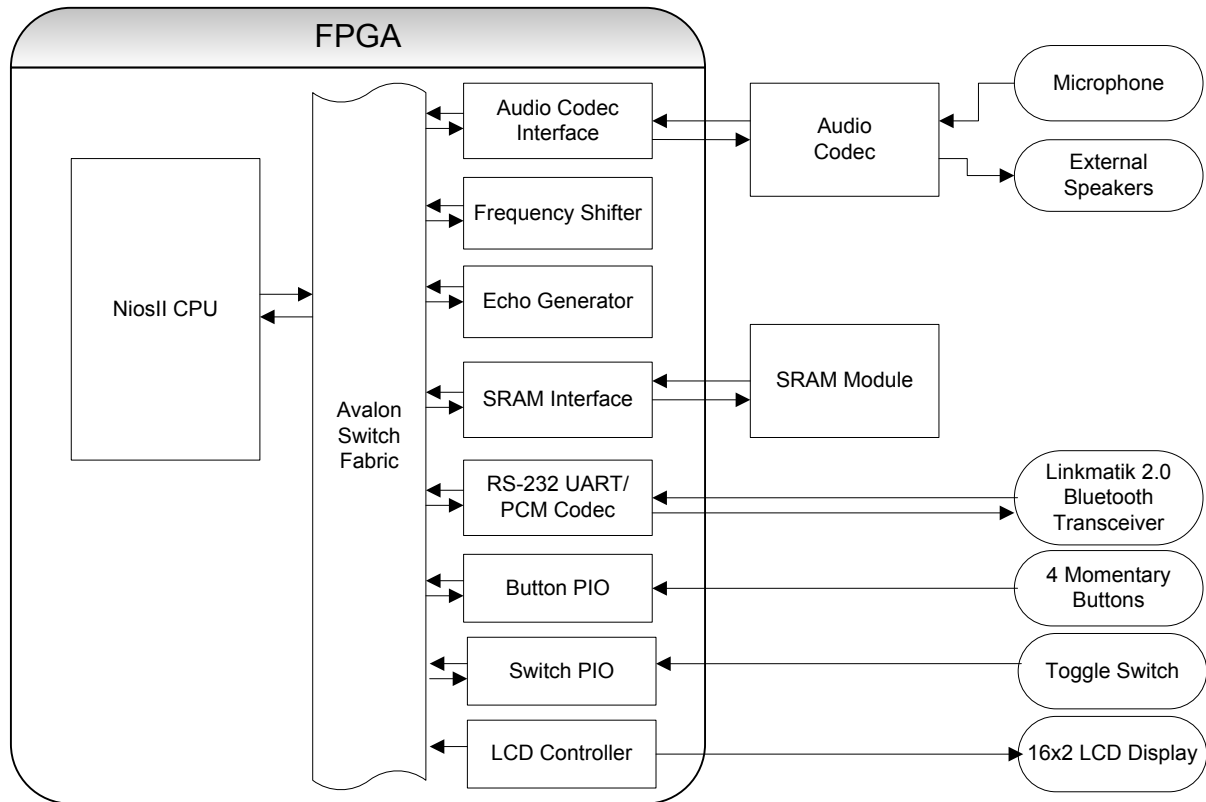


Figure 14 – Block Diagram of Voice Manipulator hardware

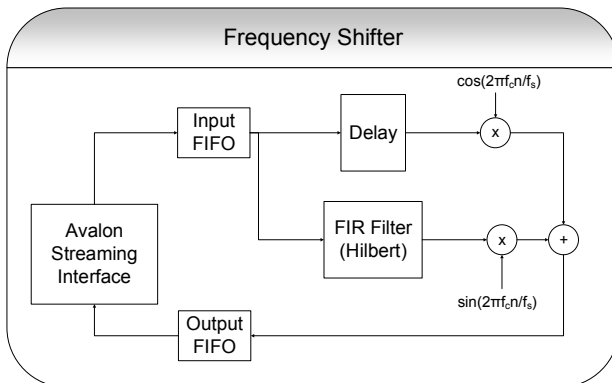


Figure 15 – Block Diagram of Frequency Shifter

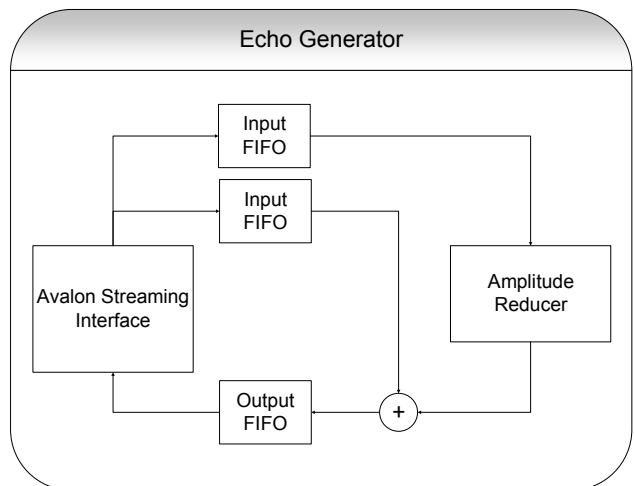


Figure 16 – Block Diagram of Echo Generator

## APPENDIX C: SOURCE CODE

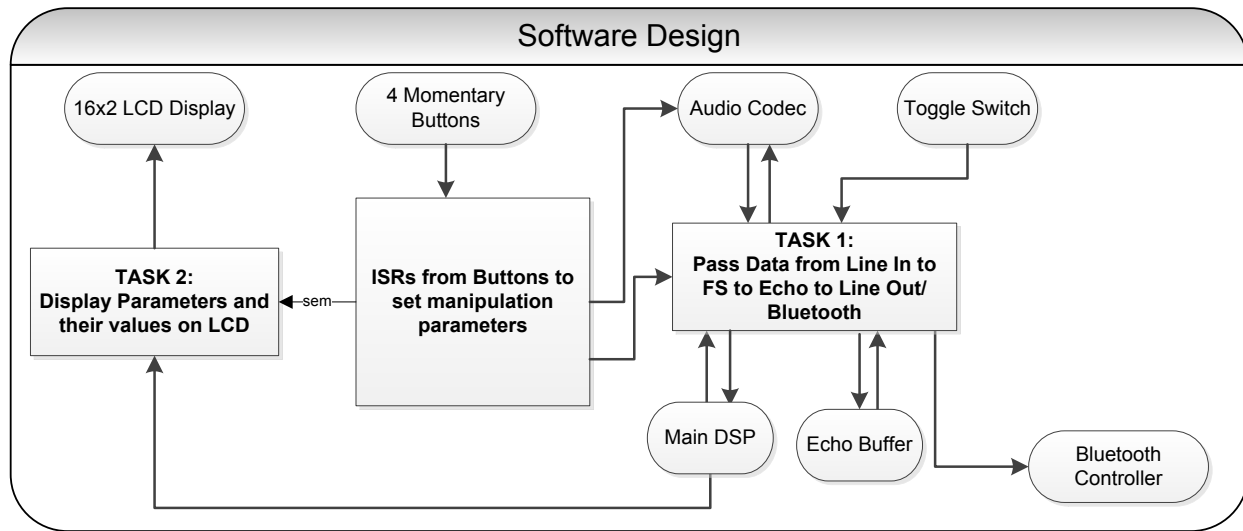


Figure 17 – Software Design for Voice Manipulator

The software for the Voice Manipulator is relatively simple, and can be written in one file without sacrificing readability. See attached .zip for current source code.

### Source Code Index:

File Name	Status	Description
main.c	T	MicroC code defining tasks to be run by the NiosII and interrupt routines
samples.h	T	File containing full period of sine-wave samples for use in software governing frequency shifter
VoiceManipulator.vhd	T	Top-level VHDL file for the FPGA
sram_controller.vhd	T	VHDL file for SRAM controller
pcm_interface.vhd	T	VHDL file for PCM interface to the Bluetooth module.
echo_generator.vhd	T	VHDL file for the echo generator unit
freq_shifter.vhd	T	VHDL file for the frequency shifter unit
resize_tools.vhd	T	Contains VHDL functions that resize bit vectors; used by frequency shifter

## **APPENDIX D: FUTURE WORK**

This project offers multiple paths for expansion. One is to extend the audio manipulation capabilities, to include more effects. Some examples of additional effects are background noise removal or volume levelling.

Another potential upgrade could be to the way the parameters are controlled. The board currently uses the buttons and switches on the board to control the parameters for the transformations, but this could be expanded to use various other input mechanisms instead, such as off-board slider switches/knobs, or even a keyboard.

The echo generator could also be modified to generate a variable number of echoes based on user input, rather than being fixed at one echo. Extra levels of echo volume decay could also be added.

Furthermore, the single sideband modulation frequency shift technique, while effective at shifting frequencies in a linear fashion, does result in some distortion in the output audio due to the destruction of harmonic relationships. A true pitch shift could be implemented instead of the frequency shift used in this project as another avenue of improvement.

Finally, we could also expand on the Bluetooth functionality to create a more complete hands free device. The LinkMatik allows us to send call and answer commands to a paired phone. We were able to implement dialing from the Voice Manipulator, but unfortunately were unable to successfully integrate this functionality into the rest of the project.