ECE532 Final Report
FPGA Karaoke Machine with Real Time Pitch Correction

Bingquan Feng, Harman Gill, Rongbo Zhang, Vimal Raj
Team #5

Table Of Contents

1 Overview

1.1 Background and Motivation

The idea of real-time pitch correction has existed for some time. It was first brought to life by Antares Audio Technologies in 1997 [1]. The device corrects the mis-phonated pitch to its nearest note in equal temperament. It has been a controversial dark magic trick used by vocal performers. After Digital Audio Workstation (DAW), a highly organized software application running on modern computers to record, edit, and produce audio files, proved its success, most pitch correction developers followed the trend and migrated this functionality to either individual software or DAW plug-ins.

Although this technique has been exploited by the music industry, pitch correction applications have not been diffused to amateur music entertainment such as Karaoke. This is because most Karaoke machines, consoles that process effects on digital voice signals, embed Android or Linux where software pitch correction applications are not supported and a hardware pitch correction machine is simply too expensive - a TC Helicon C1 costs half the price of a Karaoke machine.

Hence, our goal with this project is to implement a custom, low-latency, cheap IP that can easily integrate with any microprocessor to achieve real-time pitch correction on singers' voices. To fully mimic its working environment, a karaoke-like interface with microphone input, real-time speaker output, VGA lyrics, and UI display, and user keyboard interactions (eg: selecting songs, pausing) is also to be developed.

1.2 System Overview

The overall system is divided into two subsystems: the pitch-correction audio chain and the karaoke machine interface. Figure 1 displays a block diagram for the system, with custom IP and Vivado IP highlighted. A brief description of each IP is provided in the subsequent sections.

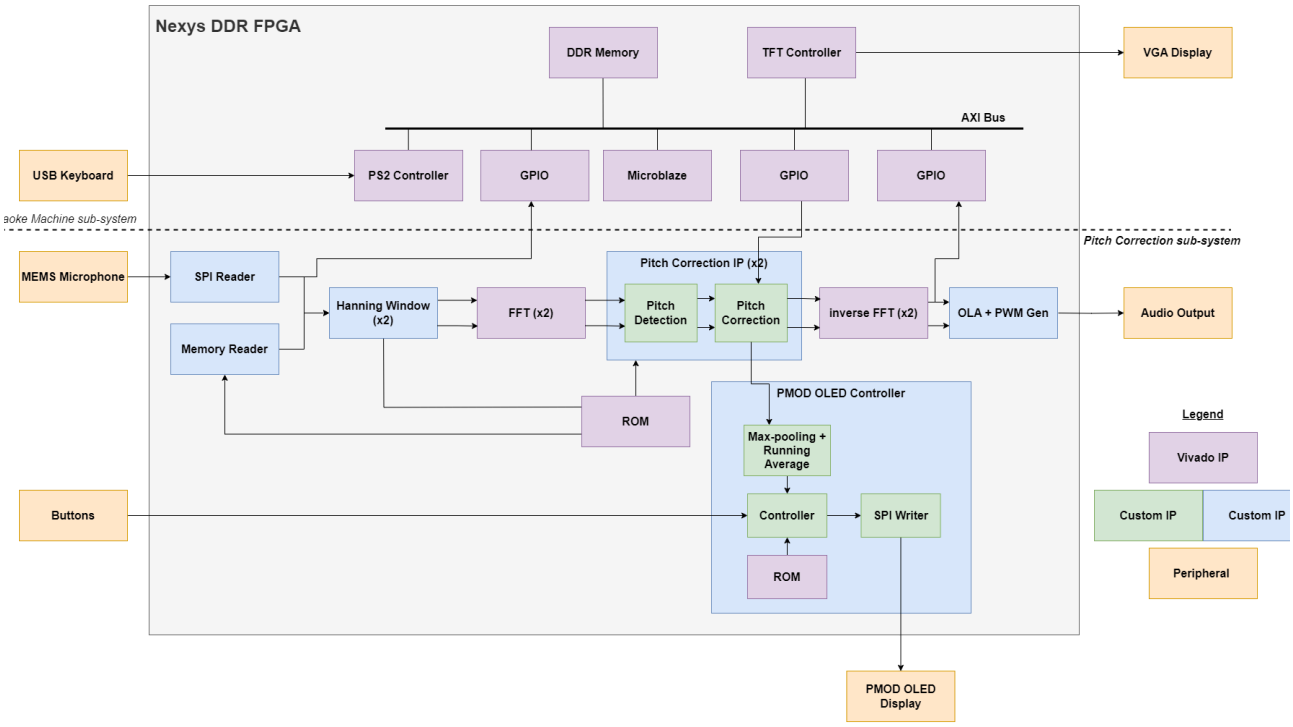

Figure 1: system block diagram

1.2.1 Pitch Correction Subsystem

The pitch correction subsystem is built around the *Xilinx Fast-Fourier Transform v9.1 LogiCORE IP* provided by Vivado [2]. It implements the Cooley-Tukey FFT algorithm. Two of these IPs are used, with one computing a forward FFT and the other computing an inverse IP. Detailed configuration information is provided in section 4.

The rest of the IPs are custom-made using Verilog:

1. **SPI Reader:** reads the incoming serial audio data from the MEMs microphone PMOD and converts it to 12-bit audio samples.
2. **Memory Reader:** reads a stream of audio samples from memory.
3. **Window:** generates two overlapping streams from the incoming audio stream and applies a Hanning window to each 256 sample window.
4. **Pitch Correction:** detects the dominant frequency and its harmonics and shifts the peak to a different, tonally correct frequency
5. **OLA + PWM Gen:** adds the two overlapped streams together and uses a ramp function to generate a PWM signal from the audio samples
6. **PMOD OLED Controller:** takes the frequency domain data before and after pitch correction and displays it on the PMOD OLED screen. Part of the OLED initialization FSM is referenced from the controller for the 128x32 pixel version [3].

1.2.2 Karaoke Machine Subsystem

The Karaoke Machine Subsystem uses a Microblaze core as the controller to display lyrics, input and pitch-corrected PCM signals, and keyboard interactions on the VGA based GUI. It uses AXI4 full protocol for large-bandwidth pixels information transmission between memories. Frames are eventually presented by the TFT controller. A separate BRAM block is instantiated as a ROM to initialize the GUI background with a preprocessed image stored in a coefficient file. All IPs in this subsystem are from Vivado Library.

2 Outcome

2.1 Results

At the end of the project, we have achieved all of the goals laid out in our proposal. Table 1 presents the core and optional features to be implemented, and whether or not they were successfully implemented.

Core Features	**Status**
The GUI must be controlled with a keyboard allowing users to do the following: 1. Select a song from a list of songs provided. 2. Pause a song. 3. Exit to the main menu during playback of a song.	Success
The GUI must display time-synced lyrics for the selected song. (i.e several lines of the lyrics must be displayed for an exact amount of time and then changed to the next lines)	Success
Users must be able to speak/sing into a microphone and hear a pitch-corrected output through a speaker.	Success[1]
Optional Features	**Status**
Display FFT information information on 96x64 OLED display	Success
Visualize pitch correction algorithm (time-domain signal before and after) on VGA display	Success
Store song lyrics in flash memory	Skipped

Table 1: feature (core and optional) requirements

[1] While successfully implemented, the quality of pitch correction is not perfect. Further details provided in section 2.2

Table 2 lists the performance requirements for our system and their completion status. These requirements constraint the performance of the audio processing involved in our project.

Requirement	Metric	Status
The pitch-correction should be done in real-time.	Signal processing latency (measured in milliseconds) must be less than or equal to 20ms (perceived as “real-time”) [2]	Success *< 10ms*
The audio quality should meet or exceed minimum audio standards.	Sampling Rate (measured in KHz) must be 16KHz or higher.	Success *25.6KHz*
	Audio bit depth (measured in number of bits) must be 8 bits or higher	Success *12 bits*

Table 2: performance requirements

2.2 Potential Improvements

While we have successfully met the requirement of being able to produce a pitch-corrected output, the quality of the output audio would generally not be considered good enough for commercial applications. There are several reasons for this:

1. **Lack of filtering:** The audio input from the MEMs microphone is inherently noisy due to the component’s high sensitivity rating. This noise gets further amplified during the FFT process and thus is present in the output. To mitigate this, a band-pass filter must be used that filters out frequencies outside the range of the human voice. This would be the next recommended step if one is to build on top of this project.
2. **FFT window sizing:** Due to resource constraints on our Nexys DDR FPGA, we were limited to using a 256 sample window size. With an input sampling rate of 25.6KHz, this window size gives us a FFT precision of 100Hz. This means that we can map a frequency to any other multiple of 100Hz. However, to perform accurate pitch correction, we would need to be able to shift frequencies with a precision of at least one semitone, which corresponds to 12.5Hz. This would mean increasing our window size to 2048 samples, which would not be feasible resource-wise on our FPGA. However, increasing window size as much as possible would lead to better pitch correction quality.
3. **FFT-induced noise:** The Vivado FFT IP being utilized is documented to add noise to the lower frequencies. However, the male human voice also exists in these lower frequencies (80Hz - 600Hz) and thus gets muddled. Advanced techniques can be used to reduce the noise even further, but are out of the scope of this project. Instead, using

a time-domain pitch correction algorithm such as the TD-PSOLA might have been a better approach for this project, as it preserves the natural timbre of the voice more than the FFT approach, leading to cleaner sounding audio.

4. **Inconsistent VGA display frame rate:** The audio waveform displayed in the GUI is all handled by the Microblaze system. The single threaded microcontroller is having trouble keeping up with the data polling from the microphone and rendering the waveform on the VGA display. The data polling rate is directly tied to the display frame rate, which resulted in significant amounts of data not being captured. This is because the speed of rendering audio waves depends on the number of pixels drawn, which in turn reflects the magnitude of input data. Thus, it creates a situation that the VGA refresh rate is depending on the input data, while the input data is depending on the refresh rate, which results in unpredictable behavior. Current fix is to draw less pixels so that the whole system does not enter this unstable state. If time permits a better solution would be to have a separate hardware block to handle the audio data sampling and waveform rendering, then using a DMA to move the data over to the VGA frame buffer.

Moreover, while working on this project, we realized that the “auto-tune” effect that is heard in pop music is not achieved by turning the pitch correction algorithm on. Instead, there is a lot of artistry involved, as musicians play around with a plethora of parameters controlling the pitch correction effect. Therefore, in future iterations of this project, a more detailed interface, potentially based in software and controlled via a Microblaze processor, should be implemented.

3 Project Schedule

Table 3 displays the original project schedule set in our proposal alongside how the actual work was done.

	Original Milestones	Actual Weekly Milestones
1	● Research pitch correction algorithm ● Find tutorial for MEMS microphone ● Find tutorial for using output audio jack ● Find tutorial for using VGA displays ● Familiarize with SPI bus protocol ● Familiarize with AXI bus protocol	● Researched pitch correction algorithm possibilities ● Researched MEMS and onboard microphones ● Researched Board’s output jack and requirements ● Explored similar existing projects for insights
2	● Algorithm (part 1): Write HDL code for the pitch detection part of the algorithm ● Set up the MEMS microphone peripheral ● Set up the audio output peripheral (time permitting)	● Created direct mic to audio jack pipeline with: ○ SPI interface ○ PCM decoding and conversion to PWM ○ ILAs ○ Volume display on LEDs ● Implemented TD-PSOLA in C code ● Researched HLS and frequency domain alternatives to

		TD-PSOLA
3	● Algorithm (part 2): Write HDL code for the pitch correction part of the algorithm ● Set up the VGA peripheral (if time permitting) ● Set up the keyboard peripheral ● Peripheral verification for audio output and microphone integration	● Explored HLS of TD-PSOLA ● Implemented FSM and buffering to integrate with FFTs ● Testbench to verify frequency domain output ● Implemented Microblaze system to control VGA ● C program to show shape on screen ● Implemented PS/2 receiver block for keyboard input ● 7-segment decoder for displaying input text ● ILAs for keyboard sub-system debug
4	● Verification and debug of the algorithm ● Peripheral verification for VGA & keyboard integration ● Integration and testing of all 4 peripherals	● Testing of FFT block with: ○ Testbench to test the block ○ Python code to generate test inputs to feed into testbench, and code to plot testbench's frequency domain output ○ Confirmed correct operation of block ● Created full frequency domain chain: FIFO to FFT to inverse FFT to FIFO ● Verified reconstruction of matching audio signal on output ● Integrating VGA ○ Implemented double buffering to reduce screen artifacts between frame updates ○ Hardcoded the pixel array for each of the 26 letters ● Integrating keyboard ○ Wrote C code for keyboard interrupt handling ○ Hardcoded the mapping between keycode and character symbols ○ Verified keypress by sending keyboard input to UART console
5	● Improving the algorithm (part 1): add noise filtering to input audio ● Basic GUI for song and options selection ● Peripheral integration verification ● Flash memory for storing musics (if time permitting)	● Full audio chain integration ● Debugging buffering problem in audio chain ● Creating pixel vector for 26 letters on VGA display ● Adding buttons into the GUI
6	● Improving the algorithm (part 2): decrease input to output delay latency ● Debug GUI ● Improve GUI ● OLED 94x64 small display Pmod (additional feature)	● Implemented Hanning Window ● Verified Hanning window through simulation ● Added logic to audio window overlapping ● Implemented peak frequency detection and correction ● Displaying lyrics text on the VGA display ● Displaying audio waveform on the VGA display ● Implemented control for enabling/disabling pitch correction
7	● Final testing ● LED light strips (additional feature)	● Implemented multi-peak pitch shifting ● implemented overlap & adding of DDR windows ● Improved GUI button interaction ● Added GUI background ● Improving Lyric display (without music sync) ● integrated PMOD OLED ● Overall system integration and testing

Table 3: Original vs. actual milestones

In the pitch correction subsystem, the main variation from the original plan is due to the full frequency domain audio chain taking longer than expected to work correctly. While subsections were quickly verified to perform correctly in simulation, debugging to ensure this in synthesized hardware took longer. In addition, the large amount of noise being generated by the FFT blocks necessitated significant work to minimize it before the pitch correction itself could be tested on the board. This led to the reordering of tasks, with the audio chain including additional blocks such as the Hanning Windows and a dual FFT - iFFT chain layout to obtain a better audio output through overlapping.

The development of the Karaoke subsystem was mostly on track, but used up all the buffer time. Thus, no additional features were added to the subsystem.

4 Detailed Block Descriptions

4.1 Pitch Correction Subsystem

4.1.1 SPI Reader

The SPI reader block contains a clock divider, a shift register, and unsigned to signed converting logic. The clock divider creates a 235x slower clock from the 100MHz clock. The slower clock is used to control the SPI communication with the microphone. The 12-bit PCM data takes 17 slow clock cycles to be captured, which leads to an audio sampling rate of around 25KHz. The data from the microphone is unsigned 12-bit data, so a conversion to signed format is needed in order to feed into the FFT block.

4.1.2 Memory Reader

The memory reader block is the subsequent block from the SPI Reader, containing the code to route data to the two staggered audio chains and an alternate option to stream audio from a stored memory location. The former is done via an initial counter that waits until 128 audio samples have been sent to the first chain before asserting the valid signal for the second chain. As the same data is going to both chains, but one begins filling while the other is already half full, it produces two staggered audio windows. The streaming from memory is accomplished by reading data from a file into an array (similar to Assignment 2) and using a mux to choose between the SPI reader's output and reading the memory array, with a counter to simulate various sampling rates. The block also houses the Hanning Window block, one instantiation for each audio chain.

4.1.3 Hanning Window

The Hanning window is a scaling factor that is applied to every audio sample within the window. The scaling factors range from 0 to 1, with the first and last (256th) samples being

multiplied by a factor of 0 and the midpoint (128th) sample being multiplied by a factor of 1. To maximize efficiency, the factors are not being calculated in hardware, instead being loaded into a memory array from a file. They are then multiplied with the incoming samples, with a counter keeping track of the samples. As the second 128 of the scaling factors are a reverse ordering of the first 128, only the first 128 are stored.

Figure 2 shows the Hanning window block working as intended in simulation, applied to an incoming sine wave signal.

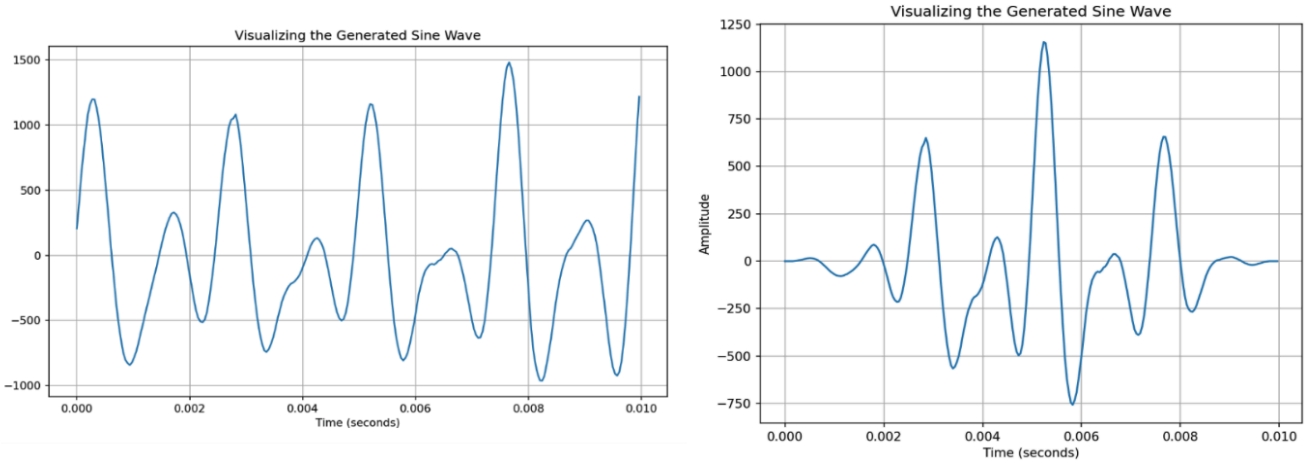

Figure 2: Hanning window in simulation

4.1.4 Fast Fourier Transform

Used the *Xilinx Fast-Fourier Transform v9.1 LogiCORE IP* provided in the Vivado library. Configuration changes were updating the data width to be 12 bits, selecting AXI Stream as the communication protocol and setting the window size to 256. It is repeated twice in the design, one instantiation for each audio chain.

4.1.5 Pitch Correction

The pitch correction block, present between the FFT and inverse FFT (iFFT) blocks, performs the frequency shifting in the audio subsystem. It is connected via AXI Stream buses to the preceding and subsequent Fourier IPs, and also provides monitoring signals to view the original and pitch corrected frequency domain signals on the OLED display.

The block is built around a central FIFO buffer of size 256 (the window size) which collects the complex number outputs from the FFT and upon having the complete window data it begins feeding it to the iFFT. This is achieved through an FSM that keeps track of the buffer's fullness and asserts the appropriate signals. On receiving each data sample the magnitude is also calculated, with the index of the largest value being recorded. The indexes of the harmonics are

multiples of this dominant frequency and the target frequency to which the dominant frequency is to be shifted. When the buffer is being emptied to the iFFT, the read pointer is monitored and for indexes corresponding to the dominant frequency and target frequency the read location is updated to instead retrieve data from the other's index. The same process is applied for the harmonics. The results of this process can be seen in Figure 3.

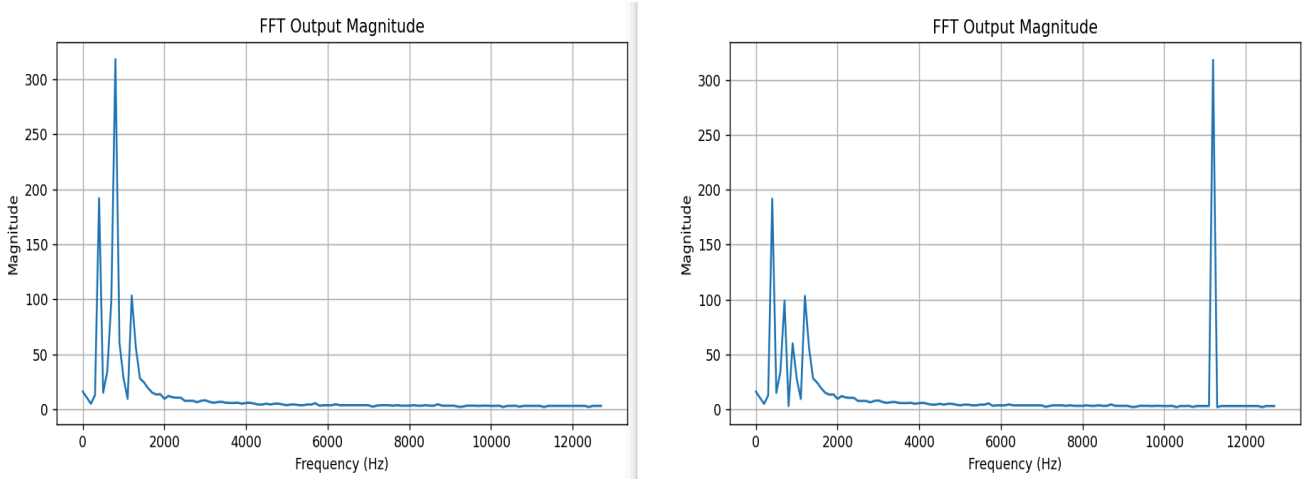

Figure 3: Pitch correction algorithm in simulation

The block also produces the AXI config channel signals used to configure the subsequent FFT block to perform inverse fourier transforms.

4.1.5 Inverse Fast Fourier Transform

Used the *Xilinx Fast-Fourier Transform v9.1 LogiCORE IP* provided in the Vivado library. Configuration changes were updating the data width to be 12 bits, selecting AXI Stream as the communication protocol and setting the window size to 256. It is repeated twice in the design, one instantiation for each audio chain.

4.1.6 Overlap and PWM Generation

This IP receives two incoming audio signals from the inverse FFT blocks. To maintain the 128 sample overlap that was created in the memory reader block, an initial counter is used to ensure a ready signal to the second chain is only asserted after 128 valid-ready handshakes have been completed from the first chain. After, the incoming audio from each chain is added together and clipped to ensure it is still a 12-bit signal. Figure 4 shows this working in simulation with a sine wave. Due to the nature of the Hanning window and the 50% overlap, the original signal is reconstructed perfectly.

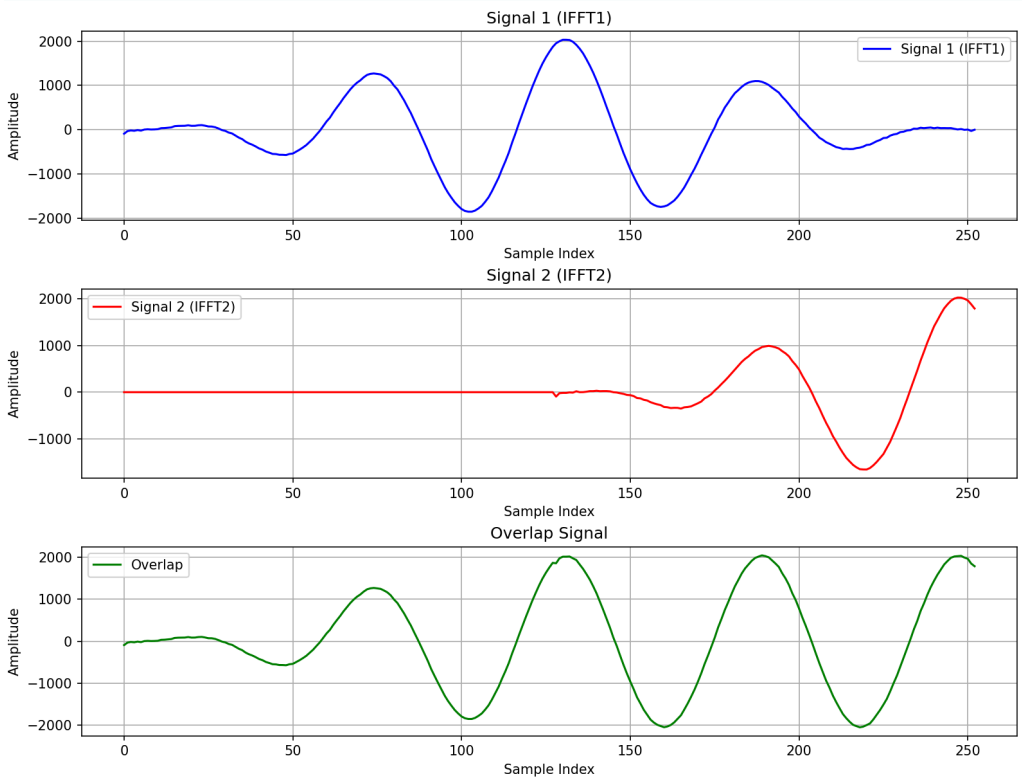

Figure 4: Overlap + add in simulation

This overlapped and added audio signal is then used to generate a PWM signal, which goes to the audio output port and is converted into an analog wave using the FPGA's built-in low-pass filter. To generate the PWM signal, each audio sample is compared to a ramp function, implemented as a counter that counts up to 4096. The output signal is asserted high if the audio sample is greater than the current count, and low otherwise. This effectively sets the duty cycle in the PWM signal for each audio sample.

As each audio sample has to be compared against the entire counter, it takes 4096 cycles of the fast 100MHz clock to process one sample. This leads to a PWM frequency of ~25KHz. This is ideal for two reasons:

1. This frequency matches the sampling rate of the microphone, ensuring that output is generated at the same rate as the input. This ensures that the buffers present inside our FFT blocks don't get filled and cause stalls in our processing pipeline.
2. The frequency is high enough to appropriately represent audio in the 0 - 2.5KHz range, which covers the human vocal frequency range.

4.1.7 PMOD OLED Controller

The PMOD OLED Controller block takes in the 32-bit frequency domain data before and after the pitch correction and displays the log scale spectrum vertically on the OLED display. The block consists of 2 main parts, PMOD control logic and data manipulation for spectrum plotting.

The PMOD has a specific startup sequence and power off sequence to increase the longevity of the device. The power off sequence is short enough to be hardcoded. The startup sequence has 32 steps, so a simple FSM is constructed, as shown in the Figure 5, to load 15-bit init-sequence OP-Codes that have the structure indicated in Table 4. The PMOD control block is built in reference to the controller for the 128x32 pixel version [3]. Because the startup sequence is different, a different OP-Code sequence is coded into the ROM. The full startup sequence can be found in Appendix A. The packaged IPs in the referenced design are failing during synthesis, so all the packaged IP blocks are removed. This project only needs to draw lines and rectangles, so the draw commands are hard coded in the FSM for convenience.

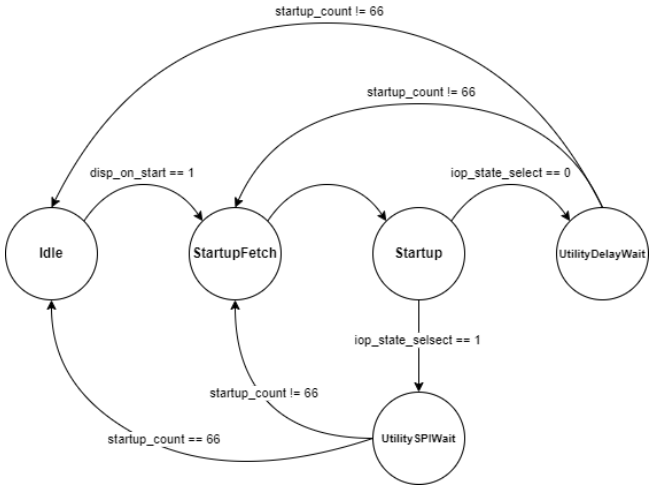

Figure 5: OLED startup sequence FSM

14	13	12	11	10	9	8	7:0
state_select	RESET_set	RESET	PMODEN_set	PMODEN	VCCEN_set	VCCEN	Data

Table 4: OLED startup sequence OP-Code

For the 32-bit frequency domain data, the lower 16 bits is the real number and the upper 16 bits is the imaginary number. The datapath has been pipelined in the following stages:

1. Converting from signed number to unsigned number
2. Calculating the squared value

3. Adding the squared real number and imaginary number
4. Finding the index of the leading 1 to get a fast estimate of logarithm
5. Max Pooling with window size of 4
6. Averaging with value from previous window for smooth transition
7. Storing the value into an array

The OLED update logic is constantly reading from the current value array while keeping track of previous value to calculate the delta for updating the content being displayed on the OLED screen.

4.2 Karaoke Machine Subsystem

4.2.1 PS2 Keyboard controller

Used the PS2 controller ver1.0 from the Vivado library. No configuration change was made. The ports are directly connected to the PS2 keyboard input. The interrupt handling is referenced from an example project [4].

4.2.2 VGA display controller

The TFT controller accepts a 25MHz clock for VGA to refresh at 60Hz. It controls its local registers to store the current frame pixel information. Five external output pins are extended from the TFT controller, which are R(4b), G(4b), B(4b), hsync(1b), and vsync(1b). All output pins use LVCMOS33 as the voltage domain, and connecting to package pins {A4, C5, B4, A3}, {A6, B5, A5, C6}, {D8, D7, C7, B7}, B11, and B12 respectively. The interrupt signal is connected to the microblaze interrupt controller, but was never used in this project.

4.2.3 Microblaze Core

The 32b wide microblaze core has 32KB instruction memory communicating through SLMB interface, and connecting to the 128MB SRAM for data storage. Operating at 100MHz, it processes the main C code for the control loop to maintain GUI functioning.

4.2.4 Peripherals

- 128MB SRAM instantiated from mig, accepting 200MHz clock.
- 2 GPIOs that transfer 12b input / corrected PCM signals to the microblaze core.
- 8KB BRAM that stores GUI background.

4.2.5 GUI

- keyboard.h, keyboard.cpp which stores keyboard interrupts mappings, keyboard initiation functions.

- vga.h, vga.cpp which stores 52 alphabets matrix, GUI load up functions, PCM drawing function, PCM window, lyrics drawing function, lyrics window.
- main.c which infinitely loops over a C-coded FSM. This FSM has four states representing four buttons - previous song, next song, pause song, replay song. By scanning for keyboard “enter” input, each state executes the same action as its name indicates. State transitioning is controlled by the up, down, left, and right button.
- main.c also switches microphone input / memory audio by key “1”, and enables FFT bypass and pitch-correction bypass by key “2” and “3”.

5 Design Tree

The github repository contains three separate breakdown modules: 1) PMOD_OLED, 2) TFT_keyboard, and 3) pitch_correction. The final_demo is the integrated version of all sub-modules. A detailed file description is in the readme. Only the most important files are listed here:

- ➔ PMOD_OLED/Packaged_IP/src/ // custom IP
  - ➔ NexysVideo_Master.xdc
  - ➔ OLEDCtrl.V
  - ➔ SpiCtrl.v
  - ➔ debouncer.v
  - ➔ delay_ms.v
  - ➔ delay_us.v
  - ➔ Init_sequence.mem
  - ➔ top.v
- ➔ TFT_keyboard/TFT.srcs/constrs_1/imports/TFT/constrs.xdc // Vivado IP
- ➔ pitch_correction/pitch_correction.srcs/ // custom IP
  - ➔ constrs_1/imports/new_pwm/Nexys_4_DDR_constraint.xdc
  - ➔ sources_1/new/
    - ➔ PWM.sv
    - ➔ PWM.v
    - ➔ buff_fft_wrapper.v
    - ➔ hann.sv
    - ➔ output_buf.sv
    - ➔ overlap_add.sv
    - ➔ overlap_pwm_wrapper.v
    - ➔ pitch_correction.v
    - ➔ spi.v
- ➔ final_demo/TFT_keyboard/ // final design

- ➔ TFT.srcs/
  - ➔ sources_1/rgb.coe
  - ➔ constrs_1/imports/TFT/constrs.xdc
- ➔ final_demo/TFT_keyboard/TFT.sdk/karaoke_full_system/src/
  - ➔ VGA.h
  - ➔ VGA.cpp
  - ➔ keyboard.h
  - ➔ keyboard.cpp
  - ➔ main.cc

➔ vivado_library/ip/* // Vivado IPs

The GitHub repository can be accessed here: https://github.com/RonZ13/ECE532_Project.

6 Tips & Tricks

A critical piece of information missing from the Nexys DDR FPGA reference manual is the existence of an amplifier enable signal for the audio output port. This signal is connected to pin D12 on the FPGA and must be tied high for the audio port to work. This piece of knowledge would have saved us a couple of hours of debugging.

Moreover, it was quite difficult to debug the audio chain due to the nature of the analog audio data and the frequency domain representation. We got around this obstacle by writing out simulation data (frequency domain and time-domain signals) to text files through our SystemVerilog testbench. This data was then read by various Python scripts and plotted, helping us visualize what is happening and speed up the debugging process.

Another point to note is that the FFT IP provided by Xilinx produces a lot of noise in the lower frequency bins. As such, it is better suited for higher frequency signals since they are more easily separated from the generated noise. For audio projects - especially those focusing on voice - using a custom FFT block found online may provide better results than the IP available in Vivado.

Lastly, DSP projects such as this one are quite hard to implement easily. Vivado provides a tool called “System Generator” that links with Simulink inside Matlab and provides a more interactive approach to designing DSP designs for the Xilinx FPGAs. It may be worth looking into this tool.

7 References

[1] D. Hughes, "Technological Pitch Correction: Controversy, Contexts, and Considerations," Journal of Singing, vol. 71, pp. 587, 2015.

[2] *Fast Fourier Transform v9.1 LogiCORE IP Product Guide* (2022) *AMD Technical Information Portal*. Available at: https://docs.amd.com/r/en-US/pg109-xfft/Fast-Fourier-Transform-v9.1-LogiCORE-IP-Product-Guide (Accessed: 12 April 2024).

[3] “Pmod OLED - Digilent Reference,” *digilent.com*. https://digilent.com/reference/pmod/pmodoled/start?redirect=1 (accessed Apr. 12, 2024).

[4] “Nexys-Video-AXI-PS2-Keyboard/sdk/appsrc/demo.c at master · Digilent/Nexys-Video-AXI-PS2-Keyboard,” *GitHub*. https://github.com/Digilent/Nexys-Video-AXI-PS2-Keyboard/blob/master/sdk/appsrc/demo.c (accessed Apr. 12, 2024).

Appendices

Appendix A: PMOD OLED Startup Sequence

```
INITIALIZATION SEQUENCE: (contained in init_sequence.mem)
1   Turn DC logic low                              delay 1ms       1   0001
2   Turn RES logic high                            delay 1ms       2   3001
3   Turn VCCEN logic low                           delay 1ms       3   0201
4   Turn PMODEN logic high                         delay 20ms      4   0C14
5   Turn RES logic low                             delay 1ms       5   2001
6   Turn RES logic high                            delay 1ms       6   3001
7   Send unlock command1                           (hFD)           7   40FD
7   Send unlock command2                           (h12)           8   4012
8   Send display off command                       (hAE)           9   40AE
9   Send remap command1                            (hA0)           10  40A0
9   Send remap command2                            (h72)           11  4072
10  Send set start line command1                   (hA1)           12  40A1
10  Send set start line command2                   (h00)           13  4000
11  Send set offset command1                       (hA2)           14  40A2
11  Send set offset command2                       (h00)           15  4000
12  Send no inversion command                      (hA4)           16  40A4
13  Send set mux ratio command1                    (hA8)           17  40A8
13  Send set mux ratio command2                    (h3F)           18  403F
14  Send use external VCC command1                 (hAD)           19  40AD
14  Send use external VCC command2                 (h8E)           20  408E
15  Send disable power saving command1             (hB0)           21  40B0
15  Send disable power saving command2             (h0B)           22  400B
16  Send set charge rate phase length command1     (hB1)           23  40B1
16  Send set charge rate phase length command2     (h31)           24  4031
17  Send set oscillator frequency command1         (hB3)           25  40B3
17  Send set oscillator frequency command2         (hF0)           26  40F0
18  Send set pre-charge speed of Red command1      (h8A)           27  408A
18  Send set pre-charge speed of Red command2      (h64)           28  4064
19  Send set pre-charge speed of Green command1    (h8B)           29  408B
19  Send set pre-charge speed of Green command2    (h78)           30  4078
20  Send set pre-charge speed of Blue command1     (h8C)           31  408C
20  Send set pre-charge speed of Blue command2     (h64)           32  4064
21  Send set pre-charge voltage command1           (hBB)           33  40BB
21  Send set pre-charge voltage command2           (h3A)           34  403A
22  Send set voltage level for logic high command1 (hBE)           35  40BE
22  Send set voltage level for logic high command2 (h3E)           36  403E
23  Send set master current attenuation command1   (h87)           37  4087
```

```
23  Send set master current attenuation command2    (h06)           38  4006
24  Send set contrast for Red command1              (h81)           39  4081
24  Send set contrast for Red command2              (h91)           40  4091
25  Send set contrast for Green command1            (h82)           41  4082
25  Send set contrast for Green command2            (h50)           42  4050
26  Send set contrast for Blue command1             (h83)           43  4083
26  Send set contrast for Blue command2             (h7D)           44  407D
27  Send disable scrolling                          (h2E)           45  402E
28  Send clear screen command1                      (h25)           46  4025
28  Send clear screen command2                      (h00)           47  4000
28  Send clear screen command3                      (h00)           48  4000
28  Send clear screen command4                      (h5F)           49  405F
28  Send clear screen command5                      (h3F)           50  403F
29  Turn VCCEN logic high                           delay 25ms      51  0319
30  Send turn on display command                    (hAF)           52  40AF
31  Wait                                            delay 100ms     53  0064
32  Finished                                                            DONE
// custom sequence //
33  Send enable fill command1                       (h26)           54  4026
33  Send enable fill command2                       (h01)           55  4001
34  Send draw rectangle command1                    (h22)           56  4022
34  Send draw rectangle command2                    (h00)           57  4000
34  Send draw rectangle command3                    (h00)           58  4000
34  Send draw rectangle command4                    (h5F)           59  405F
34  Send draw rectangle command5                    (h3F)           60  403F
34  Send draw rectangle command6                    (hFF)           61  40FF
34  Send draw rectangle command7                    (hFF)           62  40FF
34  Send draw rectangle command8                    (hFF)           63  40FF
34  Send draw rectangle command9                    (hFF)           64  40FF
34  Send draw rectangle command10                   (hFF)           65  40FF
34  Send draw rectangle command11                   (hFF)           66  40FF
35  Wait                                            delay 100ms     67  0064
```