6.111 Project Proposal

Audio-Controlled Levitation System

David Mejorado and Raul Largaespada

1. Project Introduction
2. Audio-Frequency Detection System
3. Basic Levitator Implementation
4. Possibilities for Expansion
5. Block Diagrams
6. Other Figures

1. Project Introduction

We wanted to implement a project that combined both audio processing and controls, therefore we will be building an audio-controlled levitator. The labkit's audio Ac97 will be used to obtain an audio signal that we will generate with either a tone generator or an instrument, like a piano. The main frequency of this audio signal will then be extracted via the Fast Fourier Transform (FFT) and be used as the reference signal to a PSD control system. The levitator will consist of a ping-pong ball like object that will be placed inside a cylinder. A fan at the bottom of the cylinder will act as the actuator moving the ball up and down, while a IR sensor placed at the top of the cylinder will provide measurements on the current position of the ball.

2. Audio Frequency Detection System

In implementing a system that can extract the main frequency components there are three things that have to be considered and accounted for. The first being that the signal of interest will not be the only signal being recorded therefore some preprocessing must be considered. Secondly, since we must always generate a valid reference signal for the control, post-processing and logic must be in place to ensure that no unwanted noise or lack of signal disturbs the physical system. Lastly, the FFT can be implemented from scratch or through an IP core provided by Xilinx.

Filter:

Designer: David
Inputs: 8 bit PCM data from ac97
Outputs: 8 bit filtered audio
Description: This module will interface with the ac97 and prepare the signal to undergo the FFT. I will begin the implementation with a 31-tap low pass filter, but this number might change as we begin to test the module. The ac97 is out-putting 48k samples per second. I am planning on downsampling this data to 14k samples per second to be able to retain frequencies of up to 7k Hz.
Testing: In order to test this module I will generate various tones and sound files and perform the wanted operations on matlab and verify that the output of this module is correct. This will be done with modelsim for a visual representation of the filtering and I will also store this data and perform closer verification in Matlab.

FFT:

Designer: David

Inputs: Filtered signal

Outputs: Ready

Description: This module will sample a window of size N of the incoming samples and store them to a B-RAM. Then a discrete fourier transform will be performed on this windowed data. The output of this will be N bins corresponding to different frequencies. A new set of transformed data will be outputted at a window sampling rate, $f_{w.}$ One way in which I might implement this is by using what is none as a "butterfly" operator which involves signed multiplications and additions. An alternative to this would be to use the ISE IP core to implement the FFT, however further investigation needs to be done before a decision is reached. Parameters for this FFT will also be finalized by performing some experimentation in matlab simulation. The binned frequency analysis will be stored to a memory and a ready signal will be output every time a new sample is ready.

Testing: I will supply a known signal into this module and save its output. I will then verify that this output is producing the correct transform. I will also designate specific areas in the process to verify that the timing of events are appropriate and that intermediary steps are correct.

Frequency detection:

Designer: David

Input: ready

Output: Status, peak

Description : This module will detect whether or not there is a valid frequency tone being received by the microphone. It will do this by searching for a peak in frequency bins starting at the frequency bin of the previous sample. This will be done by applying a threshold that a bin must reach in order for there to be a valid signal. If a peak is detected this module will also output which bin number has a peak.

Testing: I will input known sample and verify that its performance is in according to the data that is being inputted.

Bin2freq:

Designer: David

Input: Bin number

Output: Freq

Description: This module will be dedicated to taking a bin number and performing the necessary operations need to turn it into an understandable

frequency value in Hz. This value will then be quantized into a predetermined range of frequency. This new value is what will then be outputted.
Testing: In order to test this I will input known values and verify throughout this process that the correct operations and results are being generated by the module.

Reference Generator:

Designer: David
Input: Freq, Status
Output: Reference Signal
Description: This module will serve as a lookup table and control for what is being outputted as a reference signal to the levitator. It will take in the Freq input that will serve as an address to a specific 8 bit height location to be sent as a reference signal. The status signal well be used to determine whether the reference signal should be directly from the audio input or if it should send a resting default reference signal when there is no valid audio input.
Testing: The behavior of this module will be testing by running a basic testbench in modelsim to verify that it is functioning as intended.

Visualisation:

Designer: David
Input: Freq
Output: Display
Description: This module will serve as a real time display of information regarding the signal being received. This will include a display of the the transformed input as well as the peak frequency. If time permits I will also include a visualization of the desired height output as well as a spectrogram of the continuous audio input.
Testing: This will be tested on the lab monitors with a VGA cable. Known data will be tested first to verify that the visualization are being generated correctly.

3. Levitator Implementation

The basic levitator implementation will involve a single levitator with the output of the audio-frequency detection system being the system input and the height of the levitated object being the system output. It will be a single input single output (SISO) system controlled using a discrete Proportional-Integral-Derivative (PID) controller, known as a Proportional-Sum-Delta (PSD) controller. The levitated object will be a ping-pong ball, or ideally a cylinder of similar weight, within a 3D printed tube to

restrict movement to one axis. The controller will make use of a filtered signal from an analog IR sensor to determine the height of the levitated object. The control signal will be converted to a PWM duty cycle, which will drive a fan.

External Components:

Cylindrical tube: 3D-printed at EDS
Ping-Pong ball/hollow cylinder: to be purchased
Fan: acquired from Joe Steinmeyer, interfaces with MOSFET output
IR Sensor (GP2Y0A21YK): acquired from EDS, interfaces with ADC
Battery (7.4 volts): acquired from EDS
MOSFET (IRLZ14): acquired from EDS, interfaces with PWM generator on FPGA
Analog-Digital Converter (ADS1015IDGSR): acquired from Digikey or EDS, interfaces with FPGA via I2C
Basic ADC: acquired from EDS, interfaces with FPGA via I2C
Potentiometers: acquired from EDS, interface with FPGA via simpler ADC

PSD Controller

Designer: Raul Largaespada
Inputs: reference signal r[n], filtered sensor output y[n], tuning offsets
Outputs: control signal u[n]
Description: This is a discrete PID controller that subtracts the height outputted from the filtered IR sensor signal from the control reference signal from the audio-frequency detection system to create an error signal e[n]. This signal is then fed into the proportional, sum, and delta terms. Finally, the controller will sum the output of each of these terms into a control signal u[n].
Testing: To test, we will manually set the control reference signal to a variety of heights and see how quickly the system reaches the appropriate height, with what degree of overshoot, and if there are any offset errors. We will also use potentiometers to manually tune the controller gains within a short range from the hard-coded values. The error signal will need to be saved as a signed integer value. The width of the input signals will be 8 bits.

Proportional Term:

Designer: Raul Largaespada
Inputs: error signal e[n], proportional term tuning
Outputs: proportional term p[n]
Description: This module multiplies the error signal by a proportional gain term K_p to get the proportional term output. This output is proportional to the magnitude of the error, and will force the control signal to be more positive or

negative depending on the sign of the error signal. The full equation for the proportional term is below.

$$p[n] = K_p e[n]$$

Testing: We will display the inputs and outputs of the module with a hard-coded error signal to make sure the math occurs correctly, and will attempt to stabilize the system using proportional control only to determine its effectiveness.

Sum Term:

Designer: Raul Largaespada

Inputs: error signal e[n], sum term tuning

Outputs: sum term s[n]

Description: The sum term is meant to serve as a discrete integrator, summing up all previous error terms multiplied by the sampling time, which is set at 1/20 seconds because this is the speed at which the IR sensor can change outputs. This module would have to remember the sum of all previous error signals as a signed integer value. This sum would have a maximum and minimum value specified to prevent integrator windup. The full equation is below.

$$s[n] = K_i \sum_{i=1}^{k} e[n]\Delta t$$

Testing: We will test this module individually to make sure the calculated value is correct, and in conjunction with the proportional term module to create a PS controller.

Delta Term:

Designer: Raul Largaespada

Inputs: error signal e[n], delta term tuning

Outputs: delta term d[n]

Description: This module is a discrete approximation of a derivative, subtracting the current error signal value from the previous value and dividing by the sampling time. To divide, will will use a generated divider module, and will normalize to the width of the other two term module outputs. The full equation is below.

$$d[n] = K_d \frac{e[n]-e[n-1]}{\Delta t}$$

Testing: We will test this individually to ensure the calculated output is correct, and in conjunction with the proportional and sum terms to characterize our full PSD controller system.

PWM Generator:

Designer: Raul Largaespada

Inputs: control signal u[n]

Outputs: PWM signal v[n]

Description: This module will normalize the control signal value into a PWM duty cycle value ranging from 0-100. This value will be used to output a PWM signal from the FPGA into the fan, allowing it to levitate our object.

Testing: We will hard code a control signal and scope the output to make sure our duty cycle is at the correct value.

A2D Converter:

Designer: Raul Largaespada

Inputs: IR height signal h(t)

Outputs: Discrete height h[n]

Description: This will be a physical IC that will convert the analog reading from the IR sensor into an 8-bit value. We will communicate with this IC using the I2C protocol and convert the 8-bit value into an integer height.

Testing: To test, we will input specific voltages into the ADC and make sure our output values match the height that voltage would correspond to.

Kalman Filter:

Designer: Raul Largaespada

Inputs: Discrete height h[n]

Outputs: Filtered height y[n]

Description: This filter will use a Gaussian distribution to filter the noisy IR sensor data into a more accurate value. This value will then be fed into the controller as the actual height of the levitated object. To do this, we will save the previously estimated state in memory and use that in conjunction with the current reading from the A2D to generate a more accurate estimate of the current state.

Testing: We will compare Kalman filter outputs with physically measured levitator heights after we stabilize the system using feedforward control before closing the loop.

4. Possibilities for Expansion

If our initial levitator control implementation is successful, we will move to using two levitators to balance a ball at a specific height determined by the audio frequency. This will involve the use of two controllers to control the height of each levitator, and a separate controller to adjust the values of the levitators to balance the ball in the center. This will be a multi-input multi-output system, with the inputs being the audio frequency and the distance the center ball is from each levitator tower, and the outputs being the height of each levitator. This will require additional control analysis and a new control scheme.

5. Block Diagram

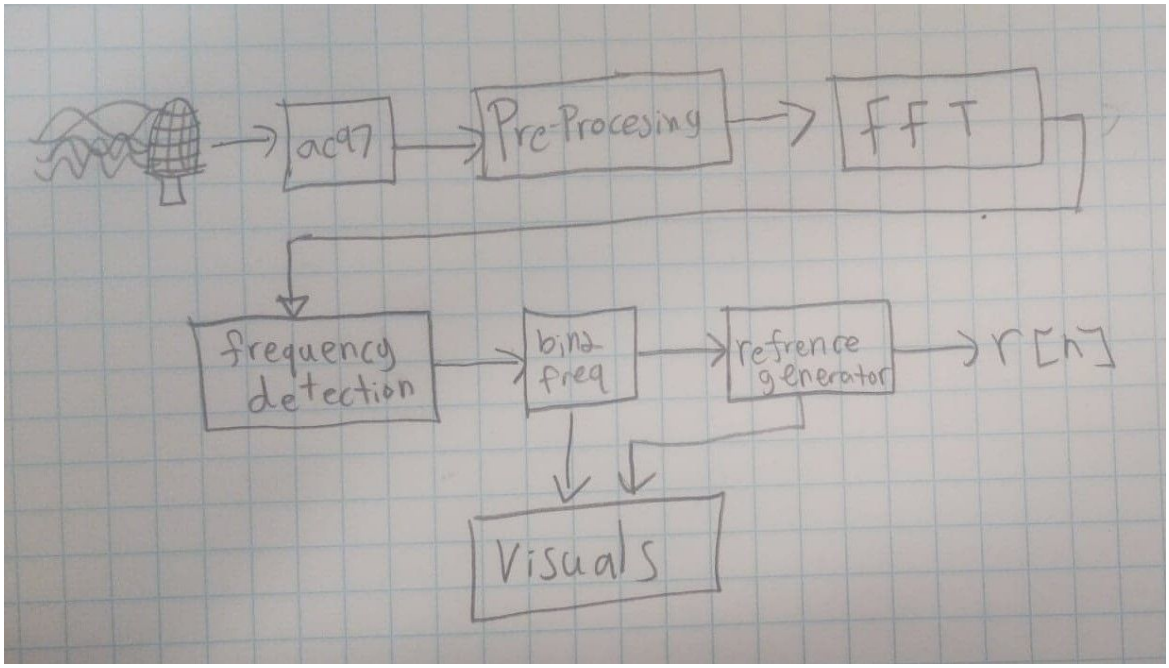

Figure 1. Block diagram for the audio frequency detection system.

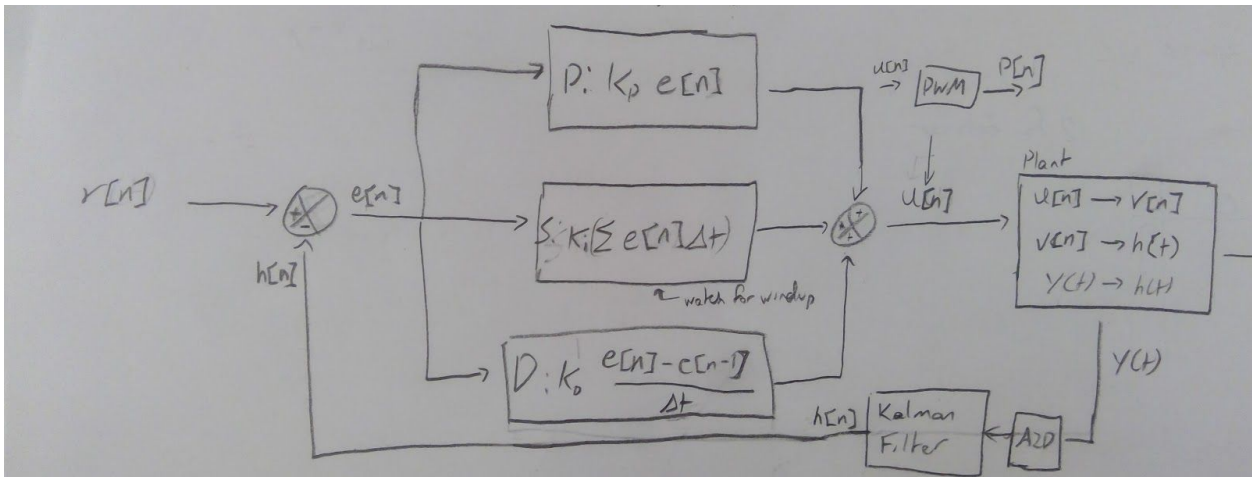

Figure 2. Block diagram for the PSD controller.

6. Additional Figures

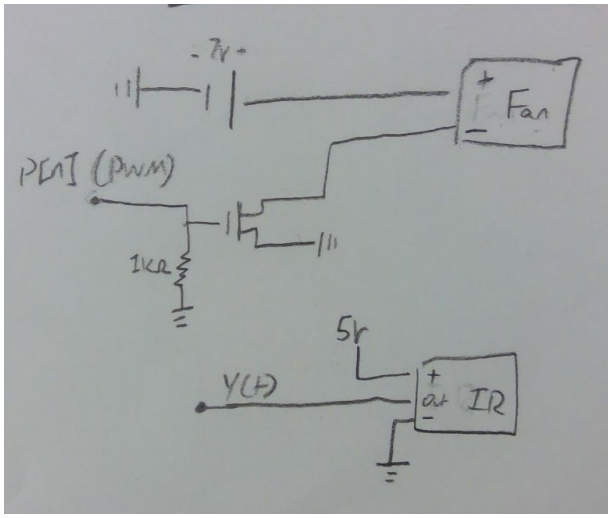

Figure 3. Circuit diagram for the IR circuit and fan circuit.