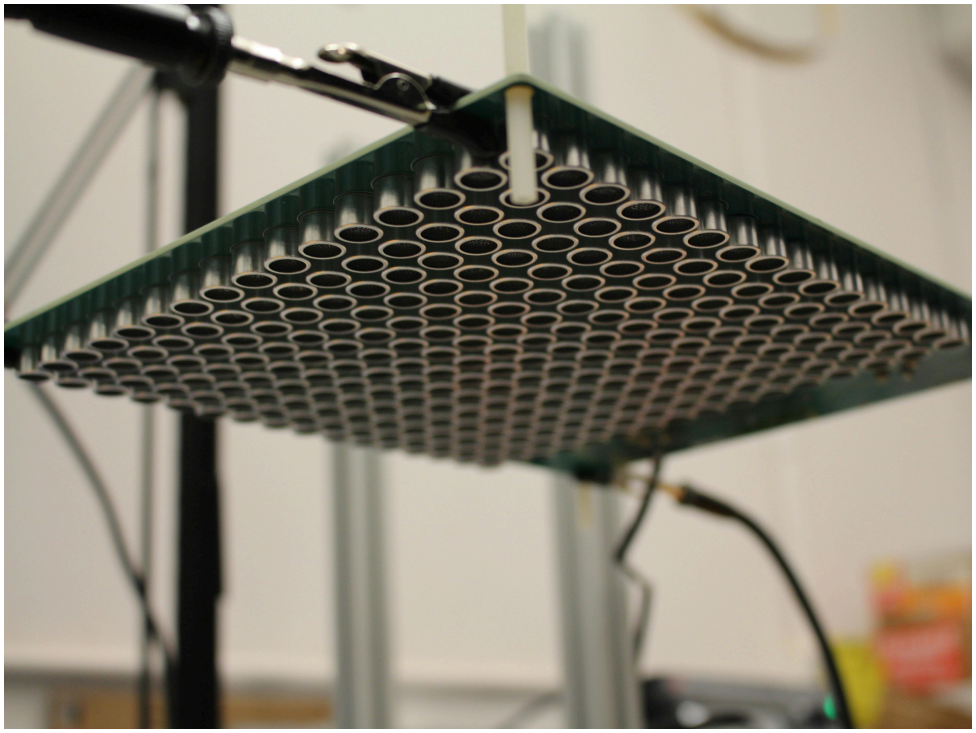
Pushing the Frontier of Open-Source Ultrasonic Phased Arrays for Multi-Modal Stimuli



Andrew Chen, Quinn Ferbers, Kevin Lin, James Seto

Project 2365
ENPH 479
Engineering Physics Project Lab
The University of British Columbia
April 16th, 2024

Executive Summary

Ultrasonic phased arrays have been the focus of dozens of research papers spanning many fields. Although many of these papers introduce state-of-the-art abilities using these arrays, most do not share specifics about their engineering designs, much less any usable instructions to recreate their findings. Our goal was to recreate some of the phenomena seen in these papers, primarily focusing on the different modes of interaction that the phased arrays can produce. To do this, we started with an open-source project with all the information and files we needed to build a basic but usable board. Using what we learned from the open-source project, we created a custom system. Our custom system includes a custom 10-layer PCB, a custom implementation of physics Solvers, Python and Rust user input software, and a dual-FPGA hardware layer with logic designed with SystemVerilog. We reproduced the high-speed levitation of styrofoam beads, hand-detectable haptic feedback, and human-perceptible sound via modulation. Despite our success, our system had some stability issues and did not perform at the level we intended in our design. As such, we have recommendations for further development and improvements to our design. Overall, we were able to thoroughly learn about the engineering design problems in this increasingly popular field of research and create interesting demonstrations of multi-modal stimuli. We also intend to provide all the files and documentation of our project, pushing the open-source baseline forward.

Table of contents

Table of figures

Introduction

Ultrasound is defined as sound that is higher than 20kHz frequency, outside the human hearing range. Transducers are devices that can emit or receive sound, and ultrasonic transducers are cheap and commonly used, making them easy to buy in bulk and work with. Ultrasonic transducers can be used for a variety of applications related to stimulating the senses, which we will discuss more below.

Levitation

When a single transducer emits ultrasound at a reflecting surface, it creates a longitudinal standing wave. The Gor'kov potential, an approximate metric for the force experienced by a particle, is proportional to the **mean-squared pressure** at a point (Bruus, 2012) [1]. Since the nodes of the longitudinal waves have constant pressure, they have the lowest Gor'kov potential and thus are prime locations to trap small, lightweight beads. By increasing the number of transducers, forming an array with them, and modulating their relative phases (hence "ultrasonic phased arrays", or "phased array ultrasonics"), the morphology, strength, and location of traps can be controlled finely. Updating the phases of transducers synchronously and rapidly is key to moving particles quickly. Styrofoam beads, being light, cheap, and close to the 1/10th wavelength recommendation from Ospina et al. are the particles of choice for levitation experiments (2022) [2].

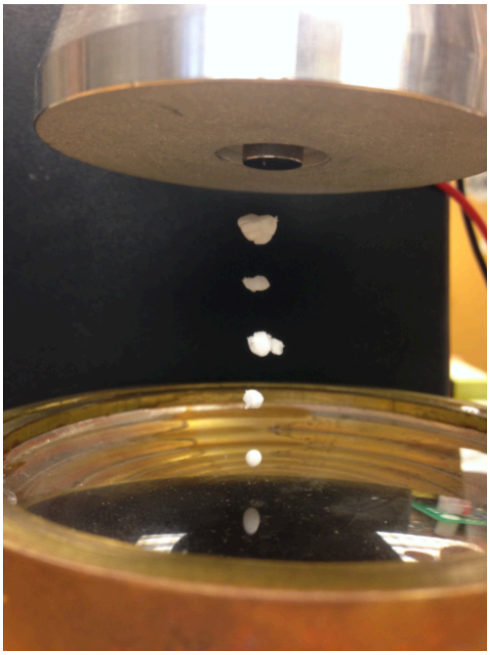


Figure 1: An image of a single transducer levitating multiple styrofoam shapes. This demonstrates that just a standing wave is enough for levitation along 1 axis. Picture by Luke Wortsmann, 2016 [3]

Haptic Feedback

Acoustic pressure traps are disturbances in the air that can be felt by human skin. Some research, such as (Sun et al., 2021) [4], suggests that the ultrasound should be modulated to a lower frequency to be felt, but some other research such as (Long et al, 2022) [5] simply uses unmodulated 40kHz ultrasound while successfully creating tactile sensations felt by test subjects. An important distinction is that Long et al. move the position of the particle, whereas Sun et al. seem to imply that the trap can be left stationary and still be felt. Low-frequency change (on the order of 200 Hz) seems to be the key to any kind of tactile feedback.

Human-Perceptible Audio

Although ultrasound is not normally audible to human ears, there are various ways of modulating the signal to the transducers such that the sound can be heard. Further, since

ultrasound has a higher frequency than audible sound, it has lower attenuation and dispersion, meaning that it can be transmitted to a listener who is far away with high fidelity. Due to this effect, a listener standing next to the target listener may not hear anything at all.

Putting It Together

Combining all the discussed abilities so far, ultrasonic phased arrays can create visual, auditory, and tactile stimulation, which have many applications in modern technology, such as pharmaceutical testing, cancer treatment, and human-computer interfaces. Although it does not yet exist, an impressive product that leverages ultrasonic phased arrays could function as a 3-dimensional display, doing everything a 2-D monitor could do and more. Our goal for the rest of this report is to explain our efforts to recreate these phenomena.

Discussion

SonicSurface

Our team approached development in two phases. Initially, we assembled an open-source ultrasonic phased array, the SonicSurface [6]. The SonicSurface supported a 16×16 transducer array, driven by a single CoreEP4CE6 FPGA development board. The I/O for the FPGA is multiplexed, allowing for a high count of outputs with relatively simple electronics. The project had PCB fabrication files available, and some Java software that allowed us to start levitating light styrofoam beads. The goal for this initial revision was to set up a board known to reliably perform ultrasonic levitation, allowing for software iteration to start without the long delays involved with electrical hardware development. Additionally, we used it to explore the phenomenon of ultrasonic levitation and helped build intuition for the technology. The combination of open-source hardware and custom software is referred to as Revision 0.

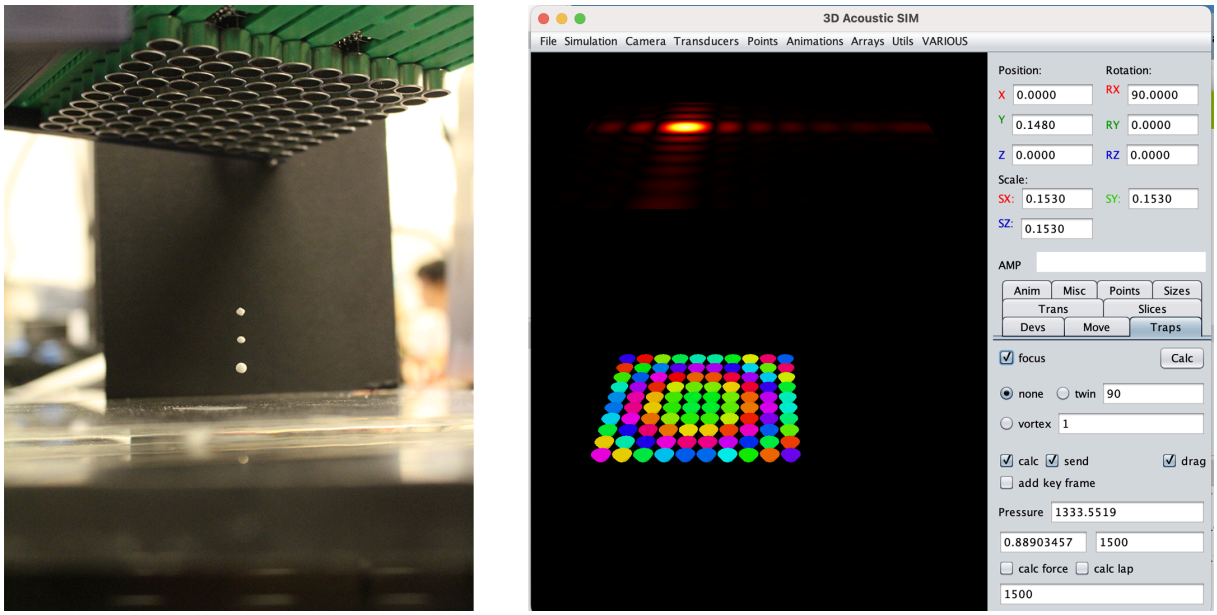


Figure 2: Left: our built version of the SonicSurface, levitating 3 beads. Right: Open-source software “Ultraino” allowing for the placement of a trap on a 2D slice a set distance away from the board.

Experimenting with SonicSurface for several months, we learned the following:

1. We estimated the update rate of SonicSurface to be around 100 Hz - FPGA multiplexing and using the Arduino Nano’s CH340 chip to convert Serial input to the FPGA seemed to be major limiting factors for the update rate of the board
2. Although some research papers used a 10×10 grid, it did not feel powerful enough for haptic feedback and fast levitation

3. Controlling motion in 3D by fixing 1 axis, as the open-source software does, was unsatisfactory for creating interesting motion
4. It was difficult to locate the position of traps in real life after sending them in software

Using these findings and additional literature review, we developed a custom ultrasonic phased array system from scratch.

Rev. 0 to 1

We developed a custom PCB, developed new digital logic, and we interfaced the hardware with the software developed using Rev. 0 to create Revision 1, which is the focus of this report. Rev. 1 aimed to improve all aspects of Rev. 0 to achieve state-of-the-art performance. We based the target performance on Hirayama et al.'s phased array design, highlighted with an update rate of 10k FPS[1] as a key metric for enabling the features of their proprietary setup, specifically the creation of holograms through rapid movement of particles (2022) [7]. In addition to levitation, Hirayama et al. also demonstrated audible sound generation and tactile feedback using ultrasonic phased arrays, and all three simultaneously could be achieved by rapidly swapping between the three modalities within the duration of a single wavelength of the driving signal to the transducers (2019) [8]. Below, we will discuss the design considerations we made to reach the 10k FPS requirement and how we recreated the three abilities of levitation, tactile feedback, and audible sound. The main focus of the design discussion will be levitation since it is the most technically constraining feature.

Rev. 1 Requirements

We settled on the following key requirements for Rev. 1:

- 256 transducer channels driven with 40 kHz square waves with a precise, configurable relative phase
- A phase resolution of 256 divisions per period (~100 μs divisions)
- A frame update rate of 10 kHz

The key requirements also drove the required bandwidth between the host and board. With a rough calculation, the estimated bandwidth requirement is 4.8MB/s:

$$\text{Bandwidth} \sim (\text{Update Rate}) * (\#\ \text{Transducers}) * (\text{Phase Resolution})$$

Solver

A "Solver" refers to the software that solves the required phases of the transducers to levitate one or multiple beads at certain locations. The points at which these beads levitate are called "traps". There are many solvers in the literature, and we compared the performance of a few prominent ones to satisfy our needs.

Design

After a review of the literature and some rough prototyping, the two most promising algorithms were found to be the HAT algorithm (A. Marzo et al., 2019) [9] and the Gor'kov algorithm (Hirayama et al., 2022) [7]. Both algorithms use an iterative approach: the HAT algorithm uses the Gerchberg-Saxton method derived from optics to solve for the transducer phases that maximize the average pressure just above the trap, whereas the Gor'kov algorithm uses gradient descent to solve for the minima of the modified Gor'kov potential:

$$U'(r_j) = K_1|p|^2 - K_2|\frac{\partial p}{\partial z}|^2$$

where K_1, K_2 are constants that depend on the physical system, and p is the complex pressure. Both algorithms use the piston source model [10] to calculate the acoustic field of a transducer. For both algorithms, the reflection of the transducer waves off the reflecting plate below the

[1] FPS = frames per second. One frame = all transducers set to their desired phases.

transducer board is simulated by mirroring the transducers across the reflecting plane, keeping their phases identical.

Comparing the two algorithms, the Gor'kov algorithm theoretically produces stronger traps than the HAT algorithm, but in practice, the outputs of both algorithms are very similar. The derivative in the modified Gor'kov potential requires at least 2 points to be sampled per trap over the single point per trap of the HAT algorithm, so the Gor'kov algorithm is generally slower than the HAT algorithm. Gradient descent can also be less stable than the Gerchberg-Saxton method due to the possibility of getting stuck at local minima when learning rates are too large or small. In practice, the Gerchberg-Satxon algorithm tends to converge in only a few iterations, while gradient descent takes 10 to 20. Because of the limited benefits that the Gor'kov algorithm provided from our testing, we picked the HAT algorithm, favouring the lower processing requirements to hit our target of a 10 kHz update rate.

Although the solver algorithms are agnostic to our choice of programming languages, speed was an important consideration when choosing one due to our 10 kHz update rate. The compiled languages that members of our team are familiar with are C, C++, and Rust. Speeds between these three languages are similar depending on implementation details, so our team went with Rust to gain more experience with the language and to have confidence in the thread safety of our application.

Evaluation

Once we implemented the HAT algorithm, we evaluated the algorithm and performed some simulations to explore and verify various phenomena encountered while testing the solver with the SonicSurface board.

The Rust implementation of the HAT algorithm reached speeds of 14k updates per second on a Dell XPS 13 laptop after small optimizations. This speed increased to 100k updates per second using a multi-threaded batch processing approach, scaling nearly linearly across 8 cores. Our HAT algorithm therefore handily managed to exceed our target 10kHz update rate, allowing for real-time processing and interactive demos.

A simulation was implemented in Rust and used it to analyze different trap arrangements and situations. A typical example of a generated trap can be seen in Figure 3. Note that all mentions of "Pressure" refer to the magnitude of pressure deviation from ambient pressure. From this figure, we can deduce that the HAT algorithm generates high-quality traps without explicitly solving the full Gor'kov potential. Of note is that the trap is made of many traps layered on top of each other due to the natural interaction of acoustic waves. This makes it challenging to confirm that a particle is trapped in the strongest of the traps, although it could be remedied by placing a particle in each generated trap.

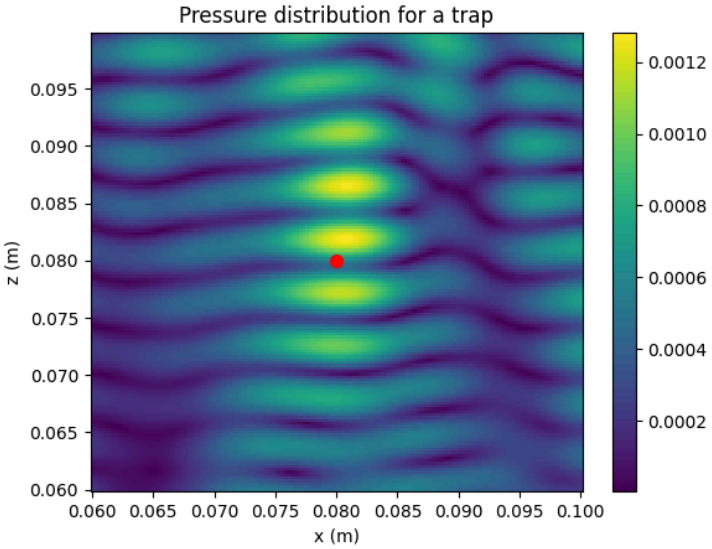


Figure 3: Simulated pressure distribution for a trap. The red dot sits where the trap is intended to be. Although HAT only solves for an average pressure maximum, a trap gets formed due to the standing waves produced by the reflections off of the base plate. The pressure levels are relative; useful for comparison between simulations but they are only proportional to the physical pressure values.

The setup itself is relatively sensitive to misalignment, as shown in Figure 4. Even just a 2-degree misalignment can produce traps at only 50% of the intensity of a properly aligned setup. There is also a phenomenon where traps near the reflecting base plate fail to move upwards in the z-axis (Figure 5). This occurs due to the boundary conditions imposed by the reflection and makes it challenging to produce traps that break the reflecting base plate's natural interference pattern.

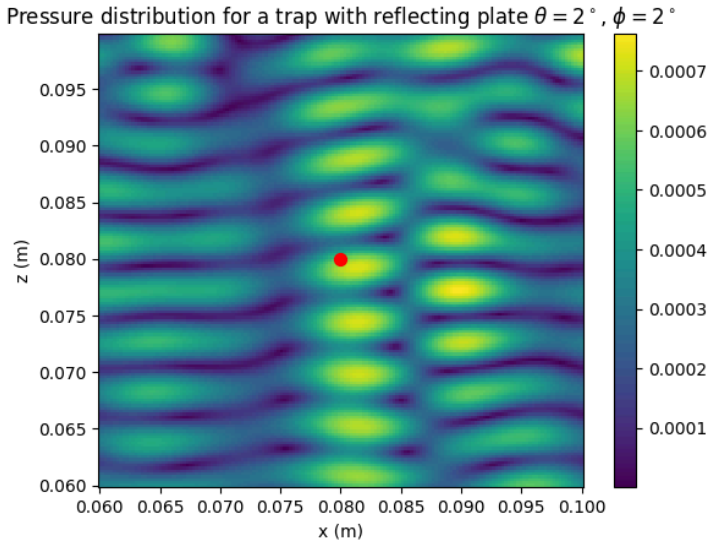


Figure 4: Simulated pressure distribution for a trap with a misaligned base plate. only a 2-degree shift in both directions drops the maximum pressure by nearly 50% compared to Figure 3. Accurate alignment is thus critical for performance, especially at larger distances between the transducers and the base plate.

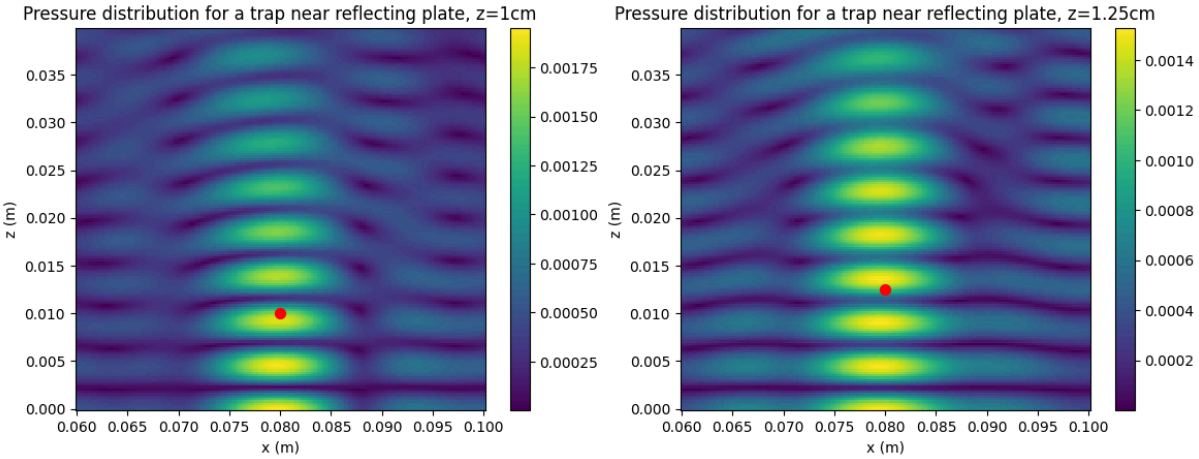


Figure 5: Simulated pressure distribution for two traps near the reflecting base plate. The boundary conditions established by the reflecting plate cause the standing wave to always have a maximum at the reflecting plate surface, which inhibits the ability to lift particles in traps near the plate. The two traps shown as red dots are at z = 1cm and z = 1.25 cm, but it's clear that the actual standing wave doesn't properly produce traps at these heights, being limited by the standing wave pattern enforced by the boundary. This makes it challenging to move particles along the z-axis, especially for low values of z.

Software Design

The role of the solver is to convert desired particle positions to phases. The rest of the software in our system either allows the user to provide positions to the solver or provides other utility/ enables some other feature of the system.

Position Input

The open-source software used a keyframing system to allow the user to pre-set the trajectory that the trap/bead would follow. Although this was a valid approach, we felt it wouldn't be a fun demonstration of levitation if the only paths the bead could take were preset, without customization, and slow to modify. So, we developed the following methods of creating positions:

1. Simple geometry (Rust): paths that are easy to describe mathematically, such as a circle with constant velocity or a bead moving along a line in simple harmonic motion. The user can control parameters such as the frequency of motion, radius of the circle, amplitude of the line SHM, etc. using the terminal.
2. Hand tracking (Python): using just a webcam, the user can control the position of the user's hand (including depth) to create a position in 3D. We performed this hand-tracking using the pose-estimation features of the MediaPipe library. Without a depth camera, the depth tracking was poor and led to discontinuous positions, so this mode was not used much, although controlling a particle's position with your hand is highly practical and interesting and worth further exploration with better 3D cameras.
3. Mouse tracking (Python): the position of the bead is controlled by the mouse position and scroll wheel, using the simple GUI features of PySide6. The high update rate and continuous nature of the mouse allow smoother motion than the hand tracking while maintaining the satisfaction of rapid, responsive user input.
4. Shape contouring (Python): with the eventual goal of creating holograms, we created scripts that converted solid-colored shapes into a list of points on their contours (using OpenCV). These points could be traversed by a particle quickly or played simultaneously on a surface such as salt or water to display them. We tested the shape contouring algorithms on letters, symbols, and frames of the famous Bad Apple!! music video.

The output of each of these systems was simply a stream of 3D coordinates in the range of the board, approximately 0-10 cm in all three axes for SonicSurface and 0-11.26, 0-11.26, 0-12.9cm for Rev. 1.

Blender and Redis

To address the difficulty of not knowing where the acoustic traps were located, we used the simulation/animation software Blender to display the current desired position of the acoustic trap(s). We created a small microservice architecture to facilitate communication. At the center, we used the in-memory and fast software Redis as a publisher-subscriber system. Positions are published from one of the input methods described above and sent to Blender for visualization and to the Solver for eventual output to the phased array board.

Figure 6 summarizes this architecture:

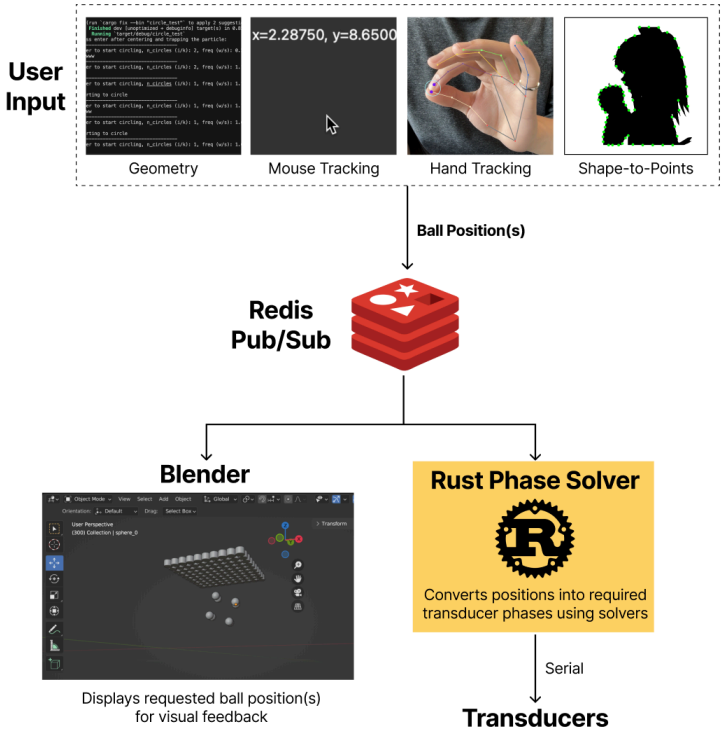


Figure 6: Software system architecture for levitating styrofoam beads. Note that only one position input is used at once. Also, note the omission of some non-trivial code that sends the phases generated by the solver to the boards.

Redis, being in-memory, is known to be very fast, and should not be a limiting factor in our design. This pub/sub architecture also allowed us to have abstract interfaces; positions could be published from Python rather than Rust, enabling easier prototyping and integration with advanced Python libraries. We also made pure Rust scripts that skipped Redis and Blender, sending geometry-defined positions defined in Rust directly to the solver. These were marginally more performant since they allowed us to solve for phases in batches, but it seemed this was the only source of the speed-up i.e. Redis is probably not a bottleneck.

Haptic Feedback

To achieve tactile sensations on human skin, we used the mouse control capability of the levitation mode of our board, as if we were levitating a particle 2-3cm from the bottom of the reflecting surface (where the hand is placed). Most people reported a sensation comparable to wind or static electricity, and most felt greater sensations on their palms than on the backs of their hands. Mouse control functioning well for haptic feedback aligns with our understanding of the phenomenon, as the trackpad/mouse we used had polling rates of 125 Hz and 500 Hz respectively, close to the recommended 200 Hz.

Human-Perceptible Audio

To produce audible sound, we digitally modulated all transducers at once. Digital modulation means turning off the transducer output for half a specified period, then oscillating the transducers at the carrier frequency, 40 kHz, for the other half. The modulation introduces frequency content at the modulation frequency. For instance, specifying a modulation frequency of 440Hz produced an A4 note with a square-wave timbre. To play music, we used scripts to decode Musical Instrument Digital Interface (MIDI) files live, converting notes to modulation periods, and sending commands to the FPGA. Each FPGA could play a note independently, so we leveraged the two FPGAs on board to play two melodies simultaneously. Higher polytonality could probably be achieved by modulating more groups of transducers independently.

Digital Logic Design

Rev. 0 (SonicSurface) Architecture Overview

The SonicSurface uses serial communication at 115200 bauds to speak from a host to the board. The 256 transducer channels are managed by an off-board Cyclone FPGA by writing to 32 multiplexors controlling 8 transducers each.

The major advantage of this design is a simple PCB. The design uses an FPGA development board and an Arduino as a serial device instead of integrating ICs onto the PCB.

The first limitation is the low phase resolution of 32 divisions per period. The multiplexing scheme on the SonicSurface required a shift clock at 10.24MHz, and a latch clock at 1.28MHz[2]. The phase resolution can be increased if we increase the shift frequency, but two concerns arise. First, the hardware design has a maximum clock speed, as register setup and hold times must be met. Second, high-frequency trace effects become a design consideration, such as trace radiation, capacitive coupling, radiated coupling, and attenuation from impedance mismatches. Azier et al. recommend better filtering for traces with signals above 200 MHz (2021) [11].

The second limitation is that the serial communication is capped at tens of kilobytes. This bandwidth is insufficient for our targeted 10 kHz frame rate.

[2] Shift clock: 10.24MHz = 8 channels per multiplexor * 40kHz * 32 divisions per period. Latch clock: 1.28 MHz = 40kHz * 32 divisions per period

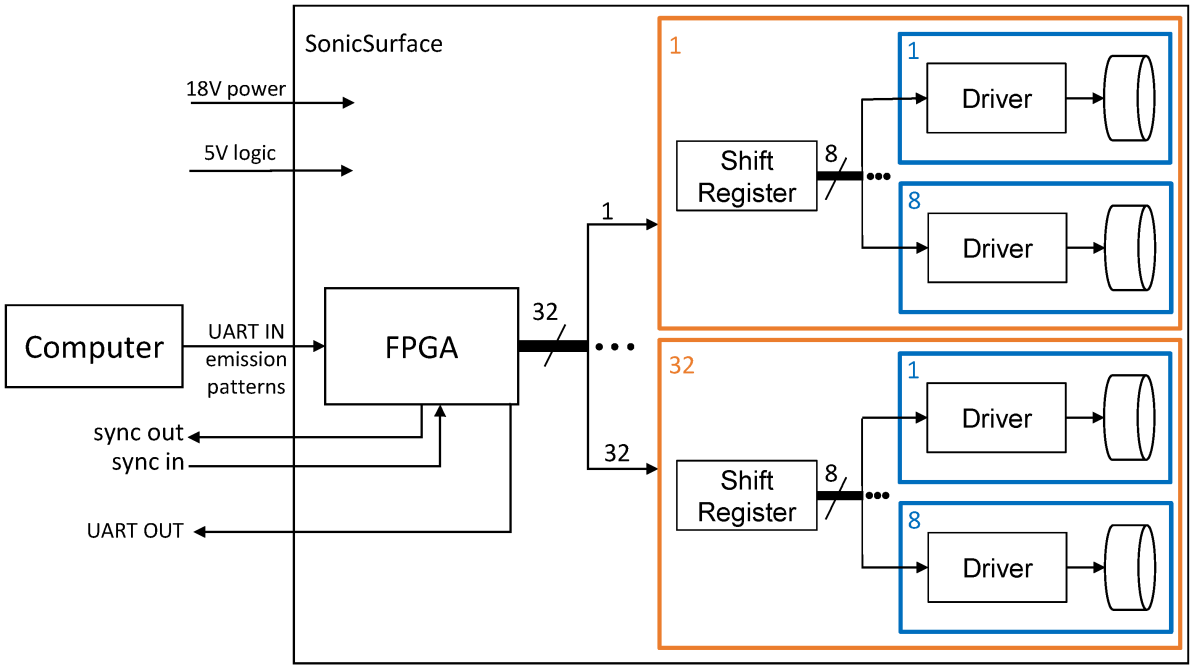


Figure 7: The Rev. 0 (SonicSurface) system level diagram. Source: [11].

Architecture

We considered solving for phases in hardware. Hardware solving would be fast and not bottlenecked by the communication between the host laptop and the board. However, only a simple set of commands or routines can be implemented with high hardware design complexity. Additionally, decoupling the software solver from the hardware design allowed for parallel development, greatly reducing development time.

Therefore, for Rev. 1, we decided that receiving and parsing phase data and generating the square waves for each transducer channel could be delegated to hardware.

We selected a dual-FPGA architecture for the high I/O and phase resolution requirements. Each FPGA is paired with a configuration device. The configuration device is a programmable persistent memory device that configures the FPGAs on power-up. We selected Cyclone 10 LP series FPGAs for familiarity with the workflow (Modelsim, Quartus), low cost, and because we already had development boards for the Cyclone FPGA family (eg. the *DE1-SoC*).

We selected the SystemVerilog language for FPGA logic. We chose SystemVerilog for its simplicity, readability, and powerful macros for test-benching. The module hierarchy was flat for plug-and-play of new submodules, and ease of independent test-benching of each submodule.

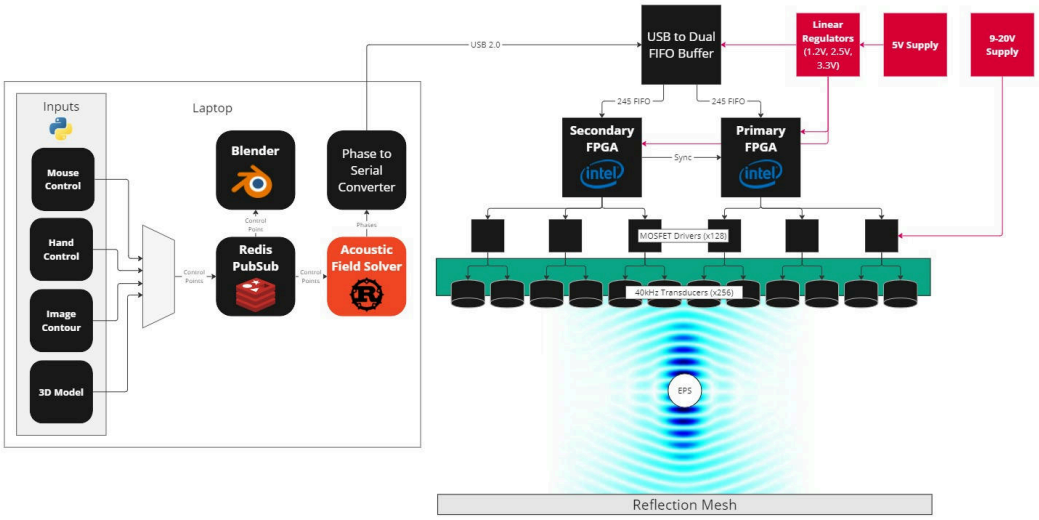


Figure 8: The complete system level diagram of the Rev. 1 software and hardware stack.

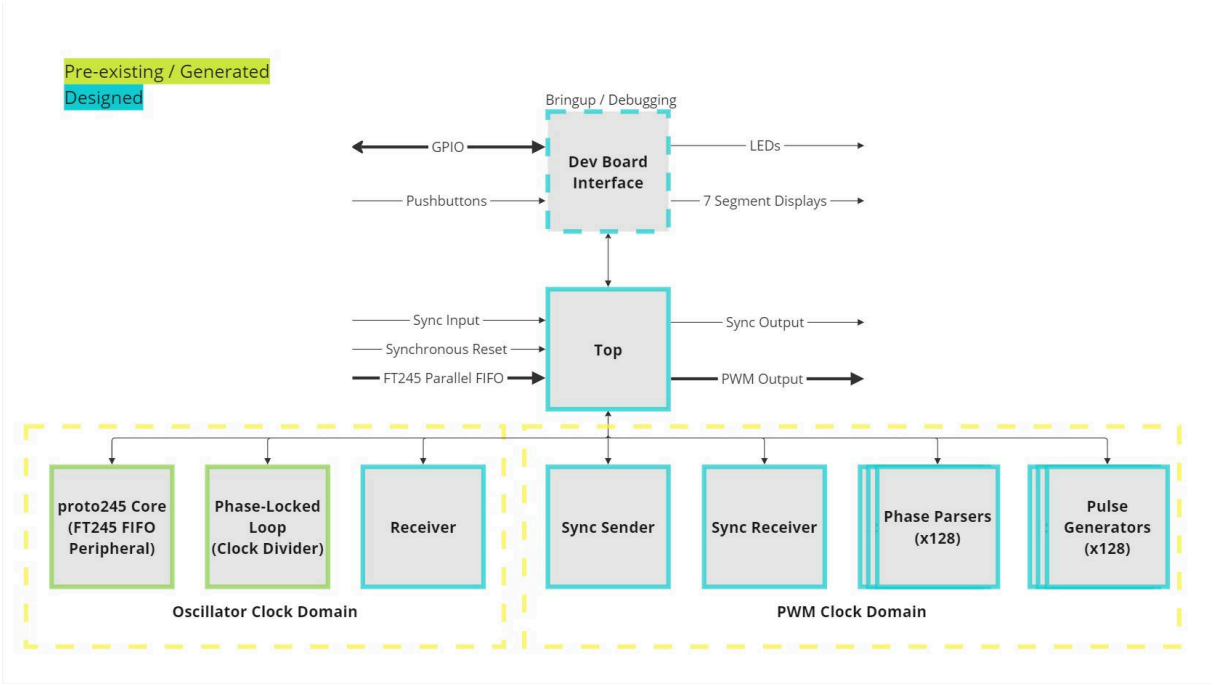


Figure 9: The hierarchy of the FPGA hardware description. Each block represents a submodule specified in its own SystemVerilog (.sv) file. Each submodule can communicate with one another through the top layer. Note the two clock domains: the oscillator clock domain runs at 24.576MHz, while the PWM clock domain runs at 10.24MHz.

Synchronization

Despite the stability of the oscillators selected for the FPGA, two identical oscillators can still drift out of phase from each other due to manufacturing differences[3]. Since the relative phase of the transducers needs to be precise, a synchronization scheme was necessary.

We used a 40 kHz square wave signal to synchronize the two FPGAs for convenience and because the sync signal frequency must be significantly smaller than the clock speed to prevent data loss. An incoming sync signal is first sanitized with a two-flop synchronizer. This is

[3]Indeed, tests on our board revealed a clock drift on the 40 kHz signal of a whole period every few seconds.

a common technique to reduce the probability of metastability. Briefly, metastability describes the brief instability of a register's value between 0 and 1 when the register input changes simultaneously with a clock edge. Metastability occurs commonly when the source and destination clocks differ.

Resource Usage

Table 1 shows the hardware resource usage in the final hardware design of the FPGAs.

Resource	Used	Total	Percentage
Total logic elements	5024	10320	49%
Total pins	147	177	83%
Total memory bits	32768	423936	8%
Total PLLs	1	2	50%

Communication

Communication Protocol

For the communication protocol between the computer and FPGA, we immediately ruled out I2C and RS232 for insufficient bandwidth. Between the FT245 parallel digital protocol and SPI, we selected FT245 for its 8MB/s bandwidth, an open-source peripheral module, and a USB driver wrapper written in Rust. We selected part number *FT2232H* for the dual-channel communication and purchased a Mini Module development board for bring-up. Initial data write tests resulted in actual write speeds of over 6.4MB/s.

Data Encoding

Several considerations went into encoding the data sent from the host to the board. The encoding needed minimal overhead to be efficient with the limited bandwidth but still had to be flexible enough for experimentation. The encoding also needed to support auxiliary commands, such as calibration and modulation (for audio). The final encoding is shown in Table 2 and Table 3, consisting of a command packet of 8 bytes, with an optional payload with a specified length in the command data. The payload contained phase data enumerated by the channel number. Enumeration allowed for configurable transducer grid sizes or selective phase updates. Error detection and correction were not implemented because our use case was tolerant of errors.

Byte Meaning	Suffix	Data				Code		Prefix
Hex Value	55	(D0)	(D1)	(D2)	(D3)	(C0)	(C1)	AA

Table 2: The big-endian message structure per byte of the Rev. 1 communication encoding. Each cell in the second row represents a byte with the hexadecimal value (eg. 55) or name (eg. (D0)).

Command	Code [C1, C0]		Data [D3, D2, D1, D0]			
Write Phase	00	01	XX	X(E) (E)nable [0-1]	(AA) (A)ddress [00-7F]	(PP) (P)hase [00-FF]
Burst Mode	00	02	(NNNNNNNN) (N)umber of Bytes in Payload [00000000-FFFFFFFF]			
Calibrate	00	03	XX	XX	XX	XX
Modulate	00	04	XX	X(HHHH)(E) (H)alf Period, (E)nable [0000-FFFF], [0-1]		

Table 3: The command code and data fields corresponding to each command for the Rev. 1 board. All values are in hexadecimal. X = Unused, (D) = (D)ata that corresponds with the specific byte, [range] = Range of values accepted.

Communication Speed Optimization

We performed a variety of tests to benchmark the communication speed. Speed is measured in frames per second, where a frame is the data to update the phase for every transducer on the board.

To summarize our key findings:

- Writing one large buffer was significantly faster than writing many small buffers. Writing a single buffer with multiple frames approached the max bandwidth speed of 8MB/s. However, as of writing, the hardware cannot display these frames at that rate.
- There were minor improvements if the host wrote simultaneously to each FPGA using multithreading. However, we did not properly synchronize the threads in these tests.
- We achieved our target 10kHz update rate on the hardware side, albeit with some unrealistic optimizations and idealizations, as shown in Table 4.

Single-Phase Writes[4]	Frame Write, Burst Mode[5]	Frame Write, Burst Mode, Parallelized[6]	Bulk Write, Burst Mode[7]	Bulk Write, Burst Mode, Parallelized	Bulk Write, One FPGA Only
73 Hz	4.3 kHz	5.2 kHz	6.1 kHz	8.6 kHz	**12.8 kHz**

Table 4: Phase write test results with different levels of optimization, averaged over three runs. Each test run involves writing 256 frames, with each frame containing the phase data to update every transducer channel on the board.

PCB Design

To enable the 10kHz update rate, the key change between Rev. 0 and Rev. 1 was the removal of the multiplexers. This drove the choice of FPGA and heavily influenced the physical PCB stackup. The other major change comes from the use of the USB to 245-FIFO buffer rather than

[4] Single-phase writes refers to using the “Write Phase” command.
[5] Burst Mode refers to writing phase data in a payload with the “Burst Mode” command.
[6] Parallelization refers to simultaneously writing to each FPGA with multithreading.
[7] Bulk Write refers to writing multiple frames as one large buffer, rather than a buffer for each frame.

USB to UART chip on the SonicSurface. The method of driving the transducers stayed the same, where we used a MOSFET to drive the transducer with a 9-20V square wave based on low voltage I/O signals from the FPGA.

Component Selection

The following key components required selection

- Transducers
- Transducer Driver
- FPGA
- FPGA Crystal
- FPGA Flash Memory
- Power Supplies

Transducers

A high volume of transducers was required, which heavily constrained the design space. The requirements were to emit at 40kHz and to have a diameter of 10mm to minimize the footprint. The choice of 40kHz is driven by existing work, as well as the fact that 40kHz is the most commonly available frequency used for ultrasound, both reducing technical risk. The directionality and sound pressure levels also vary between models, but of the options available at reasonable prices only the *TCT40-10T/R* transducers met the requirements for frequency and size, as well as shipping in a reasonable amount of time.

Drivers

To select a transducer driver, options were evaluated against the *MIC4127* used in Rev. 0. Due to the relatively low frequency of ultrasound, almost all options can easily meet the electrical rise and fall times needed. The main requirements were then the support for the output and input voltages, where the output was required to have 9-20V to match that of the transducers, and the input was required to work with typical 3.3V logic from the FPGA I/O pins. Other factors considered were cost and footprint size. Ultimately no options showed any performance benefits over the *MIC4127*. Additionally, the *MIC4127* was already verified by the Rev. 0 board to meet the requirements, mitigating technical risk.

FPGA and Peripherals

For FPGA choice, we first limited the design space to the Intel family of FPGA due to familiarity with the toolchains and logic design process. Most specifications such as clock speeds, memory size, DSP functionality, and power consumption were not compared in detail due to the simple processing needed; almost any FPGA could support what was needed. The key requirement for FPGA selection was instead the I/O count. The design space was further limited to the Intel Cyclone 10LP family due to their lower costs and simplified integration needs. Ultimately we selected the *10CL010YU256I7G*. The 10CL010 has 176 user-configurable I/O pins in a 256-LGBA package, which meets the 145 general I/O required for our hardware design, including transducers, communication, and synchronization, while still allowing for future expansion to other peripherals.

In addition to the FPGA itself, we selected an *EPCQ16ASI8N* flash memory chip based on the calculated project size from the prototype work done in Quartus.

Although the selected FPGA has internal clocks, they are prone to clock jitter, drift, and temperature instability. Instead, we selected an external oscillator. The upper bound on frequency was constrained by the hardware design to meet the register setup and hold times. The Quartus compilation report provided a worst-case maximum frequency for the design of 111.42 MHz. The lower bound was constrained by both the communication rate and the phase resolution. Estimating two clock cycles per byte parsed at 8 MB/s gives a required 16 MHz. To achieve the desired 256 divisions per period phase resolution, the minimum frequency was

10.24 MHz[8]. We selected a 24.576 MHz oscillator with 50 part-per-million frequency drift. This frequency can be divided easily to 10.24MHz, with a ratio of 12/5; the smaller the numbers in the fraction, the higher the stability of the phase-locked loop that generates the 10.24 kHz frequency.

Power Supplies

The voltages required for the PCB were 1.2V, 2.5V, and 3.3V for the FPGA Core, PLL, and I/O supplies, as well as 3.3V for the FTDI and a 9-20V supply for the transducers. Additionally, the FPGA required the voltage supplies be low noise, which largely constrained the choice of regulator to linear regulators to avoid switching noise. To permit the use of common linear regulators, we used two separate input supplies, one variable supply for driving the transducers and one 5V supply for stepping down to the lower voltages. Additionally, many FPGA families require a specific power-on sequence for the different supplies, but the Cyclone 10LP family is designed to handle this internally instead, greatly simplifying the power section. As power efficiency was not a requirement, the main requirements were the current and startup times. We calculated the current required based on estimates generated by Quartus, from the *FT2232H* datasheet, as well as a leakage current from charging the MOSFET gates in the *MIC4217*. To meet these requirements, and the target voltages, we selected an *AP7366* adjustable linear regulator capable of providing 600mA at all three input voltages.

The key components of the Rev. 1 board are summarized in Table 5.

Component	Part Number	Quantity	Unit Cost (CAD)
Transducer	TCT40-10T	256	0.27
Driver	MIC4127	128	0.87
FPGA	10CL010YU256I7G	2	16.51
FPGA Oscillator	ABM8G-12.000MHZ-4Y-T3	2	0.35
FPGA Configuration Device	EPCQ16ASI8N	2	10.38
PC-FPGA Communication	FT2232H	1	5.30
Voltage Regulators	AP7366	3	0.40

Table 5: A table summarizing the components used in the Rev. 1 PCB. Passive components or components pre-determined by the choices covered are not included for brevity. Due to sourcing from AliExpress, the cost of the transducers fluctuates, but remains close to the specified amount.

Passive Component Selection

For both the drivers and the FPGA power supplies, we ensured the proper filtering of signals. Voltage ringing was our primary concern for the drivers due to the parasitic impedances introduced by the long traces connecting them to the FPGA. The ringing could cause unexpected behaviour in the driving of the transducers, and in the worst case, permanently damage the drivers. The I/O of the FPGA was also a concern, but the Cyclone 10LP series FPGAs have built-in zener diodes for protection. We used an experimental setup with the *DE1-SoC* development board and the *MIC4127* drivers to assess the ringing that could be generated, what termination methods would be effective for mitigating ringing, and what range of values we could expect to work. The results can be seen in Figure 10. Ultimately, we only needed 20Ω resistors added in series at the driver-end to negate ringing.

[8] 10.24 MHz = 256 * 40kHz transducer frequency

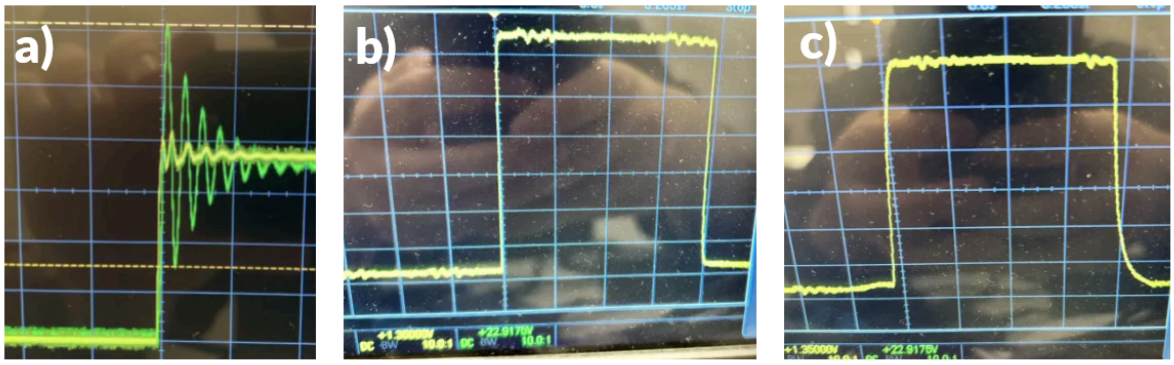


Figure 10: A: Oscilloscope traces from a test setup for ringing. The green signal was seen at the driver, while the yellow part was seen at the driver. This setup included a bundle of wire that acted as an extreme case of parasitic impedance. B: We found that adding a 20 Ohm resistor in series near the driver and adding a 1nF filter capacitor prevented ringing, which we then attempted on the Rev. 1 board. B: The figure shows the waveforms of the Rev. 1 board populated with only the 20 Ohm series resistor, showing well-formed signals with no over or undershoot. C: The same setup but with an additional 1nF filtering capacitor for each input, which increases the slew rate and is undesirable.

The power supplies of the FPGA chips required bulk and filtering capacitors to achieve the ripple voltages specified by the datasheets. These calculations were performed using Intel's PDN tool, which took in current draw and board geometries to produce plots of impedance as a function of frequency. Different capacitors were added until the calculated impedance fell below the target impedance for all frequencies of interest.

Other passives were all selected based on component datasheets for the desired configurations.

Layout

The high number and density of traces needed to connect all the transducers posed a challenge. We configured the board to have ten layers, five for signals and five for ground and power planes to simplify routing and improve EMI integrity. Additional ground pours were also added to the top and bottom.

Power supplies, drivers, transducer electronics, the *EPCQ16A*, and *FT2232H* were placed and routed first, leaving the 256 connections between the FPGA and transducers for last.

To minimize repetitive work, Altium Rooms were used extensively for the application of rules as well as automated tiling of the transducer and driver positions. Once the transducers and drivers were placed, the inputs were broken out to the right side of the board roughly around the two FPGAs. By using one signal layer for four inputs per row, no vias were needed in the initial breakout.

To make the connections to the FPGA, we first performed a fanout to break out all the connections, through vias, to the exterior of the LBGA footprint of the FPGA. The signals from the transducers were then brought around the FPGA, and we used a combination of manual and automatic pin swapping to optimize the final layout of the pins. The fanout and connections are shown in Figure 11. We exported the optimized pin configuration and used a custom Python script to perform assignment in Quartus.

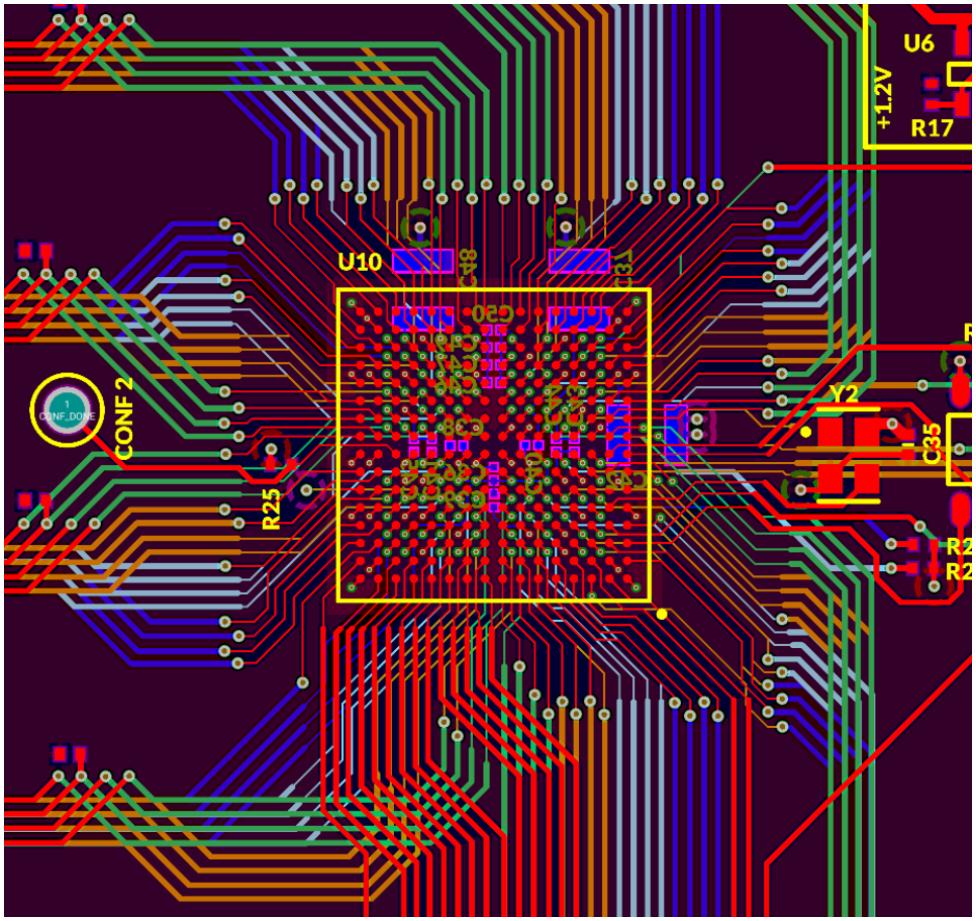


Figure 11: A section of the PCB layout showing the fanout and connection to the LBGA footprint of the FPGA. Microvias were sized according to JLCPCB design tolerances and placed between the pads, allowing for signals to be broken out to different layers.

Additionally, extra spots for voltage filtering capacitors were added to the drivers and power supplies in order to prevent the possibility of rework being required if the noise or ringing was greater than expected from simulation or experimentation.

Calibration

Due to variations in the impedance and capacitance of each signal trace that leads to the transducer drivers and physical variability in each transducer, we saw a phase offset in each transducer when supposedly driven by identically timed square waves. We performed calibration of the transducers by using an oscilloscope with probes attached to another transducer placed flush against the transducer being calibrated. Setting the rising edge of the driving square wave as a reference point, we used a phase offset to move the peak of all transducers such that the peak matched the rising edge of the square wave. We sent this calibration data to the FPGA before any session, and any phase data sent to the FPGA was added to these calibrated phases so that the relative offset of peaks would be correct.

Results and Capabilities

The speed of particles that was possible in Rev. 1 was much faster than possible in the SonicSurface. Due to time constraints, we could precisely calculate the speed of particles, but from analyzing video footage of both boards, Rev. 1 seems to be more than twice as fast at moving particles. However, this result only came after tuning the board's height and its mount's rigidity. The SonicSurface could operate fine at different heights and did not seem as sensitive to mounting requirements. Additionally, the speeds we saw did not yet reach those created in (Hirayama, 2019) [8], where the persistence of vision would turn the styrofoam beads into a blur. It is unclear where the main bottlenecks in the system are located, and further testing is needed.

We were able to increase the perceptibility of haptic feedback on hands, although this is mostly due to the increase in the quantity of transducers.

Lastly, since we were able to have full control of the FPGA logic, we were able to create modulation and accept frequencies as input, allowing us to play recognizable music via MIDI files.

Conclusions and Recommendations

- The Rev. 1. board we designed outperformed the SonicSurface in almost all ways but did not approach the state-of-the-art in terms of capabilities. Multiple factors can explain this, but further work is required to fully understand all the variables.
- Due to constraints in time, we used a simple reflecting surface, but introducing a second phased array is expected to drastically improve stability of particles, and should be one of the main things to work towards.
- Many improvements can also be made from the electrical design side, with changes being driven by further exploration of configurations in the solver software. Starting with transducers, a more reliable model with very little variation between components is desirable, otherwise, additional calibration steps like amplitude calibration must be added. Some transducers also performed anomalously, seeming to have higher sensitivity to external noise. This has unclear consequences and requires further analysis.
- Further effort should also be directed at the sound pressure level, directionality, and size of the transducers. A less simplified model of the transducers in the software solver may be a good starting point for this exploration. Another change is to remove the small gap between the transducers and instead packing them as closely as possible in a grid, or possibly even with optimal circle packing.
- We found the mechanical design to be more important than initially expected and demonstrated in the Rev. 0 SonicSurface. The current explanation is that due to the lower resolution achieved by the 10×10 array, some mechanical problems seen in the Rev. 1 board were hidden in Rev. 0. Specifically, we observed that the levitated particles were extremely sensitive to the distance of the reflecting surface as well as the alignment between the board and the surface. After swapping the board to a more rigid and well-aligned mount, the stability of particles improved greatly, and abnormalities like particles jumping upward out of traps disappeared.
- There were several unresolved problems with the data encoding protocol. The FPGAs would become unresponsive when the suffix and prefix bytes are received in order when unexpected. The enumeration of transducer channels was not useful and introduced significant overhead. A future optimization is to streamline the communication protocol to maximize the available bandwidth.
- The FT245 asynchronous communication protocol had too small of a bandwidth margin to account for overhead to achieve our desired frame rate comfortably. Even to achieve our desired rate, we made impractical optimizations. The *FT2232H* IC offered two communication modes, the faster being a 40 MB/s synchronous mode. However, this mode can only be used with a single channel, so we resorted to the 8MB/s asynchronous mode. We recommend individual 245-FIFO protocol ICs for each FPGA in synchronous mode. Alternatively, different communication protocols such as JTAG or RS485 with higher bandwidths can be explored.
- One idea that we haven't explored is delegating more tasks to hardware. We observed that writing one large buffer to the FPGAs was much faster than writing many small buffers. To handle larger buffer writes, we could implement a framebuffer[9] and delegate frame timing to the FPGAs. We could also try sending more phase data to the FPGAs at once from software, if the hardware permitted.

- We spent significant time debugging the FT2232H and associated peripheral devices. Implementation and bring-up of future revisions of the board can be sped up by reducing the complexity of the custom PCB and integrating development boards.

Other exploratory ideas include:
- Using CUDA to accelerate the solver
- Using a trained neural net instead of a solver
- Implementing LED control synchronized to the movement of traps
- Using a 3D camera to control particles using hands

Recommendations

- Get a genuine USB Blaster for the FPGAs (Hardware-based CPLD+FTDI programmer, NOT a software-based STM32 emulator)
- Use a computer with a strong CPU, such as the M-series Macbooks
- Calculate on the FPGA instead of in Software where possible (e.g.modulating rapidly for audible sound in Rust did not work)
- If haptic feedback is the focus, the number of transducers is important
- Vertical motion was tricky for both boards, as was moving multiple points at once: this was a result of the shapes of traps as seen in Figure 3, and could be remedied with a different solving algorithm
- The Rev. 1 board seemed to have small shifts in phase over time. We understand some of the shift is due to external noise interfering with the transducers, but the effect was much more pronounced than in the SonicSurface. We recommend reviewing possible causes for this drift before blindly copying our design.
- Multiple traps did not seem to help with increasing the strength of a trap, but we did not do enough research on the optimal spacing and quantity of traps per bead.

Deliverables

- 1. Our Github repository with PCB files, solver + position software, SystemVerilog, documentation.
- 2. This report, which is the best resource for understanding our design decisions

[9] Framebuffers are queues that can store multiple frames, and the release of each frame can be controlled by hardware logic.

Bibliography

[1] H. Bruus, “Acoustofluidics 7: The acoustic radiation force on small particles,” *Lab Chip*, vol. 12, no. 6, pp. 1014–1021, 2012, doi: 10.1039/C2LC21068A.

[2] J. F. Pazos Ospina, V. Contreras, J. Estrada-Morales, D. Baresch, J. L. Ealo, and K. Volke-Sepúlveda, “Particle-Size Effect in Airborne Standing-Wave Acoustic Levitation: Trapping Particles at Pressure Antinodes,” *Phys. Rev. Appl.*, vol. 18, no. 3, p. 34026–34027, Sep. 2022, doi: 10.1103/PhysRevApplied.18.034026.

[3] L. Wortsman, “Stability of a Particle Levitated in an Acoustic Field.” 2016.

[4] C. SUN, W. NAI, and X. SUN, “Tactile sensitivity in ultrasonic haptics: Do different parts of hand and different rendering methods have an impact on perceptual threshold?,” *Virtual Reality & Intelligent Hardware*, vol. 1, no. 3, pp. 265–275, 2019, doi: https://doi.org/10.3724/SP.J.2096-5796.2019.0009.

[5] Z. Long, S. Ye, Z. Peng, Y. Yuan, and Z. Li, “Phase Optimization for Multipoint Haptic Feedback Based on Ultrasound Array,” *Sensors*, vol. 22, no. 6, 2022, doi: 10.3390/s22062394.

[6] “SonicSurface: Phased-array for Levitation, Mid-air Tactile Feedback and Target Directional Speakers — instructables.com.” 2021.

[7] R. Hirayama, G. Christopoulos, D. M. Plasencia, and S. Subramanian, “High-speed acoustic holography with arbitrary scattering objects,” *Science Advances*, vol. 8, no. 24, p. eabn7614, 2022, doi: 10.1126/sciadv.abn7614.

[8] R. Hirayama, D. Martinez Plasencia, N. Masuda, and S. Subramanian, “A volumetric display for visual, tactile and audio presentation using acoustic trapping,” *Nature*, vol. 575, no. 7782, pp. 320–323, Nov. 2019, doi: 10.1038/s41586-019-1739-5.

[9] A. Marzo and B. W. Drinkwater, “Holographic acoustic tweezers,” *Proceedings of the National Academy of Sciences*, vol. 116, no. 1, pp. 84–89, 2019, doi: 10.1073/pnas.1813047115.

[10] “Radiation from a Plane Circular Piston.” [Online]. Available: https://jontallen.ece.illinois.edu/uploads/473.F18/Lectures/Chapter_7b.pdf

[11] R. Morales, I. Ezcurdia, J. Irisarri, M. A. B. Andrade, and A. Marzo, “Generating Airborne Ultrasonic Amplitude Patterns Using an Open Hardware Phased Array,” *Applied Sciences*, vol. 11, no. 7, 2021, doi: 10.3390/app11072981.