Phased Array in Sound

Austin Brummett, Electrical Engineering

Project Advisor: Dr. Dick Blandford

Date:

Evansville, Indiana

Acknowledgements

I would like to thank Dr. Blandford for proposing this project, helping with the conceptual work, and making sure I did not fail. I would also like to thank Jeff Cron for help with 3D printing the enclosure for the project.

Table of Contents

I. Introduction

II. Project Definition

  A. Client Specifications

III. Design Approach

  A. Hardware Design

  B. Software Design

  C. Constraints, Safety, & Standards

  D. Testing

  E. Costs

IV. Results and Conclusion

V. References

VI. Appendices

  A. Pseudo Code

  B. Software Code

  C. Hardware Schematic

  D. PCB Layout

List of Figures

Figure 1: Hardware Block Diagram

Figure 2: TLC7254CN

Figure 3: LM358

Figure 4: STM32F446RET-Nucleo ARM Board

Figure 5: Software Flow Chart

Figure 6: +56˚ phase shift between Speaker 1 and Speaker 5

Figure 7: -56˚ phase shift between Speaker 1 and Speaker 5

Figure 8: Final Product Full View

Figure 9: Internal Wiring

List of Tables

Table 1: Cost of Project

I. Introduction

Phased array (PA) systems are part of many different fields such as long-range acoustic modern transmission, sonar imaging, and ultrasound. More specifically, phased array sound systems are used in the medical field for ultrasound technologies from industrial nondestructive testing to noninvasively examine a heart. They are also used in nondestructive testing for finding flaws in welds, corrosion detection, and measuring the thickness of pipes. Another method of doing nondestructive testing is using Radiographic Testing (RT), but it is much more expensive and Phased Array Ultrasonic Testing (PAUT) is also more accurate, easier to set up, can create a 3D image, and is portable. These things make it a superior option. PA's are also conceptually used more because they can be made portable and can be modified to work at multiple points in the frequency spectrum.

This project is based around five independently-controllable speakers to create a phased array speaker system that generates flexible, directional sound. The idea behind the phased array is that by changing the how the speakers are driven the angle of the maximum intensity of the output wave is shifted.

II. Problem Definition

The biggest issue with use of phased arrays is that they are expensive, generally in the thousands of dollars range. A phased array in sound provides the issue of creating a sample of the phased array system that can be used as a tool to demonstrate this concept. To create this system a true time delay needs to be between the elements which means a microcontroller is well-suited to the task. The goal then is to create an affordable line array of speakers that can demonstrate the concepts of a phased array system. The final system should have a way for the user to tune the direction via a potentiometer or similar instrumentation. The output signal will need to be put through at least a simple low pass filter to get rid of quantization error and then to an audio amplifier.

A. *Client Specifications*

- Tunable phase shift based on a potentiometer
- At least 5 speakers separated in a line array
- 8-bit ADC and DAC input and output
- Low pass filter to lower quantization error
- Amplifier to output speakers

III. Solutions

A. *Hardware*

Figure 1 shows a block diagram of the proposed design. The hardware for this project is minimal, but each part was carefully selected to be able to achieve the client specifications.

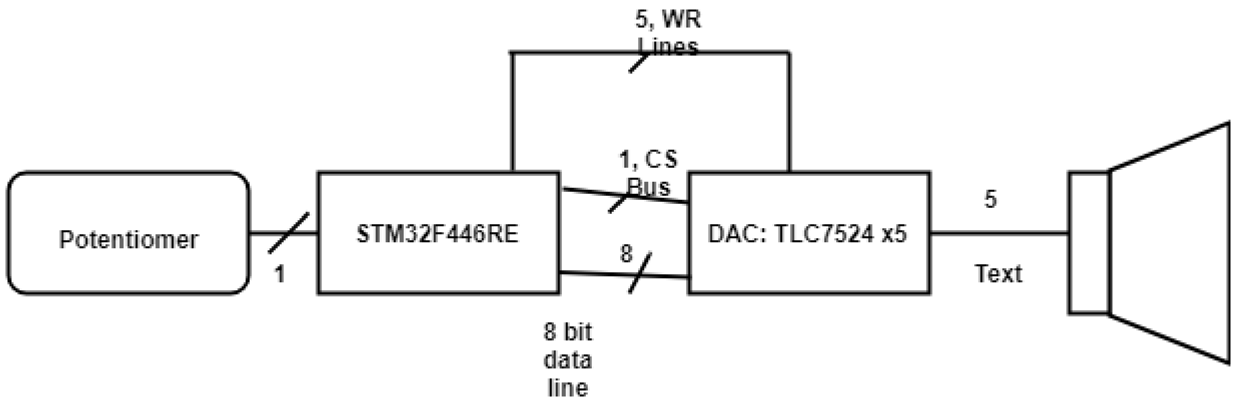


Figure 1: Hardware Block Diagram

The first piece of hardware is the digital-to-analog convertor (DAC). For this design, the DAC needed to be able to update multiple speakers simultaneously and output at a resolution of at least 8 bits. The TLC7524 DAC [1], pictured in Figure 2, was selected. This specific DAC has a Write and Chip Select pin and only one channel which led me to use five of them. The output of the TLC7524 is current unbuffered, so an LM358 Dual Operational Amplifier [2], pictured in Figure 3, was connected to get a voltage output. Connected to the output of the LM358 was a simple RC Lowpass Filter centered at 22.05 kHz, which was half the sampling frequency. This DAC was a second choice due to an oversight in the original design. This will be addressed in the *Testing* section.

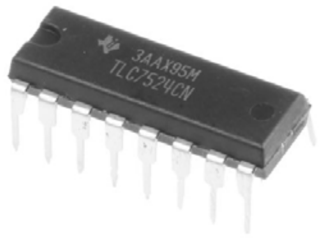


Figure 2: TLC7524CN DAC

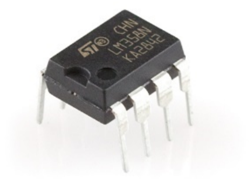


Figure 3: LM358 Dual Op Amp

The project is designed on the Cortex M4 platform, which is designed with digital signal processing (DSP) capabilities. The STM32F446 board, pictured in Figure 4, was chosen out of familiarity, but the project could be implemented on a smaller M4 microcontroller if necessary. The processor speed is fast enough for the needs of the project and there are plenty of onboard analog-to-digital converters (ADC) with multiple channels. The ADCs also have a programmable resolution and sampling clock. The Nucleo board also supports a phased lock loop (PLL) onboard that allows it to be clocked at 180 MHz, which is beneficial when working with audio signals and slow processes like the ADC.

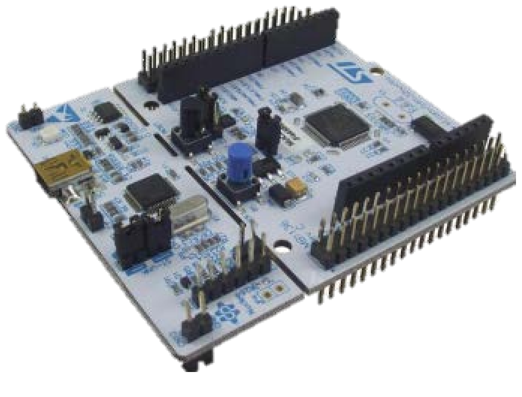
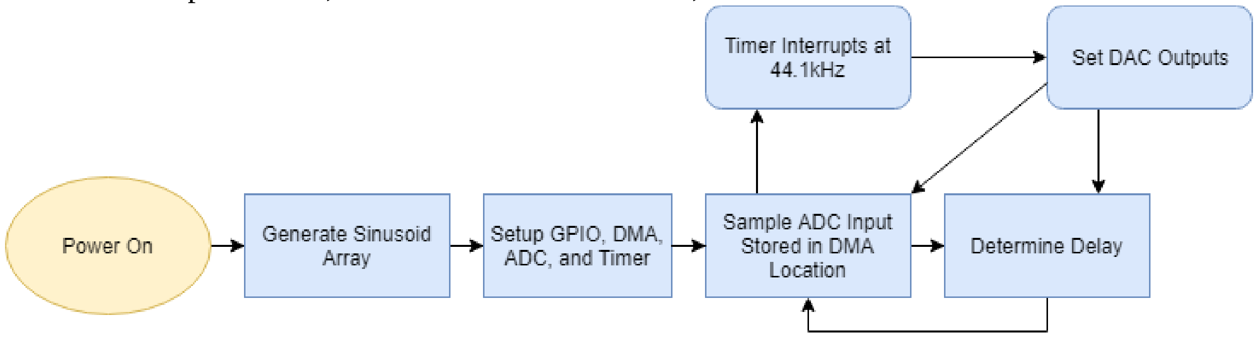


Figure 4: STM32F446RET-Nucleo ARM Board [3]

The potentiometer is connected to one of the ADC channels on the Nucleo board to determine the users desired phase shift. The adjustment is based off the potentiometers current position and inserting a time delay between each of the DAC outputs to cause a phase shift in the time domain. The Nucleo board is calculating the approximate time delay between each of the DAC channels. These are sent to the 8-bit Data Bus from GPIOC of the Nucleo Board to each of the DAC channels which are selected when needed by GPIOH0, a common chip select channel, and GPIOB5-9, as a write channel. The output of the DAC is sent through the op amp and is then sent through the passive lowpass filter. The DAC is setup in such a way that the voltage fed into the input amplifier built into the Sound Force 540 Speakers [4] to between 0-500mV so it does not damage them. A hardware schematic and PCB layout can be found in Appendices C and D, respectively.

B. *Software*

The software uses the microcontroller to grab the ADC value from the potentiometer to determine the phase shift, create a sine wave at 1 kHz, and send that to each of the external


Timer Interrupts at 44.1kHz
Set DAC Outputs
Power On
Generate Sinusoid Array
Setup GPIO, DMA, ADC, and Timer
Sample ADC Input Stored in DMA Location
Determine Delay

DACs. Figure 5 is a flow chart of the program.

Figure 5: Software Flow Chart

The complication of the software was how slow the default ADC sampling rate was because it took 15 clock cycles at 12-bits. To lower that, it was changed to 8-bit making it 11 clock cycles, but this was still causing blurbs in the output sound. To fix this, direct memory access (DMA) was used because it significantly lowered the time it took to read from the ADC1 data register, and the clock speed was increased to 180 MHz. This got rid of the blurbs that made the output sound like a helicopter.

The result from the ADC is stored in a one element array that was checked against a set of if-statements that correlated to a phase shift. Then when the timer interrupts the phase shift, which is a shift in the 360-point array generated at the start of operation, is checked to see if it is positive or negative and then the absolute value of the shift that is added from speaker zero to four if it is positive and four to zero if it is negative. For each speaker the time delay is multiplied by an increasing increment, i.e., with a shift of 55 speaker 0 is loaded at point[i+1*shift] and speaker 1 is loaded at point[i+2*shift], and so on down the array.

After the interrupt is finished the timer is restarted. After the timer is restarted a new ADC value is read in through DMA2 stream 0. Then a new shift is set and the process repeats. Pseudocode and the implemented code can be found in Appendices A and B, respectively.

C. *Constraints, Safety, & Standards*

For this project ISO/ICE/IEEE standard 24748-1-2018 [4] was considered. This standard relates to the life cycle of software. The project has software, so sustainability of the software is important. If this were to be manufactured for mass classroom use, the manufacturer would likely want to create a PCB that includes the microprocessor built into it and to create a more size efficient 3D printed enclosure because the print time on the one used for this project was inefficient.

In terms of health consideration, the project needed to be designed so that it did not cause any permanent hearing damage to anyone listening to it for extended periods of time. According to OSHA [5], that to prevent damage the sound should be at or under 60dB, and if at 85dB it should not be listened to for more than 8 hours a day. Keeping the project at or under 60dB allows it to be quiet enough to be talked over and not cause damage, it also prevents it from being a noise polluter in the environment it is being used in. This was achieved by using a noise spectrometer application to measure output of all five speakers to keep it at or under 60dB.

Other than sound levels, a danger is that there are long running wires in the enclosure to the audio jacks, so to make sure they don't become a fire hazard they should be taped down with electrical tape and bundled. This will also stop them from harming anyone who might encounter the wires while servicing the device.

D. Testing

Testing for this project had multiple stages. The first stage was the digital-to-analog convertor testing phase. Originally, the TLC5628 was used because it was an 8-bit 8 channel device that had a SPI interface. An alternate device was ordered by mistake, the TLV5628, which does not have a SPI interface, instead it had a microwire interface which was incompatible with the microcontroller that was being used. To remedy this, an 8-bit data bus was set up and five TLC7524 DAC were set up to generate the sine waves. After the DACs were interfaced the outputs were confirmed using an oscilloscope and then audibly tested using the connected speakers. Output confirmation is show in Figures 6 and 7, speaker 1 is channel 2 and speaker 5 is channel 1 in both figures. The delay came out to be around 350 μs which is about a 56˚ shift. This allowed me to more accurately spaces the speakers using the equation:

$$d_{max} = \frac{\lambda}{(1 + \sin\theta)}$$

which returned a maximum center to center spacing of approximately 10 inches. This equation is commonly used in phased array ultrasonics to prevent the issue of grating lobes.

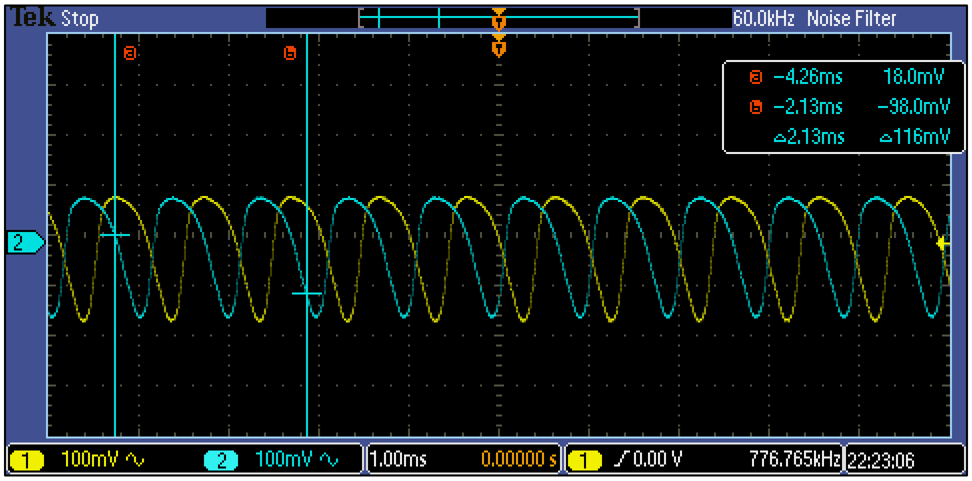


Figure 6: +56˚ phase shift between Speaker 1 and Speaker 5

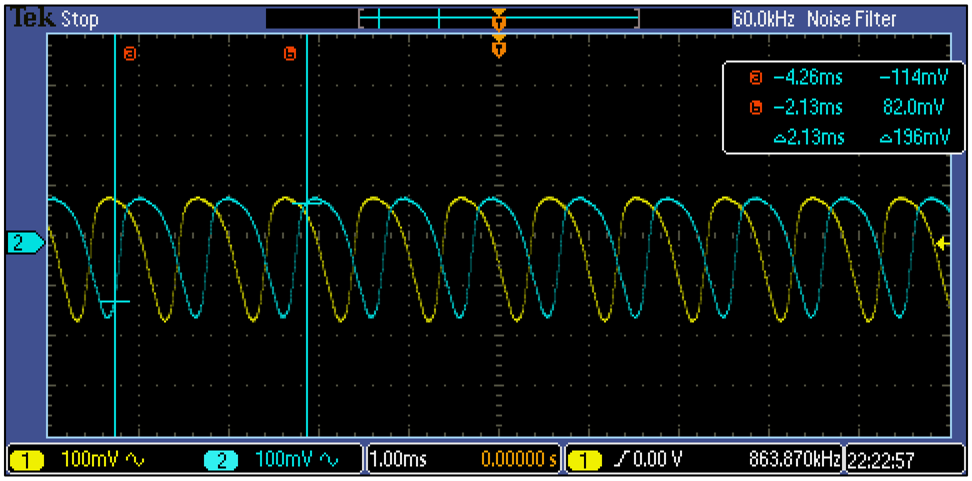


Figure 7: -56˚ phase shift between Speaker 1 and Speaker 5

E. Costs

Table 1 shows costs of this project. The total cost is $155.75, which is within the budget of $300 proposed to the Department of Engineering and Computer Science at the University of Evansville. If this project were built again it would cost $114.34 because the part list would be predefined.

Table 1: Cost of Project

Part Name	Manufacturer Part Number	Price/Unit	Quantity	Total Price
IC 8Bit 10us Octal DAC S/O 16-Dip	TLV5628IN	7.33	2	14.66
8 Ohm 3W Top Port Speaker	AS07108PO-3-R	4.25	6	25.5
450 Ω Resistors		1.26	10	12.6
.1μF Capacitors		0.87	5	4.35
10kΩ Potentiometer		7.7	1	7.7
PCB (Pack of 1)		55.37	1	55.37
CONN JACK MONO 3.5MM R/A	MJ-3536	1.25	1	1.25
Dual Operational Amplifier	LM358	0.84	5	4.2
Audio Jack 3.5mm	-----	1.25	5	6.25
STM32F446RET Nucleo Board		20	1	20
Soundforce 540(Box of 7 pairs)		58.97	1	58.97
8 bit single channel DAC	TLC7524	4.13	5	20.65
Total Sum	231.5			
Total Budget	300	Development Cost	155.75	
Leftover	144.25	Final Project Cost	173.24	

IV. Results and Conclusions

This project meets all the client's minimum requirements. There are some expansions that could be added to this project such as adding the ability to feed in an input signal to the microcontroller instead of generating the signal in the software. To further improve on the design and to lower the number of ports used on the microcontroller an 8-bit serial in parallel out shift could be used to send the information to the multiple DACs. For the purposes of the project, the current process is satisfactory. The final product is shown in Figure 8 and 9.

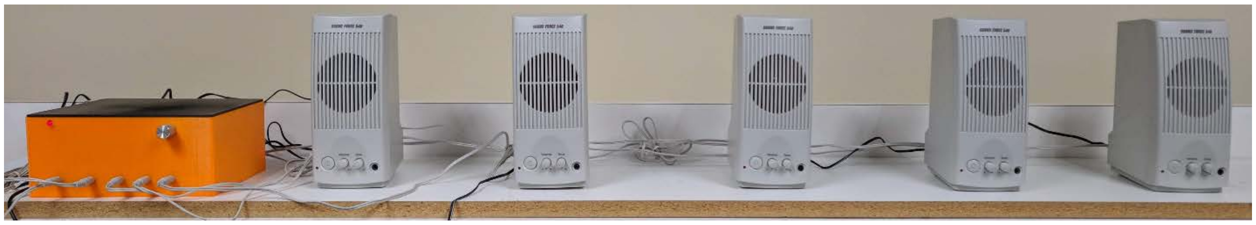


Figure 8: Final Product Full View

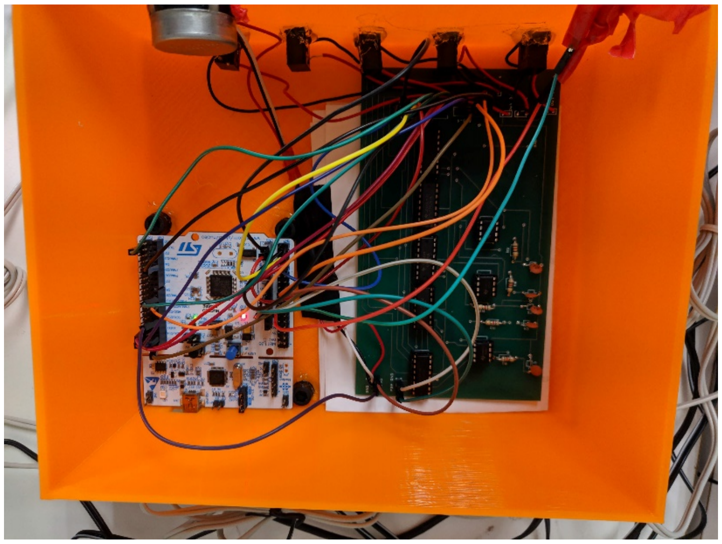


Figure 9: Internal Wiring

V. References

[1] Texas Instruments. “8-Bit Multiplying Digital-to-Analog Convertor”. [Online]. Available: http://www.ti.com/lit/ds/slas061d/slas061d.pdf
[2] Texas Instruments. “LM358 Low-Power, Dual-Operational Amplifiers”. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm158-n.pdf
[3] Amazon. “STM4446re Nucleo Board”. [Online]. Available: https://www.amazon.com/gp/product/B014IXUB1M/ref=oh_aui_detailpage_o09_s00?ie=UTF8 &psc=1
[4] “Sound Force 540 Powered Computer Speaker Pair”. [Online] Available: hhttps://www.parts-express.com/sound-force-540-powered-computer-speaker-pair--319-128 [5] ‘ISO/IEC/IEEE 24748-1:2018’ [Online] Available: https://www.iso.org/standard/72896.html
[6] ‘How Loud is Too Loud?' [Online] Available: https://www.osha.gov/SLTC/noisehearingconservation/loud.html

VI. Appendices

A. Pseudo Code

```
#include <stm32f4xx.h>
int main (){
        ▪ initialize gpio, timer, dma, adc1
              • set timer to interrupt 44.1kHz
              • adc1 to 8-bit resolution for potentiometer (0 to 255)
        ▪ while 1
              • get input from pot. and set "time delay"
        ▪ end
}
Void tim3_interrupt (){
        Set channel based on sign of time delay
        ▪ set DAC channel 1
        ▪ delay
        ▪ set DAC channel 2
        ▪ delay
        ▪ Repeat until Channel 5 set and loaded
}
```

B. Software Code

```
/* Austin Brummett
Senior Project: Phased Array of Sound
Project Advisor: Dr. Blandford
*/

// header files
#include "stm32f4xx.h"        // Device header
#include "stdlib.h"           // abs
#include "math.h"             // sin
#include "TimerDelay.h"       // Delay_ms
#include "limits.h"           // UINT_MAX
```

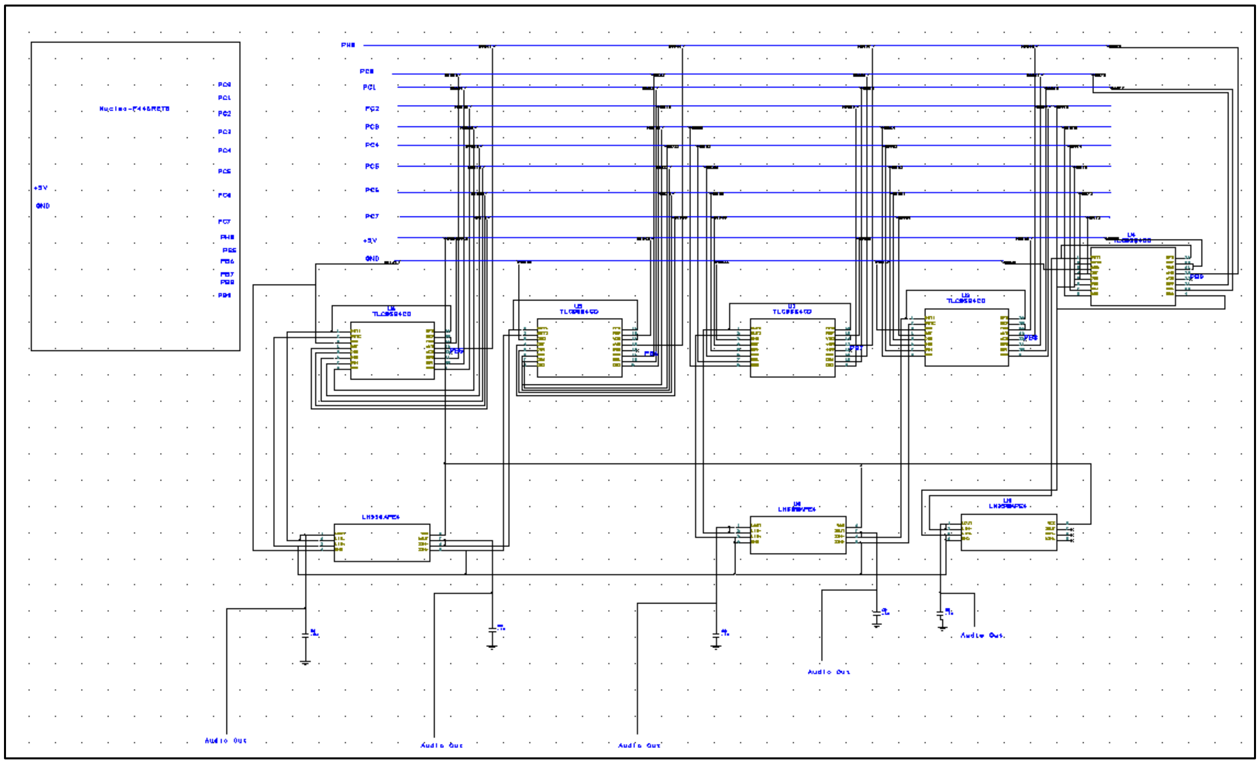
```
// Constants
#define PI 22/7   // substitute for M_PI because it wasn't available
#define MAXVOLUME 255.00 // 2^8-1 == 255

// Functions
void SetupMCU(int sampleRate); // Setup Peripherals

volatile unsigned int arr[44]; //array being accessed
// Global Variables
unsigned int i = 0; // index for arr[] in
volatile int td  = 0; // "time delay" to simulate the change in
volatile uint16_t tmp[1]; // temporary variable to grab adc data
const int sampleFreq = 44100; // sample frequency of sine wave




//********* Start Main Function
******************************************************
int main(void){
      /******** Pre-Setup *********/

      // Fill global array
      for(int j = 0; j < 44; j++){
            arr[j] = MAXVOLUME*((sin(2.0*PI*(double)j/44)+1.0)/2.0);
      }

      /******** Setup Microcontroller ***********/
      SetupMCU(sampleFreq); // Setup GPIO, Timer, ADC, DMA, and Clock
      NVIC_EnableIRQ(TIM3_IRQn); // Enable timer 3 interrupt
      GPIOA->ODR |= 1 << 7; // turn on LED to signal power on and setup done
      GPIOB->ODR = 0x3E0; // turn all DACs off
      /********* Start Everything *************/
      ADC1->CR2 |= ADC_CR2_DMA | ADC_CR2_DDS; // DMA and DDS
      TIM3->CR1 |= TIM_CR1_CEN; // Enable Timer 3
      while(1){
            ADC1->CR2 |= ADC_CR2_SWSTART; // bit 30 does software start of
A/D Conversion
            // constantly check for the new "time delay", the number of
cycles to skip
            // tmp is read in with DMA
            // td is between -7 and 7
            if(tmp[0] < 28) td = -7;
            else if(tmp[0] < 56) td = -5;
            else if(tmp[0] < 85) td = -3;
            else if(tmp[0] < 113) td = -1;
            else if(tmp[0] < 142) td = 0;
            else if(tmp[0] < 170) td = 1;
            else if(tmp[0] < 198) td = 3;
            else if(tmp[0] < 226) td = 5;
            else td = 7;


            }
}
//********* End Main Function
*********************************************************
```

```
// Name: Timer 3 Interrupt Handler
// Purpose: Loads the DACs at a frequency of 44100 Hz or 22.6us
//                             based on the current value of td
void TIM3_IRQHandler(){

      int td_t = abs(td); // absolute value so the array can be shifted
correctly
      int k[5]; // unnecessarily an array

            if(td > 0){
                  GPIOH->ODR &= ~(1<<0); // Set /cs low to set new inputs

                  k[0] = (i+0*td_t)%43; // no delay

                  GPIOB->ODR &= ~(1 << 5); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[0]]; // send data
                  GPIOB->ODR |= 1 << (5); // !WR Lock transfer

                  k[1] = (int)(i +1*td_t)%43;
                  GPIOB->ODR &= ~(1 << 6); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[1]]; // send data
                  GPIOB->ODR |= 1<< 6; // !WR Lock transfer

                  k[2] = (int)(i +2*td_t)%43;
                  GPIOB->ODR &= ~(1 << 7); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[2]]; // send data
                  GPIOB->ODR |= 1<< (7); // !WR Lock transfer

                  k[3] = (int)(i +3*td_t)%43;
                  GPIOB->ODR &= ~(1 << 8); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[3]]; // send data
                  GPIOB->ODR |= 1<< 8; // !WR Lock tran 8fer

                  k[4] = (int)(i +4*td_t)%43;
                  GPIOB->ODR &= ~(1 << 9); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[4]]; // send data
                  GPIOB->ODR |= 1 << 9; // !WR Lock transfer

                  GPIOH->ODR |= 1 << 0; // set /cs high to hold all

            }else if(td < 0){
                  GPIOH->ODR &= ~(1<<0); // Set /cs low to set new inputs

                  k[0] = (i+0*td_t)%43;
                  GPIOB->ODR &= ~(1 << 9); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[0]]; // send data
                  GPIOB->ODR |= 1 << 9; // !WR Lock transfer

                  k[1] = (int)(i + 1*td_t)%43;
                  GPIOB->ODR &= ~(1 << 8); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[1]]; // send data
                  GPIOB->ODR |= 1<< 8; // !WR Lock transfer

                  k[2] = (int)(i +2*td_t)%43;
                  GPIOB->ODR &= ~(1 << 7); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[2]]; // send data
                  GPIOB->ODR |= 1<< 7; // !WR Lock transfer

                  k[3] = (int)(i +3*td_t)%43;
                  GPIOB->ODR &= ~(1 << 6); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[3]]; // send data
                  GPIOB->ODR |= 1<< 6; // !WR Lock transfer

                  k[4] = (int)(i +4*td_t)%43;
                  GPIOB->ODR &= ~(1 << 5); // !WR -- Pick device  to WR
                  GPIOC->ODR = arr[k[4]]; // send data
                  GPIOB->ODR |= 1<< 5; // !WR Lock transfer
                  GPIOH->ODR |= 1 << 0; // set /cs high to hold all
            }
            else{ // if at 0 no delay all on
                  GPIOH->ODR &= ~(1<<0); // Set /cs low to set new inputs
                  k[0] = (i)%43;
                  GPIOB->ODR = 0; // All on
                  GPIOC->ODR = arr[k[0]]; // send data

                  k[1] = (i)%43;
                  GPIOC->ODR = arr[k[1]]; // send data

                  k[2] = (i)%43;
                  GPIOC->ODR = arr[k[2]] >> 1; // send data

                  k[3] = (i)%43;
                  GPIOC->ODR = arr[k[3]]; // send data

                  k[4] = (i)%43;

                  GPIOC->ODR = arr[k[4]]; // send data
                  GPIOB->ODR = 0x3E0;     // all off

                  GPIOH->ODR |= 1 << 0; // set /cs high to hold all
            }

            i++;
            if(i == UINT_MAX) // added limit.h, fixed the stutter in the
output
                  i = 0;
            // Restart Timer
            TIM3->SR &= 0xFFFE;
            TIM3->CR1 |= TIM_CR1_CEN;

}
// End TIM3_IRQHandler


// Name: SetupMCU
// Purpose: Sets up timer 3 to interrupt at f = 44100 Hz,
//                            GPIO ports for the D/A convertor,
//                            Sets System Clock to 180Mhz,
//                            Sets up ADC channel 1 to read from
potentiometer at 8 bits
void SetupMCU(int sampleRate){
//    // Reset clock to 180MHz
```

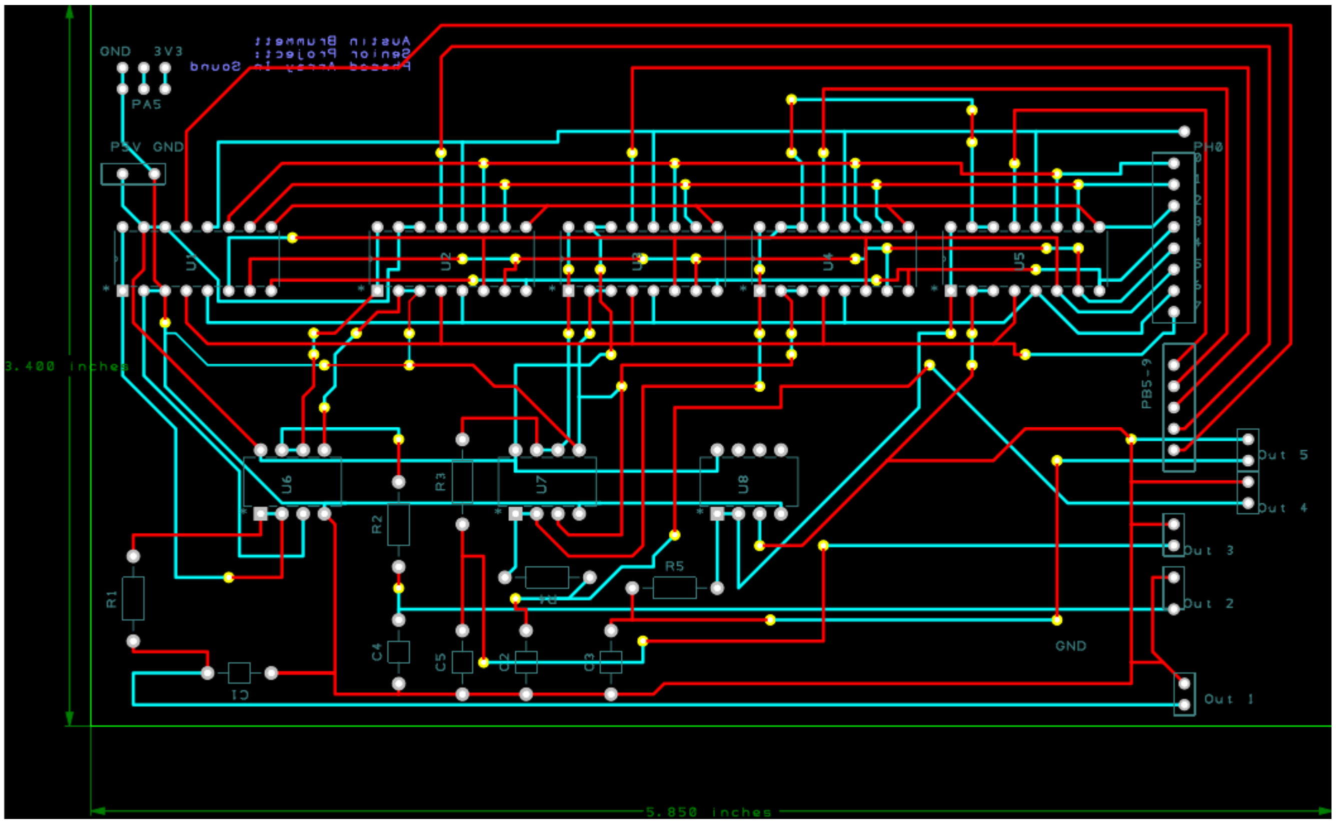
```
	RCC->CFGR = 0x00000000; // reset clock configuration reg.
	RCC->CR &= 0xFEF6FFFF; // reset hson, csson, pllon
	RCC->CR |= RCC_CR_HSEON; // turn on hse clock
	while((RCC->CR & RCC_CR_HSERDY) == 0); // wait untile hse is ready
	RCC->PLLCFGR = 0x27405A08; // set PLLP = 0, PLLN = 360, PLLM = 8,

	 // PLLQ = 7, PLL Src = HSE
	FLASH->ACR &= 0xFFFFFFF8; // set flash wait states to 5
	FLASH->ACR |= 0x5;
	RCC->CR |= RCC_CR_PLLON; // enable PLL on bit
	while((RCC->CR & RCC_CR_PLLRDY) == 0); // wait for PLL to lock on
	RCC->CFGR = 0x9402; //APB2/2, APB1/4, AHB/1
	// Clock bits
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // GPIOA
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN; // GPIOB
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN; // GPIOC
	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOHEN; // GPIOH
	RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // adc1 for pot
	RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; // timer3 for updating dacs
	RCC->AHB1ENR |= RCC_AHB1ENR_DMA2EN; // Enable DMA2
	// I/O Bits
	GPIOA->MODER |= 3<<5*2 | 1 << 7*2; // PA5 Analog -- ADC1, PA7 Dig. Out
	GPIOA->OSPEEDR |= 0xC000; // High speed PA7/ timer
	GPIOA->PUPDR &= 0xFFFFF0FF; // A4 & A5 npup npdown

	GPIOB->MODER |= (1<<5*2)+(1<<6*2)+(1<<7*2)+(1<<8*2)+(1<<9*2);// PB5-9,
!WR Bits
	GPIOB->OSPEEDR |= 0xFFC00; // High Speed PB5-9
	GPIOB->MODER |= 2<<4*2; //Timer 3 PB4 Alt Function
	GPIOB->AFR[0] |= 2 << 4*4; // timer 3

	GPIOC->MODER |= 0x5555; // PC0-7 Digital Output
	GPIOC->OSPEEDR |= 0xFFFF; // High Speed PC0-7

	GPIOH->MODER |= 1<<0*2; // PH0 !CS bus
	GPIOH->OSPEEDR |= 3 << 0*2; // high speed PH0


	// Timer Bits
	int clockDiv = ceil((90000000/sampleRate));
	TIM3->DIER |= TIM_DIER_UIE; // TIE, UIE, CCIE
	TIM3->CR1 |= TIM_CR1_ARPE; // auto reload buffered
	TIM3->PSC = 0;    // no prescaling
	TIM3->ARR = clockDiv;// (180MHZ/2)/2040 44.1kHz
	TIM3->EGR |=1; // Event Generation enabled


	// ADC Setup
	ADC1->CR2 |= ADC_CR2_ADON; // ADC on
	ADC1->CR1 |= 2 << 24;          // 8 bit
	ADC1->CR1 |= ADC_CR1_SCAN; // scan mode
	ADC1->CR2 |= ADC_CR2_CONT; // continuous mode
	ADC1->SQR3 &=0x0;       // clear out the SQR3
	ADC1->SQR3 |= 5;  // bits 4:0 are channel number for first conversion

	 // channel is set to 5 which corresponds to PA5
	ADC->CCR |= 3 << 8*2;   // fastest possible conversion time
```

```
        // DMA Setup
        DMA2_Stream0->CR &= ~DMA_SxCR_CHSEL_0; // select channel 0
        DMA2_Stream0->CR |= 3 << 17;// high priority
        DMA2_Stream0->CR |= 0 << 11; // 16  bits ADC data
        DMA2_Stream0->CR |= 0 << 13; // 16  bits memory data size
        DMA2_Stream0->CR |= 1 << 10; // memory increment
        DMA2_Stream0->CR |= 1 << 8; // circular mode
        DMA2_Stream0->NDTR = 1; // one adc conversion
        DMA2_Stream0->PAR = (uint32_t)&ADC1->DR; // set to ADC1 DR address
        DMA2_Stream0->M0AR = (uint32_t)&tmp[0]; // Store values in tmp[0]
        DMA2_Stream0->CR |= DMA_SxCR_EN; // DMA Enable
  }
  // End
```

C. *Hardware Schematic*



D. PCB Layout



Austin Brummett
Senior Project:
Phased Array in Sound
GND 3V3
PA5
P5V GND
PH0
PB5_9
U1
U2
U3
U4
U5
U6
U7
U8
R1
R2
R3
R5
C1
C2
C3
C4
C5
Out 5
Out 4
Out 3
Out 2
Out 1
GND
3.400 inches
5.850 inches