# Software

To describe briefly, the software creates an interface between the Analog to Digital Converter and the Digital to Analog Converters to sample and output audio at 44.1 kHz. This software is running on an Arduino Mega microcontroller dev board, as will later be seen in the hardware section.

The first bit of the Arduino code to run is the setup function. This function initializes the variables needed to communicate with the hardware. First, it sets the Data Direction Registers for each pin of each port used. The port the data bus of the ADC is connected to is set to input, while the data bus of the DACs, and both chips’ control pins are set to output. It also sets the values of the ports’ Data Registers to the desired default value. In the control logic of the chips some actions are triggered on the falling, some on the rising edge of the waveform. Therefore, it is important to have the right default value for each pin. These values were obtained from the datasheet of the chips. The setup function also initializes a timer-based interrupt, that handles all the calculations at regular intervals. This involves setting half a dozen registers to the right value, that specify how many clock cycles the software should wait between interrupts.

To achieve 44.1 kHz sampling rate, the interrupt fires every 22.56 nanoseconds. With the microcontroller’s clock speed of 16MHz, this equates to 361 clock cycles. Reading the analog audio input, applying the phase shift and outputting to 12 different DAC channels in such a low number of clock cycles proved to be a challenging task. This made it necessary to use C and manipulate the I/O ports of the microcontroller directly, instead of using the standard Arduino commands. The built-in functions for pin manipulation include functions for checking for errors and other dependencies, causing them to take over 50 clock cycles to execute. Direct port manipulation and bitwise arithmetic helped reduce this computational overhead, most significantly in setting the DAC outputs. For easy implementation, two macros were defined at the beginning of the code, one to set a pin’s value to digital 1, the other to clear it, resetting to digital 0.

Both the ADC and the DACs were communicating with the microcontroller using a parallel data bus. Since we had 3 DACs, but only one of them was having its output buffer updated at a given time, they were using a common data bus and common output selectors, and only the control pins needed to be written to separately. More on the connection in the hardware section. [include reference] The code sequence executed in each interrupt is the following. First, the microcontroller sends a signal for the ADC to measure the analog input and convert it to a digital value. Then it waits 5 microseconds to give ample time to the ADC to carry out the conversion. The microcontroller then reads the data from the ADC’s output pins over an 8-bit wide data bus, directly linked to one of its 8-bit wide I/O ports. Then the read input gets stored in an array that holds the 2048 most recent input values. The index used to mark the position in the array uses modular arithmetic to “loop around”, always overwriting the oldest stored value. This array stores 2048 snapshots of the audio signal, each taken 22.56 nanoseconds after the previous one. Phase advancing is implemented by outputting an audio value that was recorded at an earlier/later time than the non-advanced value. The centre, non-adjusted value was chosen to be in the middle of the array, i.e. the 1024th most recent entry, such that phase advancing could be used in both forwards and backwards in time.

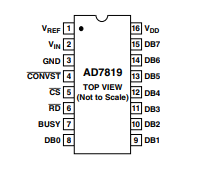
Having identical time difference between consecutive speakers produces a beam steering effect. Varying values of time delay can be used to reproduce other interference effects, such as narrowing the centre lobe of the beam.

The output is handled by 3 four-channel DACs on a common data bus as mentioned before. After the desired phase is calculated for each speaker, the value of the output port’s data register is set to the desired value. Then the output selector is set to specify which of the 4 outputs is written to. Finally, one of the 3 DACs is activated, and the data on the first output pin is set to the desired value. All the 4 outputs of each DAC are updated individually one after the other, altogether taking about 12 microseconds.

# Hardware

We take our input from any consumer audio device, such as a phone or laptop, through a 3.5mm audio jack. These devices are usually designed to drive low impedance headphones without the use of an external amplifier. They tend to use signal levels around 1V peak-to-peak at maximum volume. Our ADC needs a signal between 0V and a given reference voltage Vref, which is at least 2.5V. We decided on a 5V reference voltage as suggested by the ADC’s datasheet. This meant that the signal needed to be amplified to 5V peak-to-peak, and re-biased to be centred around 2.5V, and fed into the ADC.

The purpose of the ADC is to sample the input audio signal and transmit it to the microcontroller in a digital format. The ADC we used was the AD7819. Its schematic can be seen below. It takes an analogue input between 0 and 5V, and returns an 8-bit digital value x, given as x = (Vref−GND)/256.

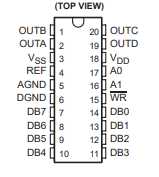


It was selected for being a cost efficient and easy to use component, capable of sampling audio at 44.1 kHz. This sampling frequency was chosen as the smallest that can reproduce signals up to 22 kHz, the human hearing range, based on the Nyquist sampling theorem. The supply voltage VDD, and the reference voltage Vref were connected to +5V, while the data bus DB0-7 pins were connected to the microcontroller [as shown in the table in the microcontroller section]. It performs sampling and conversion and is ready to transmit data in 5 microseconds after the initial trigger signal, leaving plenty of time for the microcontroller to carry out necessary calculations.

The microcontroller board used in the setup was the Arduino Mega, which is a small development package built around the Atmel ATmega2560 microcontroller. It runs on a 16MHz clock speed, giving 362 clock cycles between two consecutive timer interrupts. It is enough to do all the Analog-Digital-Analog conversion, and manipulate the individual DAC outputs to implement phase advancing. It has plenty of 8-bit wide I/O ports, 4 of which we ended up using in this project. The connection is summarised in the table below.

|  |  |
| --- | --- |
| Pin L0…L7 | Data bus of ADC |
| Pin G0 | !CONVST of the ADC |
| Pin G1 | !CS of the ADC |
| Pin G2 | !RD of the ADC |
| Pin A0…A7 | Data bus of DACs |
| Pin K0, K1 | Output select of DACs |
| Pin K2 | !WR of DAC3 |
| Pin K3 | !WR of DAC2 |
| Pin K4 | !WR of DAC1 |
| Pin K5 | BUSY for debugging |

The DACs used were the TLC7226CN from Texas Instruments. The schematic is shown below.



These are parallel DACs. What this means is the output voltage is a function of the 8-bits set to DB0-7. Given an 8-bit value x, the MX7228 sets the output to (Vref−AGND)∗x/256. The output set is defined by the two address bits A0 and A1, and the register is written when !WR is pulled low. The existence of the !WR line allows us to address both DACs from the same 2 pins of the microcontroller and still maintain independent control of all 12 speakers.

One important issue we ran across with this DAC is that when we were originally applying 12V to Vdd and 0V to Vss and AGND, the output was between 0 and VREF at 5V. Fortunately, the DAC supports native bipolar operation, if AGND and VSS are connected to -5V. This meant we could drive the speakers without needing another re-biasing stage built into the speaker amplifiers. Each of the outputs are connected to the inputs of the speaker amplifier circuits.

The powering of all the chips, together with the input amplification circuit was handled by 3 different laboratory power supplies. The amplifier needed a variable supply of around 14.5V for the fine-tuning of the bias at 2.5V to maximise the dynamic range of the signal. The supply pins VDD of the DACs were also connected to this supply. Both the ADC and the DACs needed a 5V supply for power and as reference voltage. Finally, for the bipolar operation of the DACs, a -5V supply was also needed, which was provided by a +5V supply connected to GND with its polarity reversed. To reduce electrical noise and protect the chips from interference with the high-current high-power speaker amplifiers, separate power supplies were used to drive the speakers.