Jacob Porter

CSC 240 Honors Project

Prof. Meacham

5 / 8 / 2023

Arduino Synthesizer

Introduction: Arduino is an open-source platform of both hardware and software. The hardware platform of the Arduino UNO consists of 14 read and writable digital pins, as well as 6 analog inputs. The processor is a ATmega328P that runs at 16 MHz and is not soldered directly onto the board so it can be replaced if need be.

The goal of this project is to construct a digital synthesizer. A synthesizer is an electrical musical instrument that manipulates frequencies to produce musical notes in various ways. This project focuses on using the Arduino programming language to manipulate frequencies and play different musical sounds using the tenants of event driven software design.

IO: The Arduino synthesizer has a total of 5 input devices used to produce different musical effects. The first device is the HC-SR04 Ultrasonic proximity sensor. This sensor uses a series of controlled ultrasonic pulses to determine the proximity of whatever object is in front of the sensor. A mechanical button is paired with the sensor so that the synthesizer will output sound triggered by the HC-SR04 whenever the button is pressed. The second input device is an analog joystick. The joystick works by sending the Arduino Uno updates of the X and Y position of the wherever the joystick is positioned at the time of reading. The last 2 are mechanical buttons that are programmed to trigger different output responses in the synthesizer.

The Arduino synthesizer has one main output device and that is the 1/4 inch line out that is most commonly used in electronic musical production. This ¼ inch line out can be plugged directly into any amplification devices and output the different frequencies the Arduino produces.

<u>Programming Overview:</u> The Arduino platform uses a language similar to that of the C/C++ programming languages. This custom language is imperative, with capabilities to do object-oriented programming. For this project, imperative language was the main paradigm used to complete the project.

This program relies heavily upon reading and manipulating inputs found in the IO devices to correspond to different functionalities in synth. The Arduino has two preset methods that are setup() and loop(). The setup method is the method typically used to initialize and configure settings for the IO used during the execution of the program. The loop() method is what runs continuously over and over again while the Arduino is on, constantly repeating the instructions found in the code. This type of programming where the code is being executed continuously waiting for certain user responses is called “event driven programming”. The program waits for events to happen from the user to enable certain outputs from the Arduino. In the case of the synthesizer, the loop() method runs over and over checking to see if buttons 1-5 are pressed in order to trigger certain functionality of the synth. Each button press represents a different event in the program.

<u>Mathematical analysis</u>: Every piece of music is simply the succession of divisions that use tone and time to create melody and beat. Sound travels in waves, and the interval in which

the waves start and stop is known as frequency. Frequency is measured in Hertz [Hz] and is what tonal music bases all of its intervals on top of. Music in the western world traditionally has the note middle “C” set to 256 Hz and bases the rest of the notes C#-B off of that base frequency. “C” in this case would be called the root of a scale, as the scale would start and end on an “C”. An octave is when one note has its frequency doubled, meaning the same note is being played but at twice the frequency, or at a 2:1 ration. For example, if A is tuned to 440 Hz, an A an octave above would be 880 Hz, and an A an octave below would be 220 Hz. These laws of ratios can be applied to every interval in a scale based upon whatever the starting frequency is.

The following chart is used to list each interval in the major scale and its respective ratio to the root note. An interval in music is the respective distance between two notes.

C = 256 HZ	No interval
D = 256 Hz * (9/8) = 288 Hz	Major second
E = 256 Hz * (81/64) = 324 Hz	Major third
F = 256 Hz * (4/3) = 341.3 Hz	Major Fourth
G = 256 Hz * (3/2) = 384 Hz	Perfect Fifth
A = 256 Hz * (27/16) = 432 Hz	Major Sixth
B = 256 Hz * (243 / 128) = 486 Hz	Major Seventh

These ratios mean that as long as there is a defined root note, any major scale starting at a defined root note can be played. Once all the notes in a scale are found then all that is needed to repeat them in either higher or lower registers is to multiply the value by 2 or ½ depending on whether the user wishes to play up or down.

One common feature in most electronic synthesizers is the use of an arpeggiator. An arpeggio in music is when sequential notes are played in ascending / descending manners to create melody. These notes can simply be the scale, or they can be certain combinations of intervals to provide melodies in a piece of music. An arpeggio is used in this synth as a demonstration of this combination of frequencies.

Code Review: The programming behind this synth uses 5 main methods for its functionality. The first method used is for the ultrasonic sensor. This sensor works by sending out controlled pulses of ultrasonic waves and tries to detect the latency in the echo to determine the distance at which the sound waves bounced back to the receiver. To calculate the distance, the trigger pin of the sensor is turned to HIGH, sending out a signal, then back to LOW, turning the signal off and both HIGH and LOW again to complete the pulse. The pulse is then read by the echo pin, which is turned HIGH in the pulseIn() method to capture the time between each pulse. To process the data coming in from the pulseIn(), the data is processed to float values that correspond to centimeters. This data that is being fed in real time is what is used to play each respective note in the C major scale.

When button three is pressed, the arpeggio() is method uses a base note of C and multiplies out the C to each interval after playing a tone. After a C major arpeggio is run, the base note becomes an F and then an F major arpeggio is played, to create the sound of moving chords in music. This whole process is run in a for loop so that it will play 2 times while the button is pressed.

Button three is pressed when the user wants to enable the joystick. The joystick has 2 axes, those being the x and y axis. When the button is pressed, the Arduino checks the position of the joystick and then adds 200 to each position that is being read off of the analog read pins. This is to ensure that when a tone is output, it is within a certain range of frequencies that sound audible to the player. The x-axis position of the stick controls the pitch of the note being played, the further positive the stick is in the x direction, the higher the pitch. The y-axis position of the stick controls the delay between each note. This means that as the stick is further positive in the Y direction, the delay at which each note is played one after the other increases. This method is in a for loop which runs 12 times, checking each position of the joystick.

When button four is pressed, the starwars() method plays the main theme from the Star Wars movies twice. Each note in the theme was calculated by determining its relationship from the root note, and then played after a certain delay value was set to give the melody proper time. After each note, delay() is called with an integer parameter passed in that represents the time in millisecond the Arduino pauses in its execution of the loop() method.

When button five is pressed, the octave() method plays a specified base note and doubles the value by an octave, playing the same note at double the frequency. This method runs in a loop for 5 times while a note and its double are played simultaneously with a 100ms delay in between.

Works Cited

Chip Phonic. “Simple Arduino Synthesizer Build.” *YouTube*, 4 Feb. 2021, www.youtube.com/watch?v=x07FF-A67Nc.

Musical Scales and Intervals. hyperphysics.phy-astr.gsu.edu/hbase/Music/mussca.html.