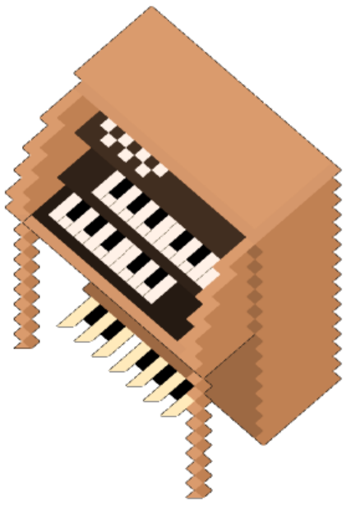


Design Document

DEC13-09 Chiptune Synthesizer

Team Members:	Dustin Amesbury
	Wallace Davis
	Darren Hushak
	Brittany Oswald
	John Tuohy
Client:	Dr. Joe Zambreno
Advisor:	Dr. Nathan Neihart

Project Charter Version Control

Version	Date	Author	Change Description
0.0	14 March 2013	Brittany Oswald	Document created
0.1	14 March 2013	Everyone	Group collaboration via GoogleDocs to fill out most of the sections
1.0	15 March 2013	John Tuohy	Assembled and formatted the document
1.1	29 April 2013	Brittany Oswald, Dustin Amesbury, and John Tuohy	Revised and incorporated feedback
2.0	9 December 2013	John Tuohy	Updated entire document

Table of Contents

Roles and Responsibilities

Dustin Amesbury -- *Project Leader* **--** As the project leader, Dustin is in charge of making sure the team stays on track throughout the semester as well as leading team meetings. Dustin will also be involved with both Raspberry Pi modules.

Wallace Davis -- *Webmaster* -- Wallace is in charge of maintaining the public website and will be involved with implementing the songbox features. Wallace will also be assisting Darren with assembling and testing the electrical components of the chipophone.

Darren Hushak -- *EE Director* -- Darren will be the main advisor and executor of any hardware component of the project, including restoration of the organ console, its keyboards, and any of the original componentry that we wish to keep. Furthermore, Darren will be responsible for the conversion of the keyboards to MIDI controllers, addition of user input controls, and the interconnection between the MIDI controllers, the song playback and recording unit, and the sound modules. Lastly, as a music technology minor, Darren will advise and direct the group regarding standard music technology practices and music technology structures.

Brittany Oswald -- *User Experience Designer* -- Brittany is going to ensure the finished product is able to be used by users of all skill levels and abilities. Brittany will work on retrofitting and equipping the organ with the buttons and knobs and other user interface elements. She will also participate in writing code for the sound generation and ensure a quality experience for the user. Brittany's focus will be to see the project to completion.

John Tuohy -- *Communications Liaison* -- Along with communicating with the client and advisor, John is involved with designing and creating the software synthesizer on the Raspberry Pi.

Definitions and Acronyms

Term	Description
Channel (MIDI)	The MIDI specification allows for 16 MIDI “Channels,” which are used to direct different streams of data to different destinations. Each event has a MIDI channel associated with it, and will therefore be routed based upon its channel. The ‘broadcast’ channel is set per controller, and the ‘listen’ channel is set per module. All MIDI channels are broadcast to every module, but it’s up to the module to pick the desired channel to listen to.
Chiptune/8-bit music	“Chiptune” music can be defined as a genre of music born out of the restrictive musical palette of 8-bit PCM audio used in the early popular gaming consoles. 8-bit refers to the bit depth of PCM audio, in that the digital form of the audio can only take on one of 2^8 values, which creates incredibly unique sounds.
Chipophone	The unofficial name of our chiptune synthesizer
Controller (MIDI)	A MIDI controller, most commonly a keyboard, is a device that takes user input and converts it to MIDI data.
Instrument/Voice (MIDI)	An instrument or voice is a software program or hardware component that takes MIDI data as an input and outputs either PCM or analog audio.
MIDI	MIDI, standing for Musical Instrument Digital Interface, is a standard for interconnection and communication between digital music devices. It is a serial protocol that only conveys events relating to musical performance, such as notes, note volume, and control values for manipulation of audio. MIDI itself does not convey any audio information.

MIDI CPU	A MIDI CPU is a hardware circuit board that takes in analog or switched input from user interface devices (switches, potentiometers, encoders, etc), and outputs MIDI data.
Module	A MIDI module is a device that hosts an instrument/voice, which takes MIDI input and outputs audio. The module is the shell for the voice, and the voice refers to the algorithm or circuitry used to actually produce audio.
Raspberry Pi	Our chosen hardware platform for both our computational and signal processing needs. It is an incredibly small, yet powerful, bare-bones computer that runs a linux operating system.
Synthesizer	A synthesizer can refer to an analog or digital piece of equipment or software that can self-generate audio using various algorithms and techniques.

Executive Summary

Our challenge is to create a device (nicknamed “The Chipophone”) that will be used as a display piece for the EE/CprE department. Our “chipophone” is a synthesizer that plays music using “chiptune” or “8-bit” sounds commonly associated with early console gaming systems. We are tasked with taking an existing electric organ (one which has already been donated to the project) and retrofitting it to play 8-bit music. All of the pedals, switches, and keys will be wired to create MIDI signals that will be interpreted and rendered into audio output. After the organ has been created, it will be placed in the TLA and will be used as a interactive showpiece to supplement the two arcade machines and the virtual pinball machine already located therein.

Project Overview

Our solution involves two stages: MIDI retrofitting the organ, and creation of an audio generation unit. We will be using MIDI CPUs to generate MIDI signals, and Raspberry Pi microcomputers to process and interpret said MIDI signals. To retrofit the organ, we will remove most of the internal mechanisms and circuitry of the organ and rewire the keyboards and button/switchboard interfaces into MIDI CPU chips. These chips will then be routed into the Raspberry Pi boards, whose audio generation code is our responsibility.

The Chipophone will also allow for many features native to digital synthesizers For example, it will hold in memory various voices loadable into five different modules simultaneously; user creatable voices; mixing of multiple channels, etc. An additional third stage, to be implemented in the second semester of the project, will be to create a “Songbox” module, which will allow for recording and playback of both preloaded and user generated songs.

System Requirements

Non Functional

1. Usability
    a. Synthesizers in general can greatly range in complexity and appear intimidating to new users. Our goal is to make our synthesizer easy to use for everyone, both advanced users who have specific goals when they play music and for a beginner user who wants to make some cool sounds.
2. Portability
    a. The chipophone will be easily transported from event to event. It will have casters so it can be moved. The only cable will be a single power cable which plugs into an normal outlet.
3. Performance
    a. The chipophone will have no noticeable latency between keypress and sound. The chipophone will also be able to boot and be ready to play in under 30 seconds.
4. Maintainability
    a. If anything goes wrong with any of the Raspberry Pi boards, they will be able to be easily swapped out for a new one.
5. Security
    a. There will be a lock on the back of the chipophone, preventing anyone from stealing the boards or other equipment.

Functional

1. User Controls
    a. Keyboards and Pedals
        i. Shall physically interact similarly to a regular organ
        ii. Each key shall produce a sound in tune with the corresponding note played on a regular (tuned) organ
    b. Voices
        i. Able to change the instrument associated with each keyboard and the pedals
        ii. Able to have the same instrument associated with more than one keyboard
    c. Buttons and Knobs
        i. Adjust pitch/frequency
        ii. Adjust ADSR envelope
        iii. Change instrument of pedals and each keyboard independently
    d. Songbox
        i. Selection and play default demo songs
        ii. Pause current selection being played
        iii. Record input
        iv. Pause recording input
        v. Play recorded input
        vi. Loop recorded input
2. Technical
    a. Use MIDI Standard
    b. Boot time shall be less than 20 seconds, but faster is better
    c. Instruments
        i. There shall be 5 types of waves: two square waves with variable duty cycles, one triangle wave, one sawtooth wave, and one representing noise

Detailed Design

User interface specification

The keyboards and foot pedals are used to play musical notes by the user. The keyboards and foot pedals can have different voices set to them along with several other characteristics about the voice edited. This is done with the user interface panel on the front of the organ above the top keyboard. There are several encoders, switches and arcade buttons on this panel. The arcade buttons were chosen to fit the video game theme of the project. These inputs all fed into an Arduino microcontroller which produces MIDI control signals sent to the Raspberry Pis to edit the voices.

I/O Specification

The user will interact with the keyboard keys, buttons, and pedals. These input components are all hooked up in a switching array to the MIDI CPUs. The MIDI CPUs will generate MIDI data based on what input they are getting. This MIDI data will enter the songbox Raspberry Pi through the GPIO pins on the Pi. The songbox will add any additional MIDI data required and pass on the MIDI data to the synthesizer Raspberry Pi, once again through the GPIO pins on the Pis. The MIDI data gets processed on the synthesizer and sound is produced through the 3.5mm sound jack output on the Raspberry Pi and goes to the speakers.

Hardware Specification

We are using two Raspberry Pi model B computer boards, four MIDI CPUs, and an existing organ that’s had its internal mechanisms removed and discarded to construct The Chipophone. The organ keys, pedals, and switchboard will interface with four MIDI CPUs that produce MIDI data that gets interpreted by the Raspberry Pi boards. The whole chipophone will be powered by three DC power supplies all tied together to a single AC receptacle.

Software Specification

All software will run on a distribution of Raspbian Wheezy. This is a Debian distribution made specifically to run on the Raspberry Pi. We will be using the PortAudio library to interface with the sound drivers on the operating system. The PortMIDI library will be used to translate the UART input from the Raspberry Pi into MIDI. The audio synthesis and MIDI handling software will be written in C++.

Simulations and Modeling

For the first semester of this project, we will have a working prototype of the Chipophone. This prototype will have basic functionality with a polished look and feel. The functionality we’re looking to achieve is the ability to play any note on the keyboard without any modulation. This means basic square/triangle/noise waves. Simulating this prototype is a very simple process that can be done entirely on a computer. We’re able to test the entire synthesizer in any Linux environment, which is what the Raspberry Pi is based on.

The second semester will be spent refactoring the prototype into modular C++ code. Simulating and testing this software can be done on a laptop independent of the rest of the system with a USB MIDI controller.

Implementation Issues/Challenges

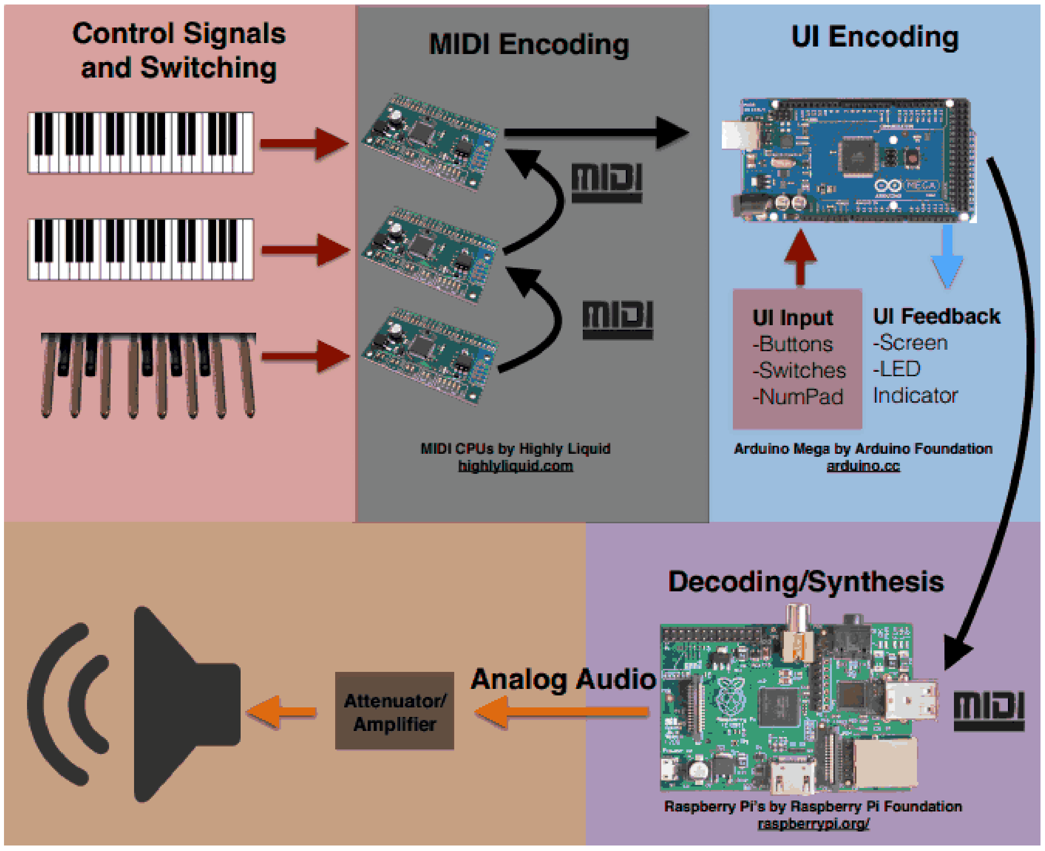
Implementation of the synthesizer comes with many challenges because of a few things. The first issue is that the Raspberry Pi has an ARM based processor, meaning that code we write on our own computers likely doesn’t have compilers designed to make binary executables for ARM processors.

Another issue we’ve come across is that the Raspberry Pi and PortAudio have their own quirks about recognizing default audio devices. This has been solved by reconfiguring the library to match our device.

The Raspberry Pi Model has a 700 MHz processor, which is powerful for a microcomputer. However, audio synthesize is a very taxing process. We have come across many software efficiency problems where we need to optimize parts of the software. Our audio callback is called 32,000 times every second, so it needs to be very efficient.

Design Documents

System Design

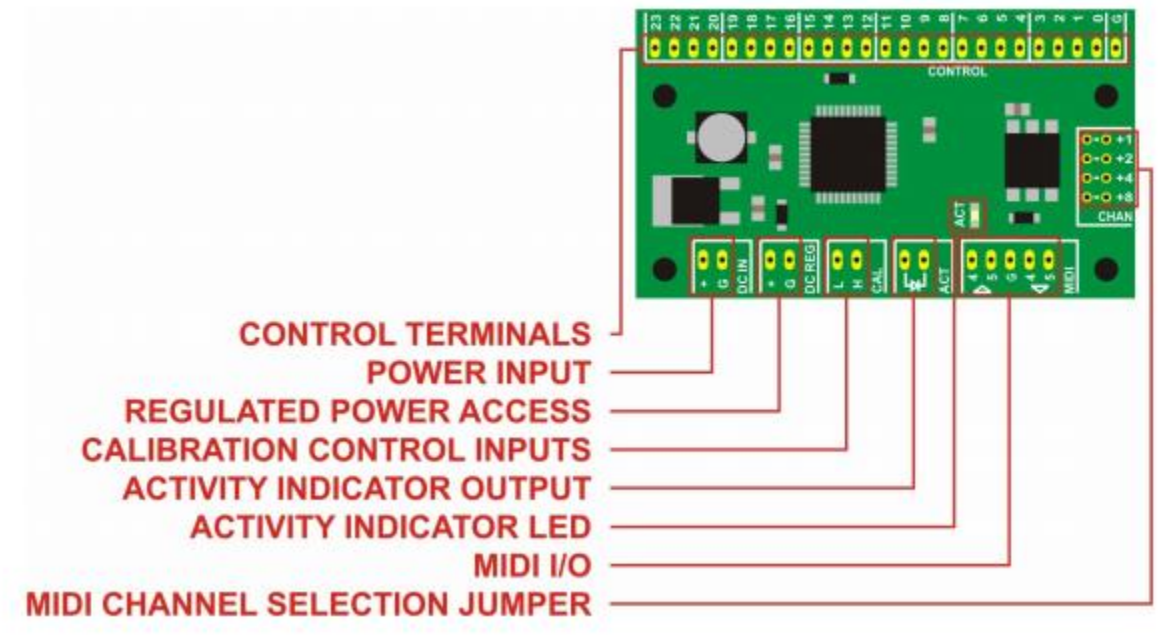


Each of the keyboards, the pedals, and the buttons will each have their own MIDI CPU, which will take the user input and translate it to MIDI data. The songbox module will receive the MIDI data from the keyboard and the Arduino and pass the data through to the synthesizer. The songbox may also record the MIDI data or inject more MIDI into the stream. The synthesizer receives all of the MIDI from the songbox and uses that information to update the data model and fill the audio buffer for the sound card to play.

Hardware Design

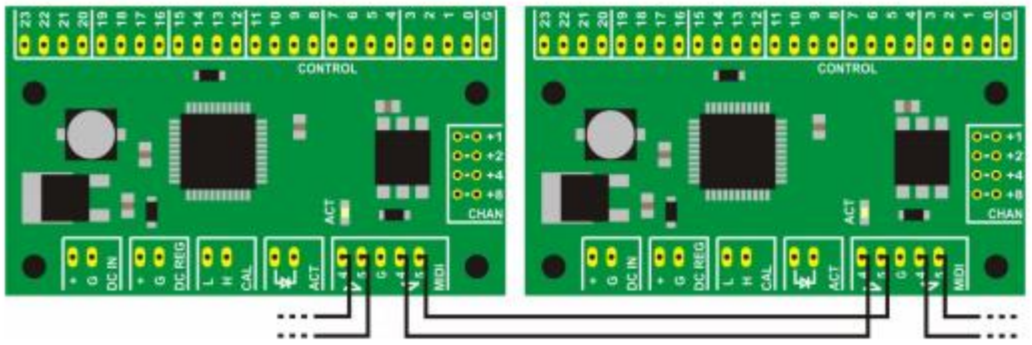
The hardware for this project, on the surface, is relatively straightforward: we're using an old organ console, its two keyboards, and its pedals as the base for user input. Each keyboard must have an independent channel, and each MIDI CPU can only output one channel at a time, therefore each keyboard and the pedals will have their own CPU to encode MIDI data.

The layout of a MIDI CPU is shown below:



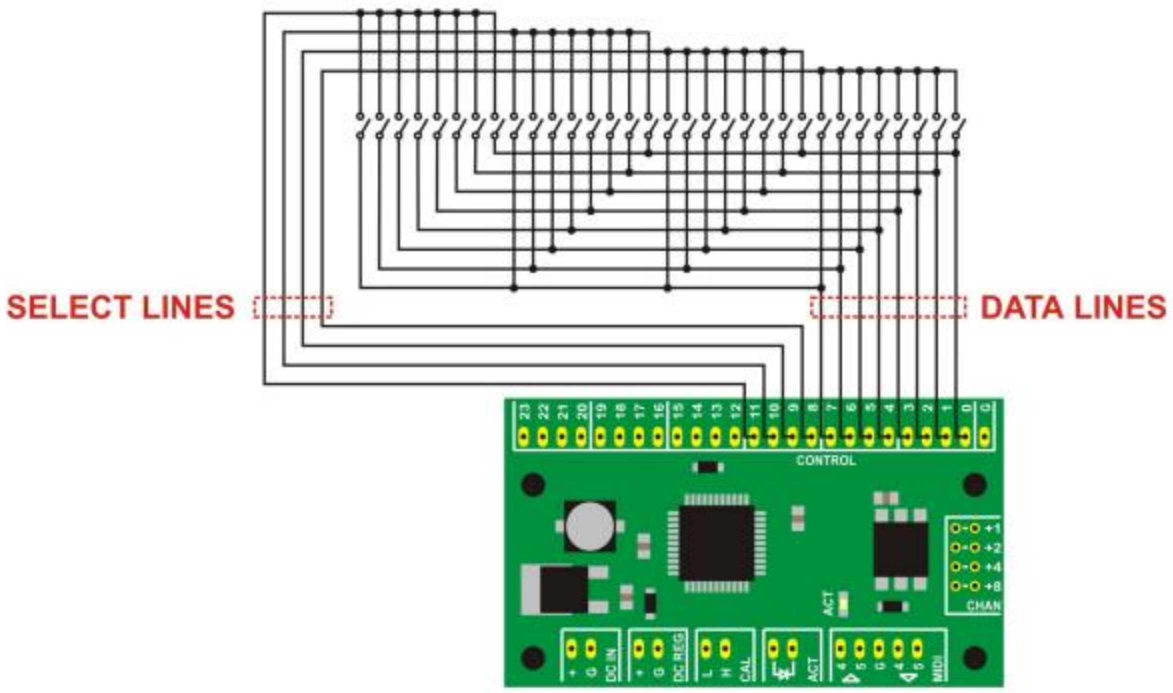
- The control terminals are what each of the switches will connect to (scheme for connections covered later)
- The power input is where the +5V supply connects
- Power access is simply a doubling of the input power pins (mainly for putting a power indicator LED in the unit)
- Calibration control inputs are for calibrating analog inputs, such as potentiometers. In this design, for any sort of variable (non-note) user input, we will be using encoders, so these connections will be unused
- Activity Indicator pins are where an LED may be placed to indicate MIDI activity on the chip; there is a built in activity monitor LED right on the chip
- The MIDI I/O pins are where the encoded MIDI data passes out of the MIDI CPU; the input is used only for programming the CPU; otherwise the input passes straight through the output
- Finally, the MIDI channel selection jumpers are used to set the MIDI channel of each CPU

Because the MIDI input of each CPU is directly passed to the output, the multiple CPU’s can be strung together, as shown in the following figure:



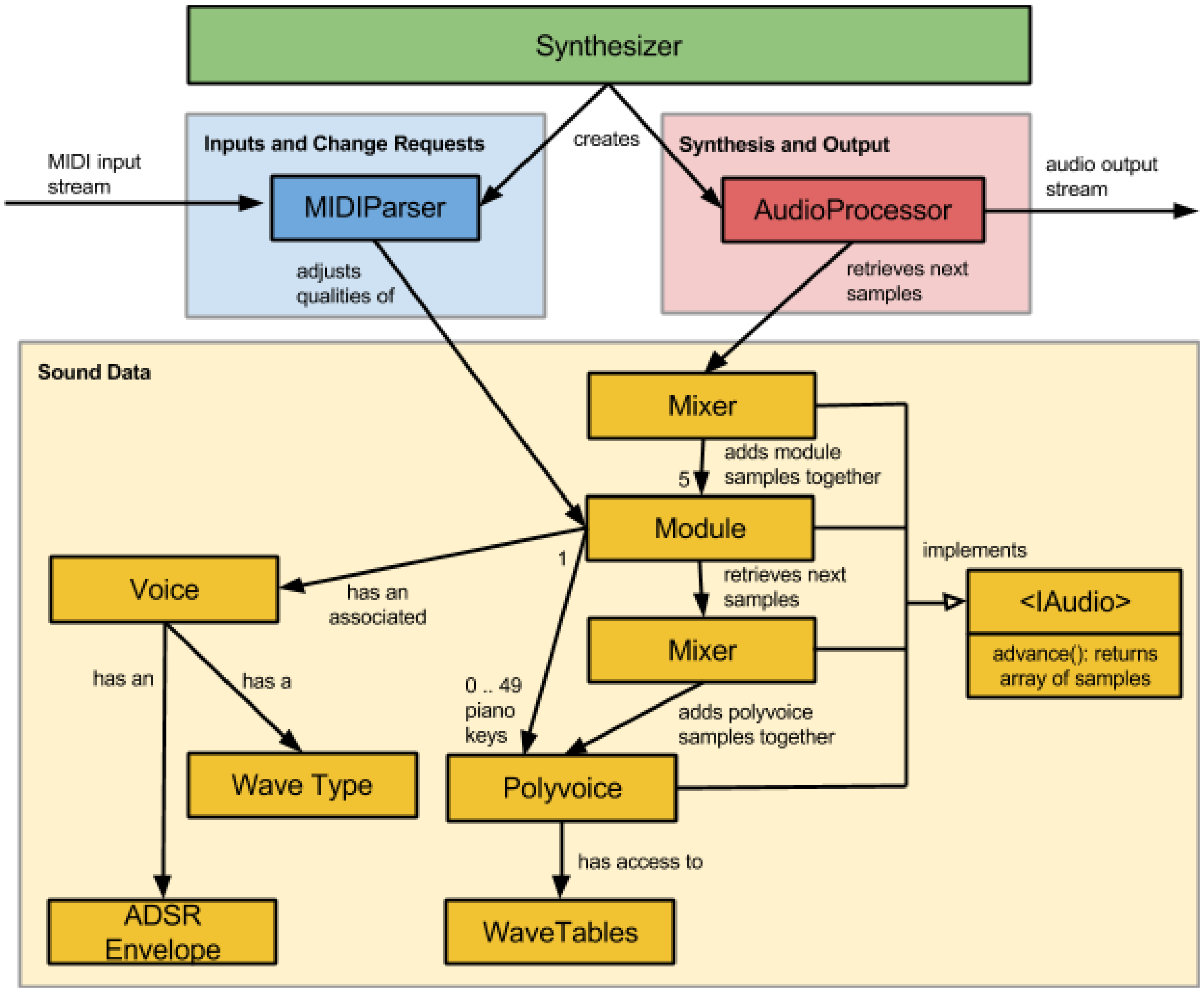
The final MIDI CPU will have a serial stream of all of the MIDI data produced by the user. The MIDI spec specifically allows for this architecture, and the concatenation of multiple serial streams is handled by the CPU without any accidental bit-mixing. This final MIDI serial stream will then be passed into the Raspberry Pis' GPIO pins.

Finally, in order to connect each key to the MIDI CPU, we need a scheme a little more complex than a one-key-to-one-input scheme. Luckily, the MIDI CPU allows for such a scheme. By reconfiguring a few of the control pins, we can create a switching matrix, allowing for more than 23 keys to trigger MIDI events on the CPU. A small overview circuit is shown below:



Between the CPU's switching matrix scheme and its ability to read encoder inputs, all of the user interface hardware is able to be handled by only 4 CPU's, all run serially together to send a single MIDI signal to the Raspberry Pi's for decoding and synthesis.

Software Design



The diagram above shows a more detailed view of the software design inside the “Parsing and Sound Generation” module pictured in the System Design Diagram. It is important to note that the input of this module doesn’t change whether the songbox is attached or not. There are three main components to this design: Inputs and Change Requests, Sound Data, and Synthesis and Output.

The Inputs and Change Requests component (blue box) will read the MIDI data and parse it. The MIDI standard describes sounds, but also includes program change requests, so this module will also be a decision maker and take actions based on the interpretation of the MIDI commands. Actions might include changing the qualities of the ADSR envelope for a certain module, adjusting pitch of a note, and more. This component has write access to the information in the Sound Data component.

The Sound Data component keeps track of the current state of the organ. This is important because the user will be able to assign different voices or “instruments” to each of the keyboards and the pedals, and the user will have the freedom to adjust the types of waves and qualities of the sounds resulting from the organ, and this information needs to be readily accessible and organized when creating the synthesized wave in the Synthesis and Output component.

In the Synthesis and Output component, there is an audio callback which fills a buffer one element at a time based on the current Sound Data and wave properties. When this buffer is full, it is sent to the sound card, and beautiful music is made.

Standards

Our chipophone uses the MIDI standard to communicate between the different components in our system. MIDI is a technical standard that describes a protocol, interface, and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. The MIDI protocol has set parameters that describe the types of information that can be sent using the MIDI interface, and the methods to send this information between electronic devices and musical instruments.

We are using this standard to provide modularity in our project design and also to allow versatility to the users of our project. By using MIDI, any user can connect a device that communicates using the MIDI specification and send the MIDI messages through the chipophone for synthesis. On the other hand, if the user has their own MIDI receiving device, they are able to plug into the MIDI out, and create their own interactions because we follow the standard methods of communicating MIDI serially. This also allows the information being communicated using MIDI within our system to be easily output and shared with anyone who might be interested.

Testing and Evaluation

The user interface interfaces with the Arduino Mega and produces the correct MIDI values for each input device. Using a laptop, a USB to MIDI interface, and a software program called MIDIMonitor, we were able to listen to the stream of MIDI and verify that the intended message was being sent from the user interface elements and from the keyboards.

The software is capable of reading from any MIDI device and synthesizing the notes from the MIDI device to produce sound. For example, we had a small USB keyboard which aided in testing code functionality without depending on the organ keyboards or hardware. To test specific classes, we developed unit tests which were run to ensure new features were working correctly and also to ensure new features didn’t break the existing features.

Each of the system components when connected together result in the overall The Raspberry Pi successfully produces audio based on MIDI input from the keyboards and user interface elements. This was tested by playing a known tune, and aurally verifying the sound quality. Specifically, the goal is to hear no loud speaker pops or other sound defects while the music is playing.

Project References

Description	Location
Linus created a similar chipophone	http://www.linusakesson.net/chipophone/
MIDI Specification	http://www.midi.org/techspecs/
MIDI Bit Table	http://www.midi.org/techspecs/midimessages.php