## I. Abstract

We present here a system designed to extract velocity and pitch information from an acoustic piano performance and translate it in real-time into corresponding MIDI messages. Such a system has a number of interesting applications, including real-time control of electronic sound sources, usage as a learning/compositional tool, and the archiving of musical performances in a concise digital format. Special attention was paid in our implementation to latency and the ease of retrofitting any existing acoustic piano with the system. With these goals in mind, we used IR reflectance sensors attached to a group of high-speed PIC32 microcontrollers as the primary components of our design. The system achieved effectively negative latency end-to-end. That is: appropriate digital data was conveyed to the target system before the corresponding acoustic event occurred.

## II. Introduction

The system specified below is designed to digitize piano performance data in a standard format called MIDI. As mentioned, there exist several interesting applications for a system of this nature. Specifically, we can mount it in a piano and utilize it in any situation where a digital keyboard can be used, while also being able to take advantage of the acoustic sound and the feel of a piano key, which digital keyboards often emulate but generally fall short of. Consider a live performance situation in which a pianist wants to control a software synthesizer. One can imagine a convoluted setup involving a digital keyboard on the side of a piano that requires rapidly switching between the two. The simultaneous triggering of complex sequences on the piano and the synthesizer in question would furthermore be all but impossible. On the other hand, with a digital interface built in to the piano, we gain a great deal of flexibility in determining how and when the synthesizer is triggered without ever having to move our hands from the piano keyboard. Another interesting application enabled by this device is feedback for learning the piano. A teacher can track a student’s progress on a piece by recording their performance digitally and comparing it to a score to identify problem areas in timing and dynamics.

## III. Problem Background

Below we provide an overview of the technical information vital to implementing the system.

### A. MIDI

MIDI is a protocol for communicating musical performance information between devices. A MIDI message consists of 3 8-bit words: 1 status byte indicating the type of message being transmitted and 2 data bytes. Real-time performance data is transmitted via Channel Voice messages which specify one of 16 MIDI channels as the destination. For our purposes, we are mainly concerned with “Note On” and “Note Off” messages in this category, which include information about note number and how hard the note was played (i.e. velocity) in the case of Note On. Note number and velocity form the two data bytes of a Note On message, and each takes on a 7-bit value. Besides these types of messages, there exist "Continuous Controller" messages that are somewhat more open-ended in their interpretation and consist of controller number and value (both 7-bit). Such messages are often used to map knobs and sliders on a control surface to timbral parameters on the destination system, for example.

## B. Piano Key Measurements

One of the most important requirements of the system is that it accurately communicate the velocity with which a key is pressed to the target system. We rely on the fact that velocity can be determined by looking at the slope of the curve representing the height of the key versus time during a note event. The steeper the slope of this curve, the harder the note was pressed. The below diagrams characterize such curves for two distinct levels of force applied to a key:

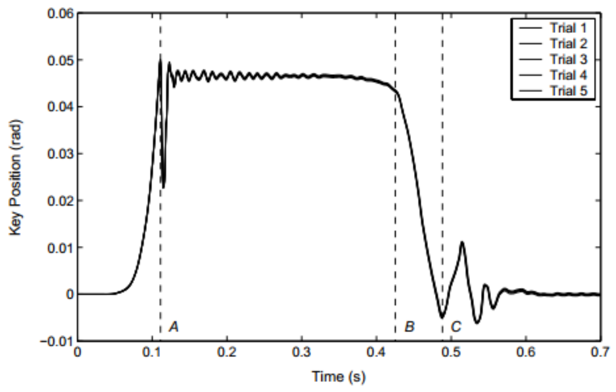


Figure 3.10: Rotation of Key for Five Trials with *Forte* Blow $(\theta_k(0) - \theta_k)$

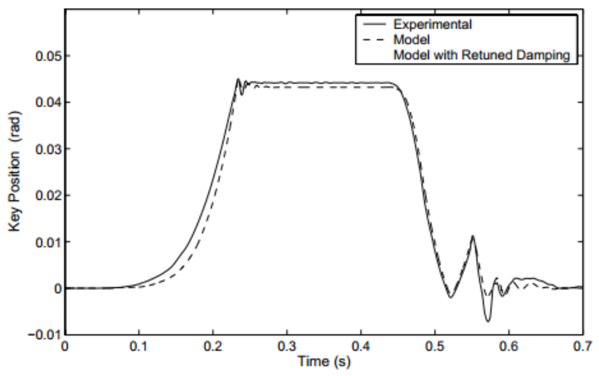


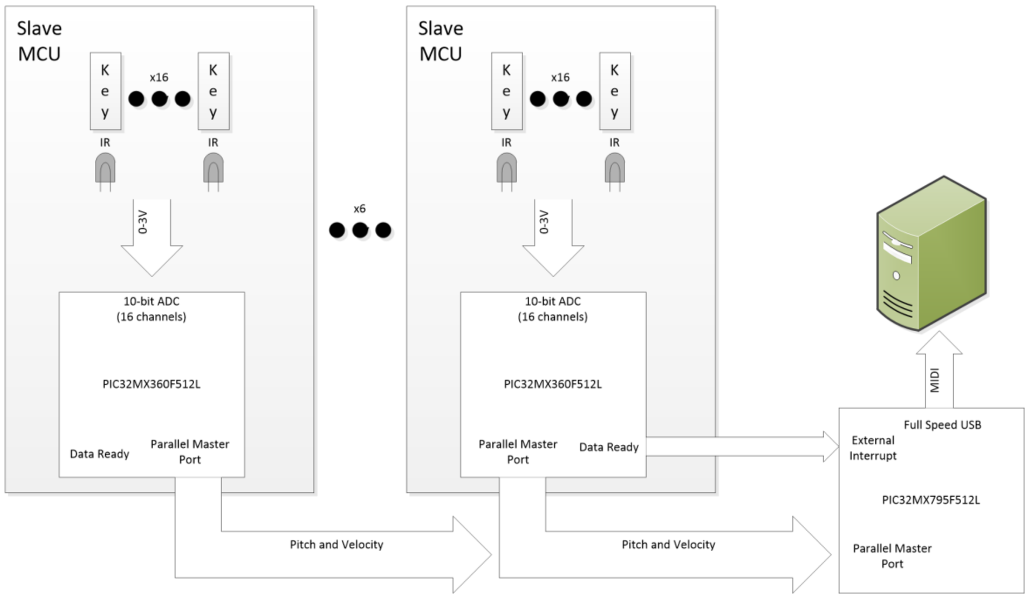
Figure 4.6: Experimental and Model Rotation of Key for *Piano* Blow $(\theta_k(0) - \theta_k)$

As seen, the time it takes for the key to achieve maximum depression varies significantly between a *piano* (soft) blow and a *forte* (hard) blow. For our purposes, we assume that estimating the curve resulting from striking the key as a linear segment is sufficient to determine the velocity.

## IV. System Architecture

### A. Overview

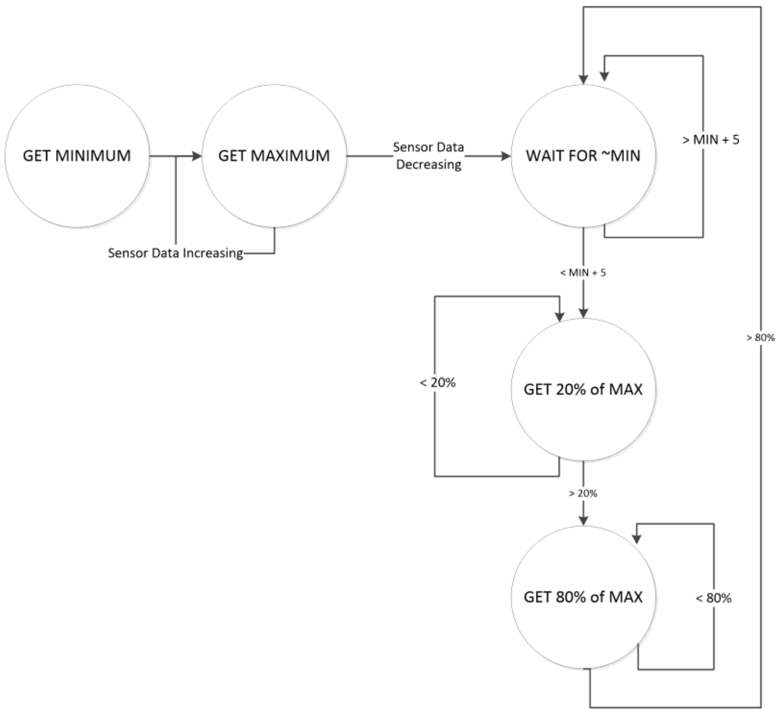
The system design is optimized for the most important performance metric, latency, while also keeping the cost of materials within a reasonable range. As such, a series of PIC32 microcontrollers interfaced to IR reflectance sensors perform velocity extraction, with each microcontroller handling 16 keys at most (one will handle only 8). When one of these MCUs detects a new note event, it relays the appropriate information to a master MCU which then translates the information into a MIDI message and relays it over a USB connection to the target system. A high-level overview of the system is shown below:



Each microcontroller is equipped with a 16-input ADC with a maximum speed of 1MSPS and scans its inputs cyclically as quickly as possible, tracking the state of each key. Information is exchanged between the slave MCUs and the master via an 8-bit parallel port specific to Microchip products known as the Parallel Master Port (PMP). Slaves indicate that they have new data for the master by triggering an external interrupt.

### B. Velocity Acquisition

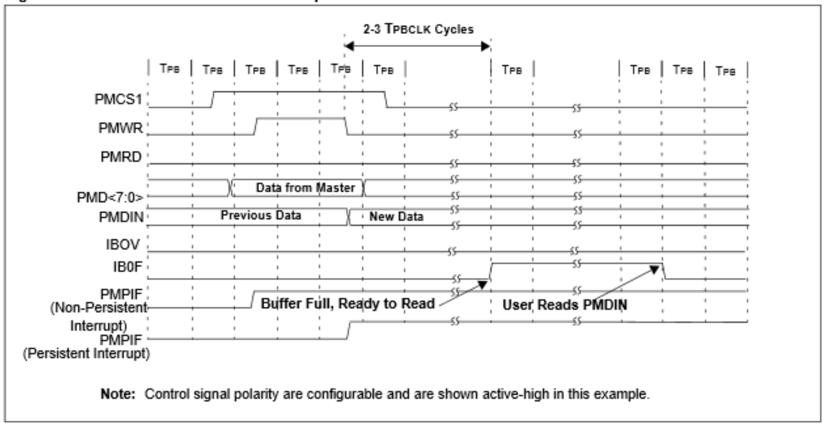
Extraction of velocity from note events is handled by the slave microcontrollers. There is an initial calibration stage on system startup to determine the minimum and maximum voltage output of each sensor, after which the system scans for note on events. To identify and characterize such events, the system keeps track of the state of each key, recording a timestamp when the sensor output reaches 20% of its maximum and another at 80% and using this difference to determine velocity. If the sensor output drops down to minimum before this 80% mark is ever reached, the note is simply thrown out. The diagram below illustrates this whole procedure:



C. **Parallel Master Port**

Inter-IC communication is handled via the Parallel Master Port. The port includes 8 data lines and two strobe lines: one to perform a read operation and one to perform a write, as well as two chip select lines. All data transfer is handled by the master, and thus when a slave wishes to write data to the master, it must wait until the master has handled the previous message sent to it. Additionally, in order to indicate to the master that there is new data to be transferred, the slave sends a pulse to one of the master's external interrupt pins. A timing diagram for a slave write operation on the PMP is shown below:

Figure 13-33: Parallel Slave Port Write Operation


2-3 TPBCLK Cycles
TPB
PMCS1
PMWR
PMRD
PMD<7:0>
Data from Master
PMDIN
Previous Data
New Data
IBOV
IB0F
PMPIF (Non-Persistent Interrupt)
Buffer Full, Ready to Read
User Reads PMDIN
PMPIF (Persistent Interrupt)
Note: Control signal polarity are configurable and are shown active-high in this example.

The timing of the operation is handled in software by configuring the appropriate wait states. The amount of time to hold the data in the buffer and the time between data setup and writing/reading are both configurable in this way. In our implementation, we specify the total amount of time that a read/write cycle takes as 20 clock cycles, or 250 ns at our system clock speed of 80 MHz. Note data from the slave is communicated as a sequence of two 8-bit writes: the first specifying the note number and the second specifying the velocity, with a velocity of 0 signaling a note off.

### D. **USB-MIDI**

For communication to the host system, the master MCU takes advantage of the fact that a standard USB device class exists for MIDI devices, eliminating the need for a custom driver. The master is configured as a USB MIDI device with one IN endpoint and one OUT endpoint set up for bulk transfer between the host and the microcontroller. Development is further simplified by the usage of Microchip's USB stack code to provide some degree of abstraction and a template for specifying USB descriptors/endpoints/jacks. Our master board contains a full-speed USB interface, which allows it to achieve a maximum throughput of 8 Mbps.

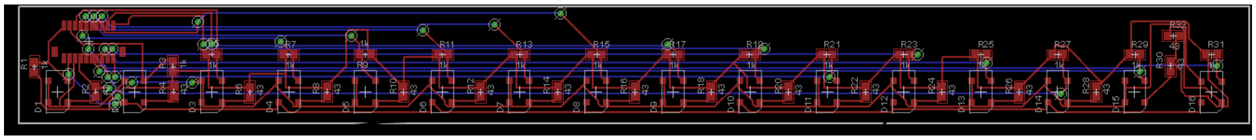
## V. **Implementation**

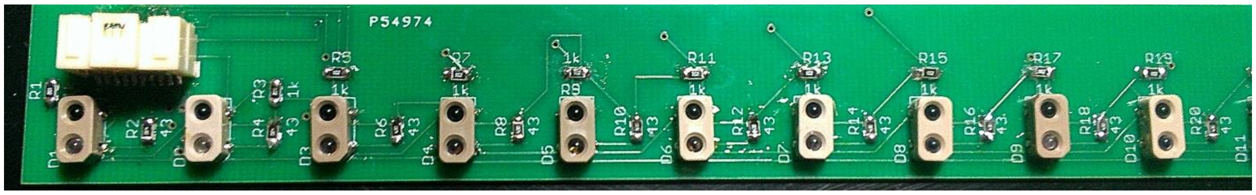
A single slave unit and master were implemented and tested. The hardware used is specified below:

### A. **BOM**

- 1x PIC32 Starter Kit
- 1x PIC32 USB Starter Kit II
- 2x PIC32 Starter Kit IO Expansion Board
- 16x OPB733TR IR Reflectance Sensors
- 16x 0805 43 ohm resistors
- 16x 0805 1 kohm resistors
- 1x Molex 20-conductor Pico-Clasp housing/receptacle pair
- 1x Sensor PCB

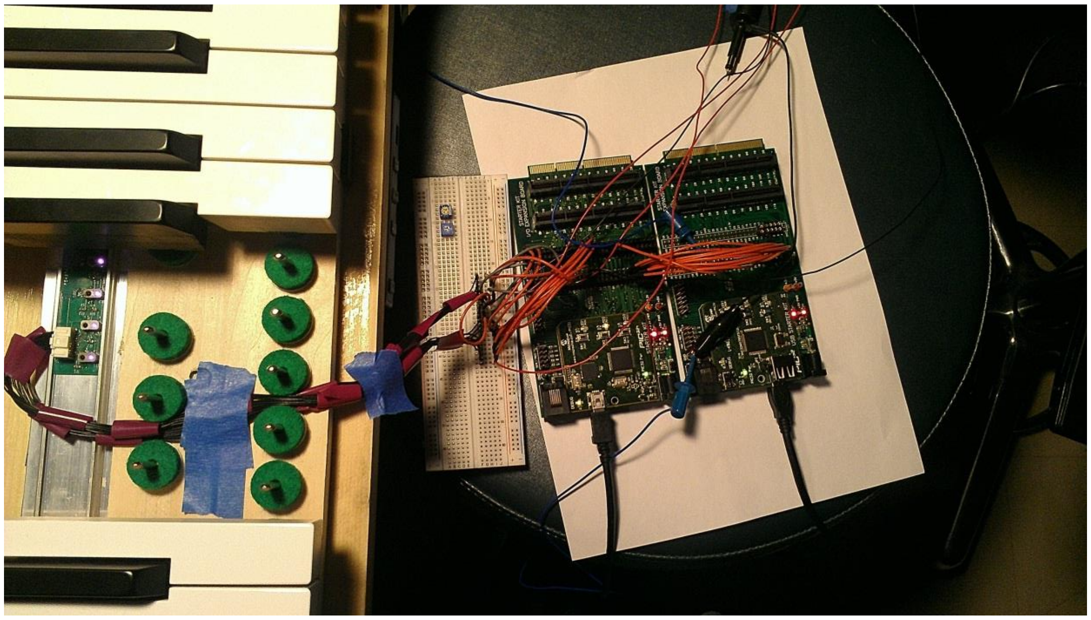
B. **Sensor PCB Layout**





C. **Setup**

The PIC32 Starter Kit was used as the slave MCU and the USB Starter Kit was used as the master. The two boards were interfaced via the I/O Expansions Boards, which expose the pins of the microcontrollers and so allow us to simply hook up the appropriate pins with prototyping wire. The sensor board is mounted underneath the keys of an upright piano and attached to the I/O Expansion Board of the slave, which is mounted external to the piano, as illustrated below.



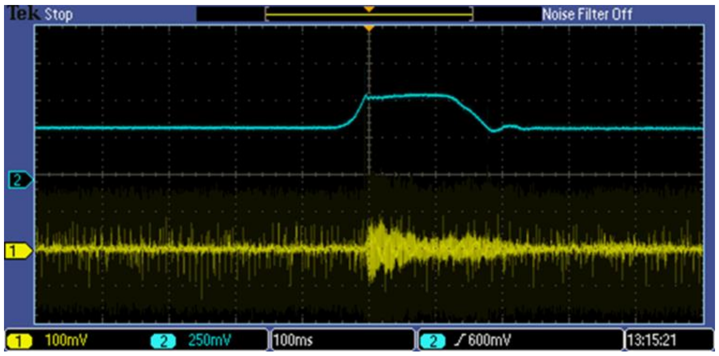
D. **Schematic**

Refer to Appendix A for a schematic illustrating the connections between the master and slave. All connections built-in to the starter kits are omitted.

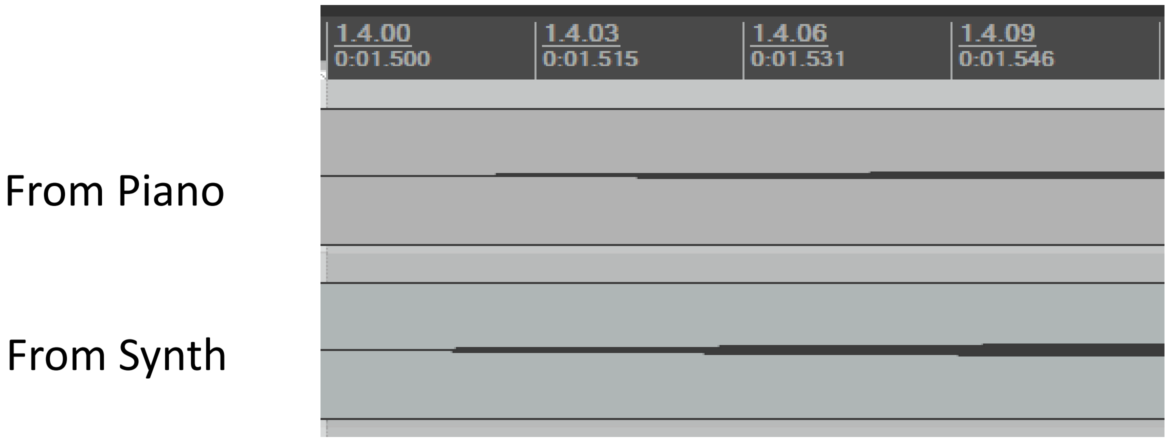
## VI. Results

The system was interfaced with a Muse Receptor to analyze the end-to-end latency. To measure the latency, a microphone mounted next to the piano and a ¼” output from the Receptor were input to an Alesis io2 audio interface and recorded into software. The time between the peaks on the two imputs resulting from striking a key was examined. As illustrated, the audio from the Receptor actually arrives before the audio from the microphone. This is partly due to the fact that we use the slope between 20 and 80 percent of maximum depression of the key as the metric for velocity, and largely due to the speed of communication, which is more than sufficient for relaying any realistic number of MIDI messages we could generate with the system in under a millisecond.

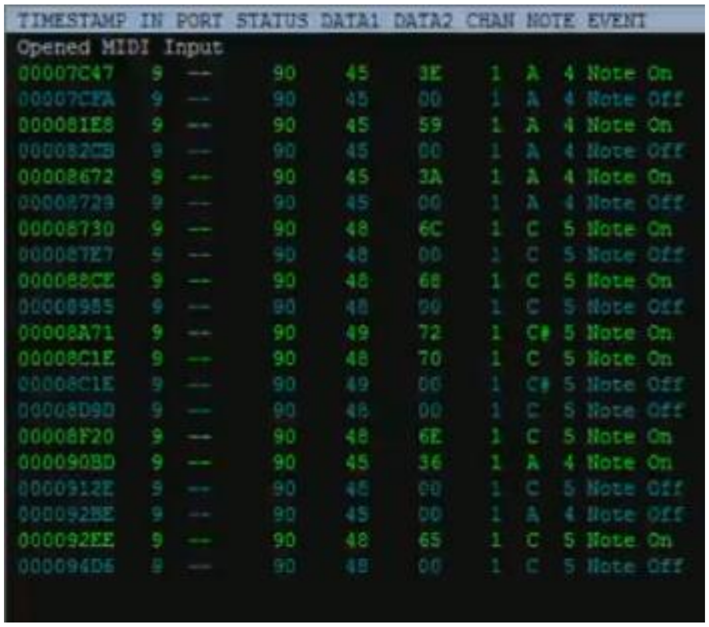
Output of the sensor(blue) and mic output (yellow) when striking a note



Output of mic vs Receptor. The Receptor note starts approximately a millisecond before that of the piano.



Additionally, we wanted to assess the range of velocities that could be generated by the system. To do this, we attached the system to a computer and used a program called MIDI-OX to look at the raw messages generated. As shown below, when playing at dynamics ranging from piano to forte, we obtain velocities from 0x3A (58) to 0x72 (114). This demonstrates that the system is able to respond appropriately to how hard a key is pressed, although more work may need to be done to optimize this response to feel natural.



```
TIMESTAMP IN PORT STATUS DATA1 DATA2 CHAN NOTE EVENT
Opened MIDI Input
00007C47   9  ---    90    45    3E    1  A   4 Note On
00007CFA   9  ---    90    45    00    1  A   4 Note Off
000081E8   9  ---    90    45    59    1  A   4 Note On
000082CB   9  ---    90    45    00    1  A   4 Note Off
00008672   9  ---    90    45    3A    1  A   4 Note On
00008729   9  ---    90    45    00    1  A   4 Note Off
00008730   9  ---    90    48    6C    1  C   5 Note On
000087E7   9  ---    90    48    00    1  C   5 Note Off
000088CE   9  ---    90    48    68    1  C   5 Note On
00008985   9  ---    90    48    00    1  C   5 Note Off
00008A71   9  ---    90    49    72    1  C#  5 Note On
00008C1E   9  ---    90    48    70    1  C   5 Note On
00008C1E   9  ---    90    49    00    1  C#  5 Note Off
00008D9D   9  ---    90    48    00    1  C   5 Note Off
00008F20   9  ---    90    48    6E    1  C   5 Note On
000090BD   9  ---    90    45    36    1  A   4 Note On
0000912E   9  ---    90    48    00    1  C   5 Note Off
000092BE   9  ---    90    45    00    1  A   4 Note Off
000092EE   9  ---    90    48    65    1  C   5 Note On
000094D6   9  ---    90    48    00    1  C   5 Note Off
```

## VII. **Conclusion**

To conclude, we successfully implemented one 16-note module of the system and demonstrated that our implementation more than met our latency requirements. Given that the host system receives the MIDI messages a millisecond prior to the hammer striking the strings, we clearly have plenty of extra time to perform further analysis of the sensor data to arrive at a more reliable and precise velocity extraction process.

For the future, we will have to construct additional slave boards and ensure that scaling up the system does not cause any issues with latency or produce any other unexpected behavior. Before this happens, it would be appropriate to produce a second version of the sensor PCB with the slave MCU integrated on the board to more accurately model the way the system would be implemented for production and eliminate the need to purchase PIC32 Starter Kits, which are an order of magnitude more expensive than the microcontroller itself.

Overall, this prototype demonstrates that the architectural design of the system is sound and gives us significantly better performance than existing systems that solve the same problem. With the key aspects of system identified and tested, we could expect to produce a full-scale production version fairly quickly.

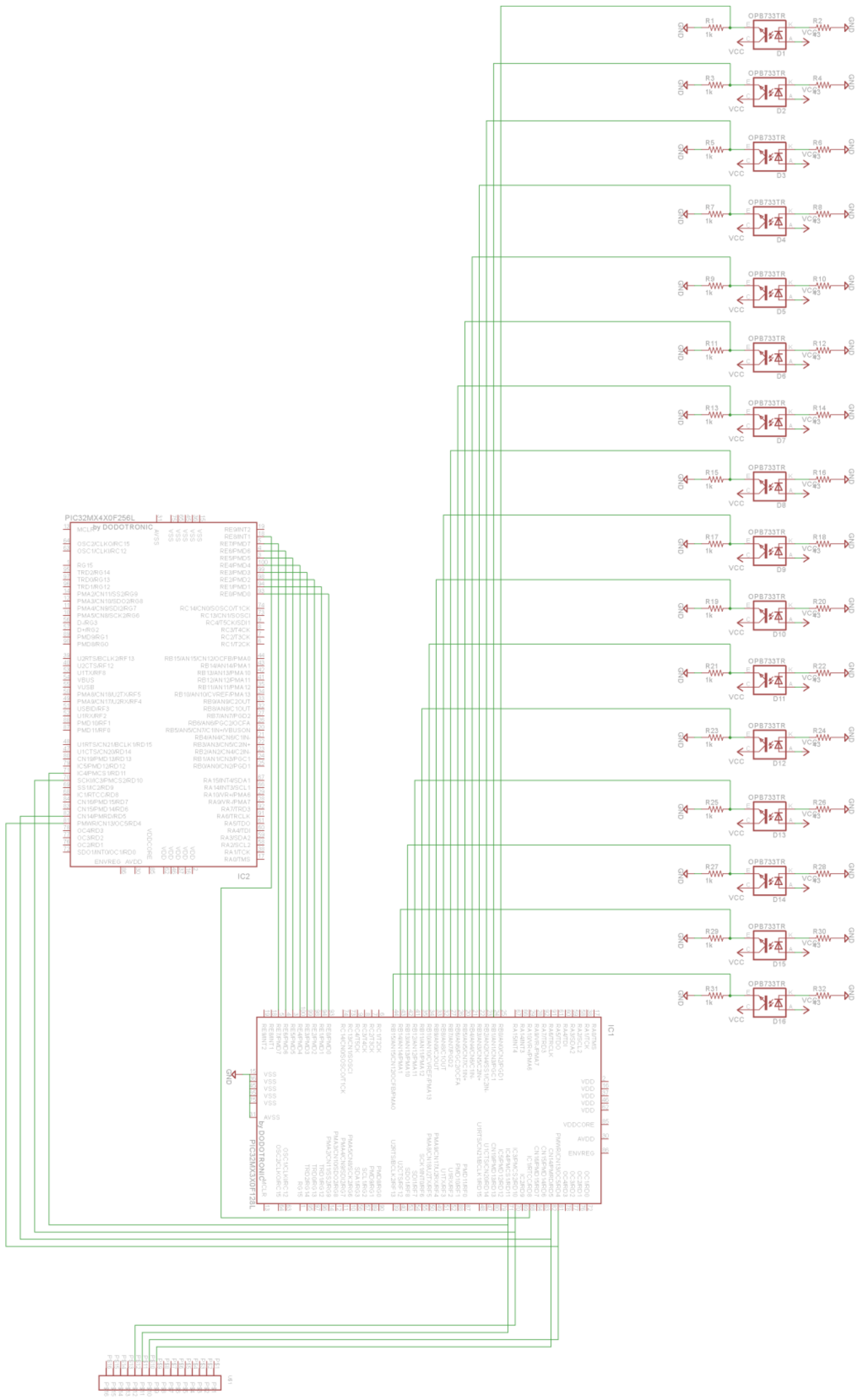
## Bibliography

"The Complete MIDI 1.0 Detailed Specification." *MIDI Manufacturers Association*. Web. 8 Nov 2012. <http://www.midi.org/techspecs/midispec.php>.

Microchip, "PIC32MX3XX/4XX Data Sheet", 2011. http://ww1.microchip.com/downloads/en/DeviceDoc/61143H.pdf

Hirschkorn, Martin. "University of Waterloo." *University of Waterloo*. Web. 8 Nov 2012. <http://real.uwaterloo.ca/Thesis-Martin_Hirschkorn.pdf>.

## Appendix A: Schematic


OPB733TR
PIC32MX4X0F256L
IC1
IC2
GND
VCC