TUFTS UNIVERSITY

Inter-IC Protocol for Low Latency Musical Applications

Steinway & Sons Research Grant

C. Hopkins, C. Powell, E. Formella

4/26/2012

Digital audio synthesis for pianos has a, undesirable level of latency and variance. This paper details the research done to identify the contribution of each subsystem to the total latency and variance. We focus on the effect of the PNOscan hardware, board protocols, MIDI conversion, and audio synthesis. Our preliminary research identified the contribution to total latency and variance from each subsystem. We then detail and explain a proposed protocol and replacement system removing the latency and improving the theoretical throughput of the system. After analyzing our prototype, we identify specifications for a final product and the changes between the current system, the prototype, and the final product that make the improvements for this system.

Contents

Overview

Digital audio synthesis for pianos as performed in the system currently used by Steinway, implemented by QRS, has an inherent, unacceptable level of latency and variance which is noticeable to listeners. Our preliminary research identified the contribution to total latency and variance from each subsystem. Data transfer through the PNOscan hardware system underneath the keys, which is in control of velocity sensing and initial data generation, incurs the largest amount of latency. Conversion from the board protocol of the PNOscan device to MIDI messages and the synthesis of those MIDI messages through a Software Synthesizer Host (SSH), such as the Muse Receptor, incur a cumulative latency of much less than a millisecond. The PNOscan hardware subsystem inherits most of its latency from its modularity, not the velocity sensing implementation. In the current system single notes incur a standard latency, still above an acceptable threshold; however, the current implementation causes the latency for polyphony to linearly increase with the number of notes. Our proposed design removes the serialization, the root cause of this delay, and replaces it with a parallelized system operating with an increased bandwidth to remove the latency and variance currently visible. The prototype developed for this system operates almost two orders of magnitude faster than the current system with the variance reduced to a negligible amount. Due to physical limits of our prototype, this paper concludes with a proposal for a final implementation useable for a full 88 key piano.

Research

The following sections detail the results of our preliminary research on subsystem contribution to latency and variance of this Digital Audio Synthesis System. The **PNOscan Protocol** section details the latency and variance findings of the PNOscan hardware boards and internal protocols. The **Software Synthesizer Host (SSH) section** outlines the functionality and delay associated with the transformation of the PNOscan's internal protocol to audio output. The **PNOscan Optics** section details the velocity sensing process performed by the optical sensors on the PNOscan and the associated delay and variance.

For the following sections, system speed will often be referred to in notes per millisecond due to the MIDI protocol's limitation of only being able to specify notes at a resolution of one millisecond.

PNOscan Protocol

The PNOscan hardware system is divided into four identical boards that monitor and register key activity, a smaller board monitoring pedal activity, and an external module used solely for conversion from an internal protocol to either Hardware-MIDI or MIDI-over-USB. Each of the four boards is connected to the others in a daisy-chain fashion, see Figure 1. The microprocessor on each board receives information from two sources: an array of sensors, monitoring key movement, and the previous daisy-chained board.

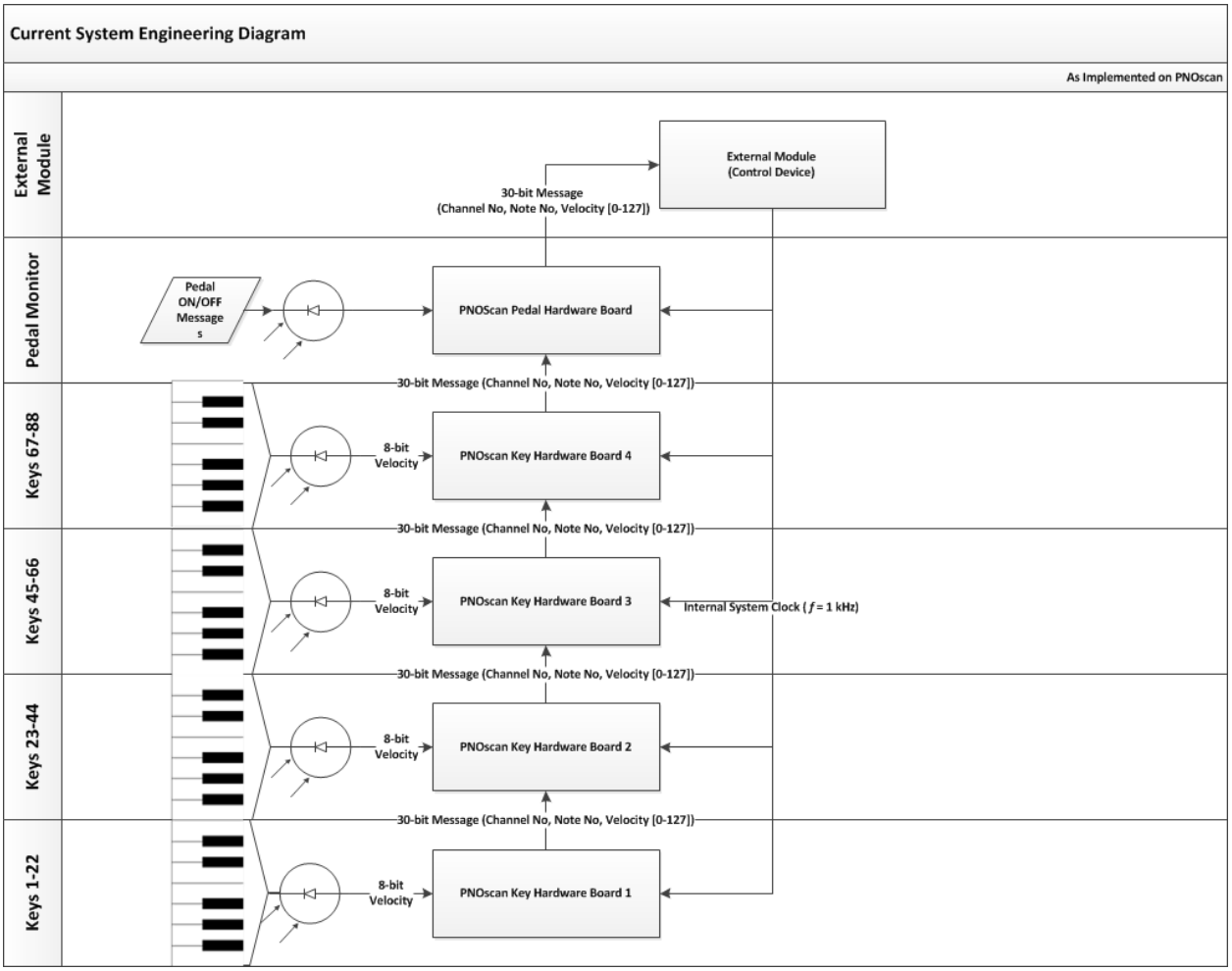

Figure 1: PNOscan Block Diagram

The device identifies a key's associated velocity when pressed, from sensor readings, and its relative note number for the board, between 0 and 21. The other board conveys a 30 bit message (duration 240μs) including an 8 bit velocity number, between 0-127, an 8 bit note number, and another 8 bit message used for channel. The duration of a message alone theoretically constricts the number of notes per millisecond to four.

Number of Notes Pressed	Minimum Delay (*ms*)	Maximum Delay (*ms*)
1	1.0	1.4
2	1.4	2.8
3	1.8	3.2

Table 1: Variance and Delay of Polyphony of Current System

An additional delay is incurred through the propagation of data through the boards. When a 30-bit message is received on a new board, the note number is incremented by 22 to keep track of the global note number to which it corresponds (A0 is globally note 88 for this protocol). The largest delay and variance come from this cascading system. **Table 1**: Variance and Delay of Polyphony of Current System shows the variance and magnitude of delay for the system. Note that we were physically unable to produce more than 3 notes in 1 millisecond.

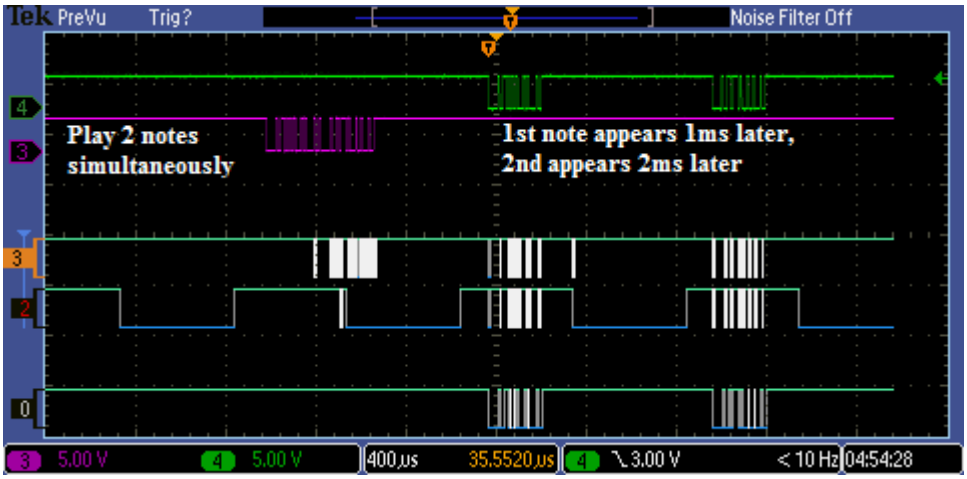

Figure 2: Line 0 (Digital) - PNOscan Output, Line 2 (Digital)- PNOscan Internal Clock, Line 3 (Digital)- Board Pre-Buffer Signal, Line 3 (Analog) - Board 4 Output Signal, Line 4 (Analog) - PNOscan Output Signal

Figure 2: Line 0 (Digital) - PNOscan Output, Line 2 (Digital)- PNOscan Internal Clock, Line 3 (Digital)- Board Pre-Buffer Signal, Line 3 (Analog) - Board 4 Output Signal, Line 4 (Analog) - PNOscan Output Signalabove, shows the waveforms resulting from two keys registering in the same millisecond. The output for the first note is registered 1.1ms after generation and for the second note 1.9ms after generation.

Software Synthesizer Host (SSH)

Two different SSHs were used for this system: the Muse Receptor 2 and the Yamaha-Motif Rack. Running with MIDI-over-USB neither contributed to the latency of the system in a meaningful way. When Hardware-MIDI is used to transmit data between the PNOscan Control Module and the SSH, the latency is increased as shown in **Figure 3**: MIDI-over-USB Latency and **Figure 4**: Hardware-MIDI Latency.

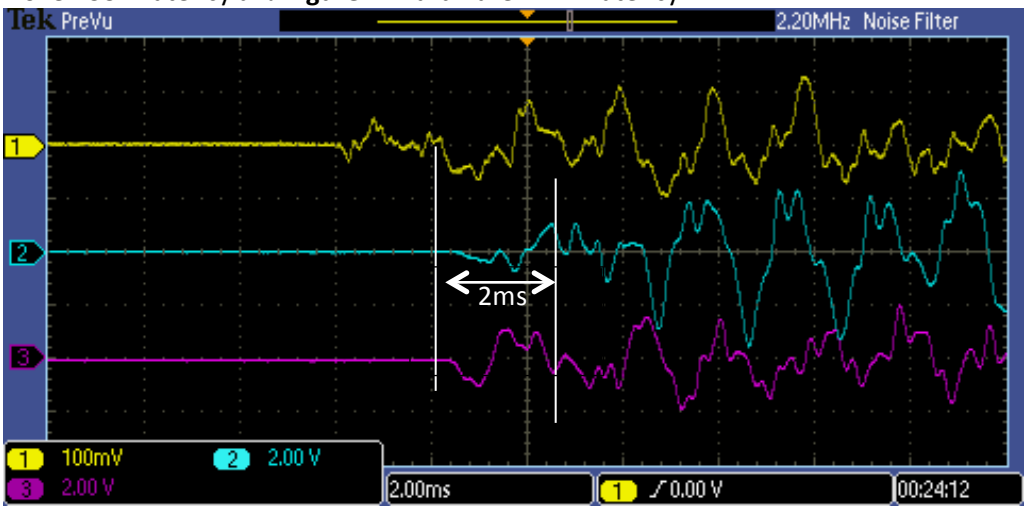

Figure 3: MIDI-over-USB Latency

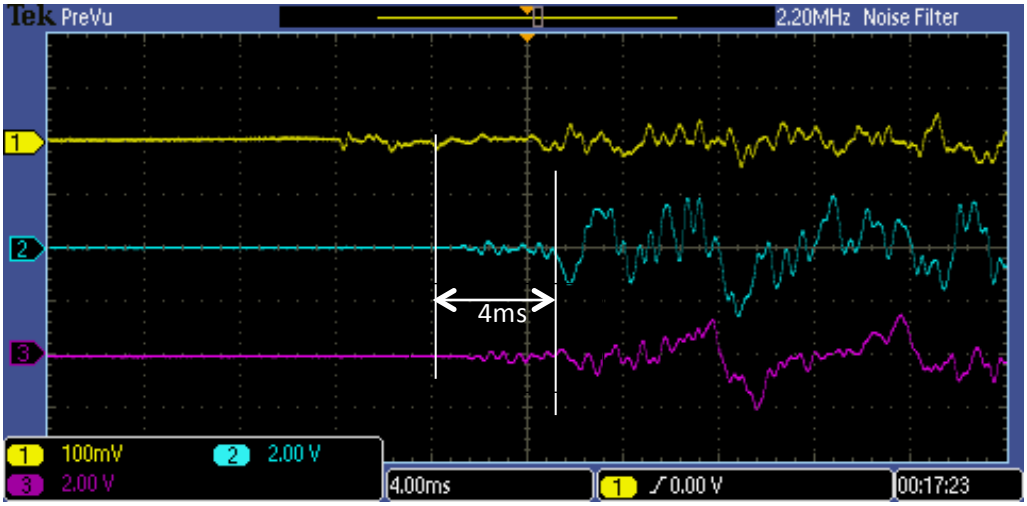

Figure 4: Hardware-MIDI Latency

This difference in latency is expected due to the limitations of Hardware-MIDI and the theoretical speeds compared to MIDI-over-USB. The limits of the transmission and their effect on the magnitude and variance of latency become even more pronounced with polyphony. Although not visible in the analog spectrum, this can be seen, in the digital spectrum, in Figure 2.

PNOscan Optics

As previously mentioned, the velocity sensing implementation performed on the PNOscan has a negligible effect on the total latency of the system. These velocities are determined by signal analysis on output of an infrared (IR) transceiver. This signal is a relative metric of distance between key and sensor. Using a linear relationship between distance and velocity, a digital signal processing (DSP) chip, shared between multiple sensors, uses the recorded curve to interpolate the key's corresponding velocity. **Figure 5**: Since a single key press can have a travel time of over 40ms, ample samples with which to calculate velocity are available and latency in velocity determination can be eliminated., below shows the point during average key movement when audio is produced by the piano.

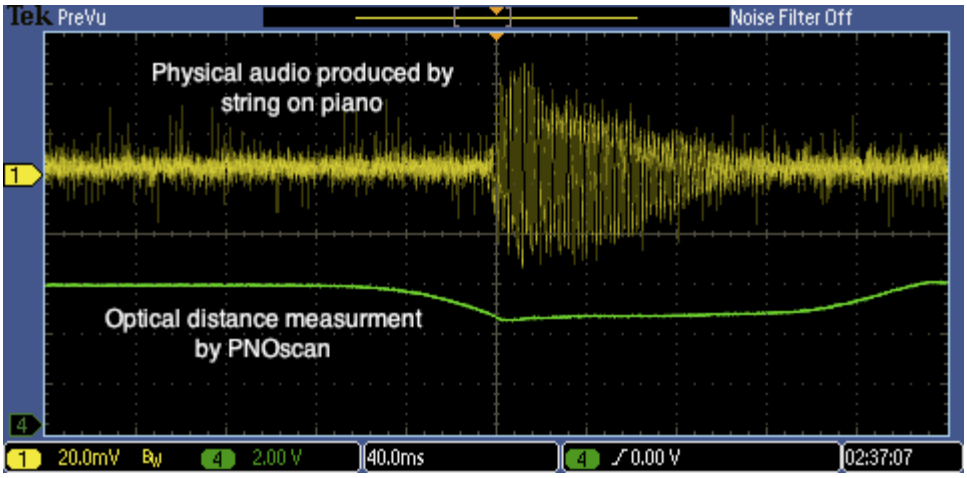

Figure 5: Since a single key press can have a travel time of over 40ms, ample samples with which to calculate velocity are available and latency in velocity determination can be eliminated.

Design

Our proposed system is specifically a reimplementation of the PNOscan hardware boards described in **PNOscan Protocol.** The SSH, running MIDI-over-USB, and PNOscan optics from the current system perform their functions within the desired latency requirements and have been preserved in our proposed system. We present here a design for the bridge between these two subsystems that keeps latency under the desired threshold in addition to increasing the transmission bandwidth by an order of magnitude.

Physical System

Our system consists of multiple slave devices communicating with a single master device. The previous system used the external control module as the master device responsible for converting and transmitting serial data to the SSH. Where the previous control module would be required to shift in 8-bits for each note number or velocity, our master reads the data in parallel off of 8 lines at an even higher baud rate to increase the bandwidth for the system.

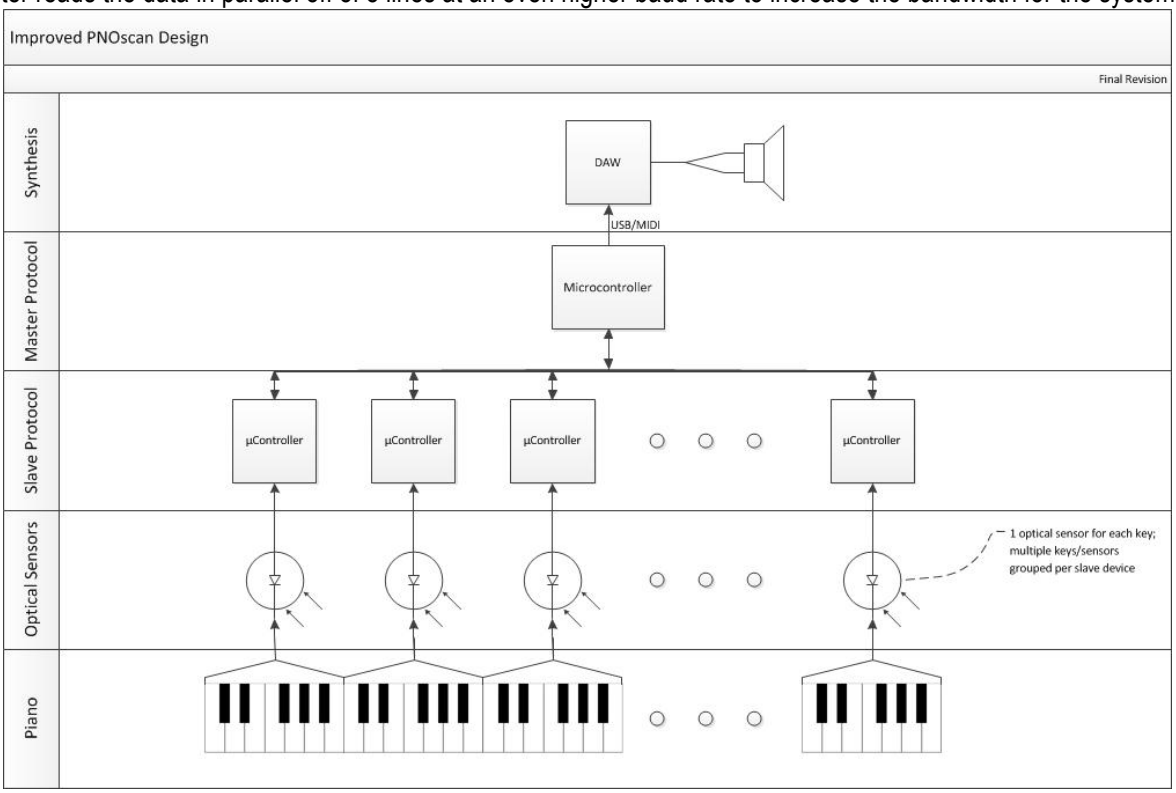

Figure 6: System Diagram for Proposed Replacement System

The slaves for our system communicate with the master over a shared bus reducing the delay caused by the daisy chaining of boards in the PNOscan system. Bus sharing adds an additional complication of interference on the bus; however, a slave select bus, shared between all slaves and asserted by the master device, operating in round robin, identifies which slave has bus write permissions. Each slave receives data representing velocity (7-bits) and corresponding note number (number of bits dependent on implementation) from the PNOscan optics subsystem. Slaves store information in a buffer queue which outputs data to the shared bus upon receiving write permission. **Figure 6** above shows the corresponding system diagram.

Protocol

This system uses an asynchronous protocol between multiple slaves and masters implementable by two Finite State Machines (FSMs), one each for slave and master. The master asserts a slave select (SS) signals, identifying which slave has write privileges, and an acknowledge signal (ACK) allowing for the asynchronous nature of the protocol. Each slave asserts a control signal (CTRL), corresponding to 4 different message types (`note-on`, `note-off`, `velocity`, `no-data`), and a data signal (DATA) which holds the data associated with the current control signal.

Master FSM

The master constantly asserts a signal on the SS bus corresponding to a specific slave. After this signal is asserted, if CTRL has changed during a predetermined, implementation-specific time interval, the master knows it has received a message. If the CTRL signal has remained in the `no-data` state throughout this interval, the master changes its SS signal to specify another slave and repeats this process.

Two CTRL signal transitions are possible during this time interval corresponding to the assertion of a `note-on` or `note-off` message. Due to the reliability of an asynchronous protocol, the master must toggle the ACK line to acknowledge a message was received. If the CTRL line is in a `note-off` state, the master waits for the CTRL line to return to a `no-data` state, changes its' SS signal and returns to the start of the FSM. If the CTRL line is in a `note-on` state, the next message is guaranteed to be a velocity message. The master waits for CTRL to change to the velocity state, and toggles its ACK signal. The master then proceeds to wait for the no data CTRL signal, changes its SS when the change is observed, and returns to the beginning of the state diagram. This FSM, implemented for 4 slaves, is displayed below in **Figure 7** and the corresponding ASM is shown in **Figure 8**.

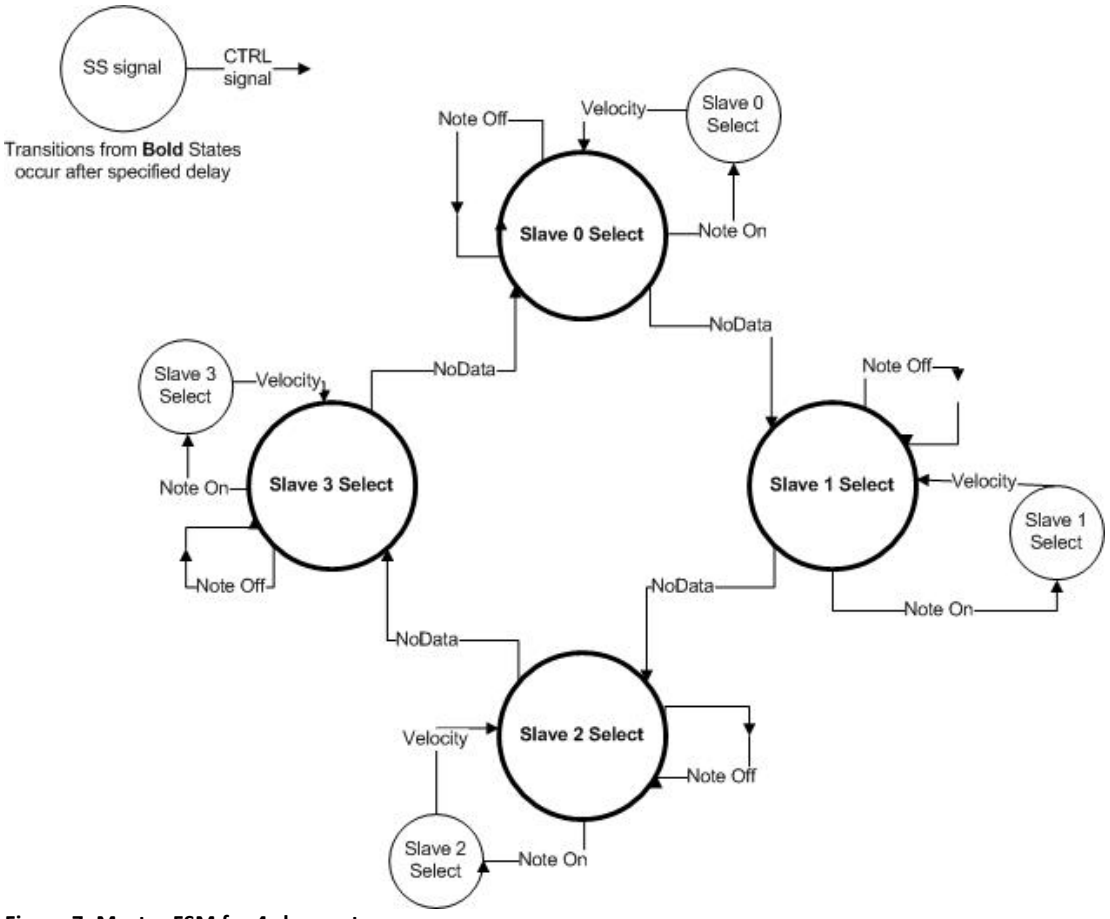

Figure 7: Master FSM for 4 slave setup

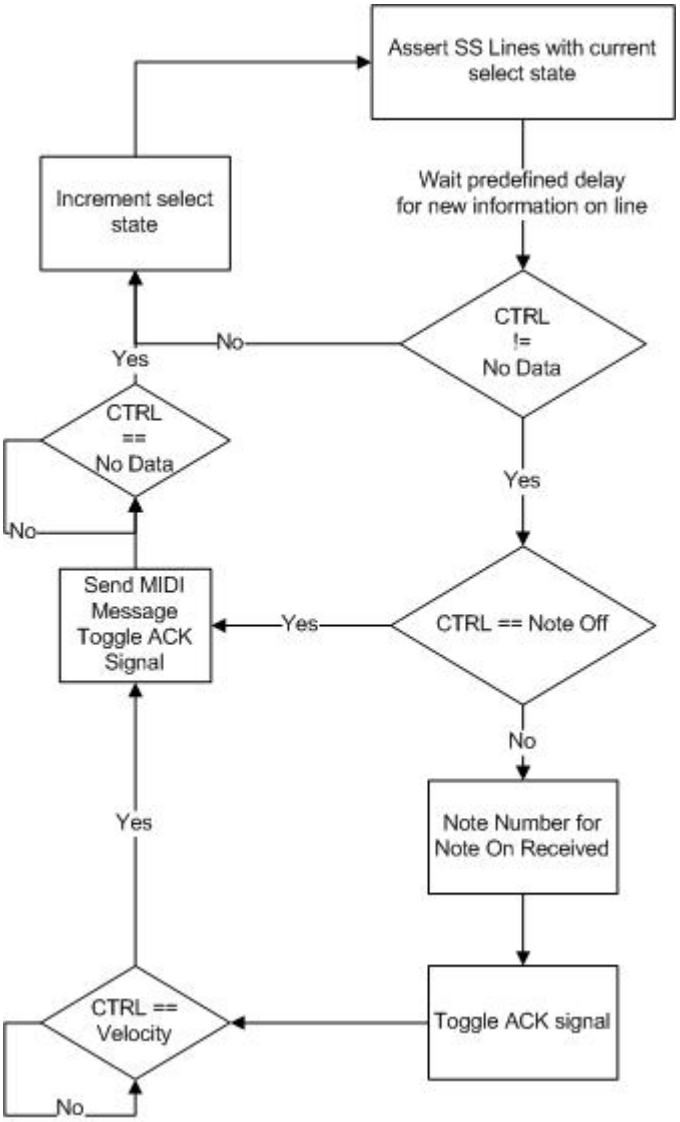

Figure 8: Master ASM for Proposed System

Slave FSM

The slaves have simpler states, involving less computational power and I/O bandwidth than the master. A specific slave only asserts data when it receives its respective SS signal; each slave is assigned a unique ID corresponding to the write enable state of the SS signal. During a select cycle, a slave only sends one message; either in the form of a note number and velocity, in the case of a `note-on` message, or solely a note number, for a `note-off` message. After sending the note number or velocity the slave waits for the ACK signal to toggle before asserting the next piece of data. After the full `note-on` or `note-off` message has been sent to and acknowledged by the master, the slave asserts the `no-data` CTRL signal and data message to allow the next slave to write proper information. The ASM in **Figure 9** illustrates this process.

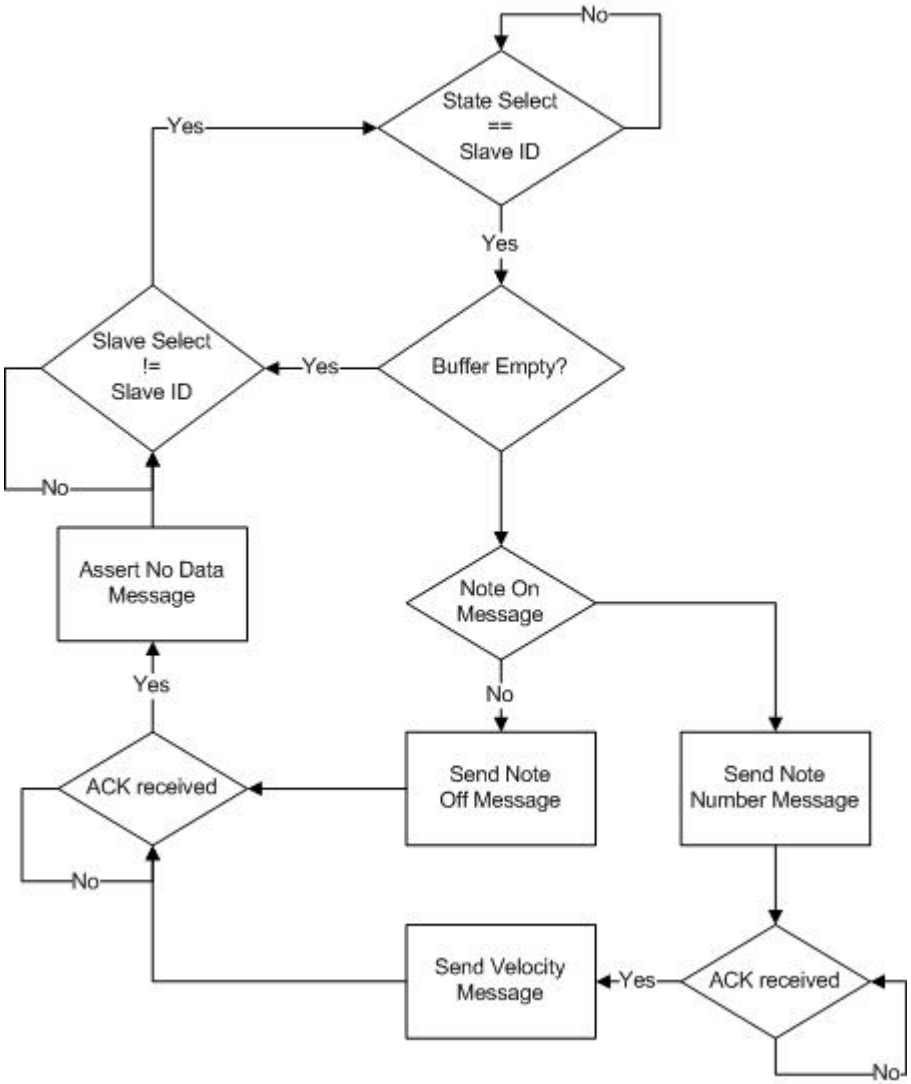

Figure 9: ASM for Slave Device

Implementation

Our prototype implements our replacement for the PNOscan Hardware sub-system. For demonstration purposes, we primarily worked with readily available, off-the-shelf components with which we could efficiently develop and test. These results can be seen in the **Prototype** section. Our prototype is limited by the IO and processing capabilities of the hardware used; however, given more resources, an even greater speedup is possible and is described in the Error! Reference source not found. section.

Prototype

Our prototype was implemented using a set of Arduino Uno boards as the devices for the master and slave boards. The Arduino Uno utilizes the ATmega328 chip by Atmel, Inc. This chip has a maximum operating frequency of 20 MHz and 23 I/O pins. However, not all of these pins are available through the Arduino board, which forced us to make certain concessions in our design. In a proprietary design, one could simply use the Atmel chip itself to gain

access to each pin, which would double the speed of communication when there is note information to be transmitted.

Below is a list indicating which pins on the Arduino have been used and for which purpose(s).Below, MISO stands for master input, slave output; MOSI stands for master output, slave input:

- PORT C [5:0] (Analog pins 5:0) – Optical input on the slave only
- PORT B [7:0] (Digital pins 13:8) – DATA: Note number/velocity data (MISO)
- PORT D [7:6] (Digital pins 7:6) – CTRL: [Velocity or note number]:[note on or off] (MISO)
- PORT D [5] (Digital pin 5) – ACK: Acknowledgement bit (MOSI)
- PORT D [1:0] (Digital pins 1:0) – SS: Slave select up to 4 slaves (MOSI)

The pin mappings above allow note number and velocity to assume a value between 0 and 255, corresponding to the 8-bit width of PORT B. Our prototype also uses up to 6 bits to specify a note number; however, due to I/O limitations of the board used, only 6 distinct inputs are available directly into the slave device, limiting the throughput of input to the slave devices. As limited by the bit-width of the SS bus, we have implemented the system using up to 4 slaves. Additionally, the final product outputs MIDI-over-USB; however, the Arduino Uno board communicates to its USB module over a slow serial protocol and therefore cannot take advantage of the MIDI-over-USB speeds needed for this product. For this reason, our prototype does not have MIDI output functionality implemented. Instead it displays the reduction of latency in transferring data between the hardware boards which - as discussed in the Research section - is the largest bottleneck in the system.

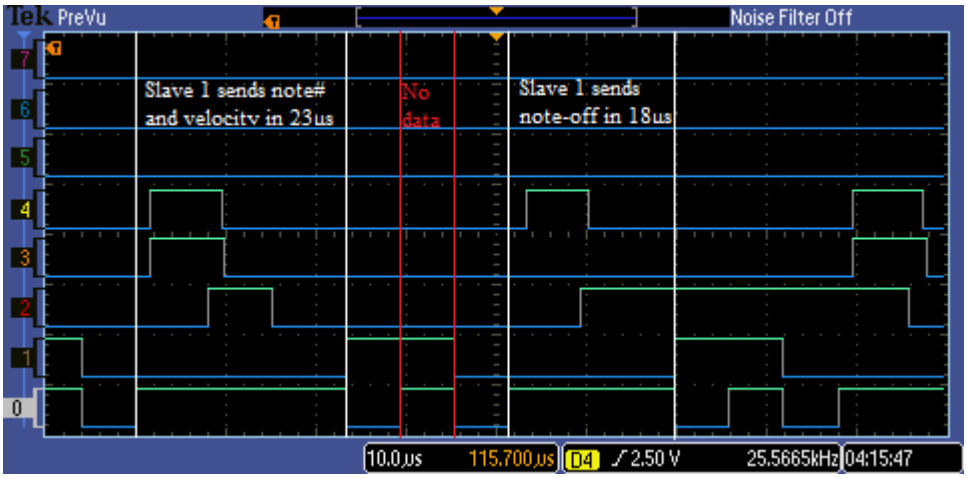

Figure 10: Lines [0:1] - Slave Select, Line [2] - ACK, Lines [3:4] - Control Signal (message type)

Because our prototype is meant to demonstrate the proposed protocol, our system is setup for 4 slaves; however, only a single slave (Slave_ID == 1) is enabled to transmit data, the other two slaves' write cycles are visible in the SS bus but send no data. When the master polls a specific slave, that slave will send any information it has in a maximum of 23µs. `Note-on` messages take this maximum time due to their requirements to send `note-on` and `velocity`. In the case of a `note-off` message, this only requires 17µs since no velocity data is required. Lastly, if there is no information to send, the slave requires ~7µs to communicate this. These latencies mean a maximum of 43 `note-on` MIDI messages per millisecond or an amortized 50 messages per millisecond maximum, computed through an average `note-on/note-off` combination.

Conclusions

Our hardware proposal for the final system removes the constraints imposed by the hardware for our prototype. Additionally, our proposal removes the serialization of note number and velocity during transmission of a `note-on` message by giving note number and velocity their own dedicated busses.

A normal piano has 88 keys (7 octaves from A0 to C8 plus a third), which are monitored by 4 slaves, meaning 2 bits of information. This specification means each slave device monitors 22 keys for messages. The PNOscan optics from the original system already monitors this number of keys allowing for easy integration into the rest of the proposed system. This means that each board will generate note numbers between 0-22 which can be conveyed over 5 bits of data. 22 ports are needed to monitor the input of the 22 keys. The shared velocity bus can be implemented with a 7-bit bus to convey data between 0-127. The control signal is implemented using 2 bits. ACK is implemented using 1 bit. This specification requires 39 ports on each slave.

Utilizing a chip, such as the AT32UC3A0128 by Atmel, with a maximum frequency of 66 MHz its increased number of I/O pins, operating at a faster switching speed, gives us a speedup of 2-3 times that of our prototype. This gives us a final data transmission rate of approximately **300 MIDI messages per millisecond**. This speed is faster than both the theoretical limit of MIDI-over-USB and the physical capabilities of a pianist.

Bibliography

1] P. Lehrman and T. Tully, "What is MIDI?," in *MIDI for the Professional*, Music Sales America, 1993, p. 255.

2] G. Ashour, B. Brackenridge, O. Tirosh, M. Kent and G. Knapen, "Universal Serial Bus Device Class Definition for MIDI Devices," 1999.

3] Arduino, "Reference," 26 November 2011. [Online]. Available: http://arduino.cc/en/Reference/HomePage. [Accessed January 2012].

4] Arduino, "PortManipulation," 17 January 2010. [Online]. Available: http://www.arduino.cc/en/Reference/PortManipulation. [Accessed February 2012].

5] Tektronix, "Mixed Signal Oscilloscopes," 19 March 2012. [Online]. Available: http://www.tek.com/datasheet/mixed-signal-oscilloscopes-9. [Accessed January 2012].

6] Atmel Corporation, "ATmega328," 2012. [Online]. Available: http://www.atmel.com/devices/atmega328.aspx. [Accessed March 2012].

7] Atmel Corporation, "AT32UC3A0128," 2012. [Online]. Available: http://www.atmel.com/devices/at32uc3a0128.aspx. [Accessed March 2012].

8] Arduino, "ArduinoBoardUno," 20 February 2012. [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardUno. [Accessed January 2012].

9] Tektronix, "AFG3000 Function Generator," 2012. [Online]. Available: http://www.tek.com/signal-generator/afg3000-function-generator. [Accessed January 2012].

10] QRS Music Technologies, Inc., *PNOscan II User Guide,* Rev. 1.00 ed., vol. Manual #OM79217.