HIDUINO: A firmware for building driverless USB-MIDI devices using the Arduino microcontroller

Dimitri Diakopoulos[1]
California Institute of the Arts[1]
24700 McBean Parkway
Valencia, California 91355
ddiakopoulos@alum.calarts.edu

Ajay Kapur[1, 2]
New Zealand School of Music[2]
P.O. Box 2332
Wellington, New Zealand
akapur@calarts.edu

ABSTRACT
This paper presents a series of open-source firmwares for the latest iteration of the popular Arduino microcontroller platform. A portmanteau of Human Interface Device and Arduino, the HIDUINO project tackles a major problem in designing NIMEs: easily and reliably communicating with a host computer using standard MIDI over USB. HIDUINO was developed in conjunction with a class at the California Institute of the Arts intended to teach introductory-level human-computer and human-robot interaction within the context of musical controllers. We describe our frustration with existing microcontroller platforms and our experiences using the new firmware to facilitate the development and prototyping of new music controllers.

Keywords
Arduino, USB, HID, MIDI, HCI, controllers, microcontrollers

1. INTRODUCTION
A core goal of the Music Technology program at the California Institute of the Arts is to teach students how to connect the physical and virtual world. *Interface Design for Music and Media Applications* is a yearlong class that introduces students to artistic interactivity through microcontrollers and sensors/actuators. Modeled after the template presented at CCRMA by Bill Verplank *et al* in 2001, "A Course on Controllers [11]," and Gideon D'Arcangelo's course at ITP [3], the class required a modern microcontroller platform that was neither too high-level nor too complex.

The Arduino turned out to be the ideal solution for the class, although there was a major usability issue which we felt restricted its potential as the core of a musical controller, described later in section 2.2. Combined with a redesign of the Arduino platform in 2010 and an open-source USB communication library for Atmel AVR microcontrollers (on which the Arduino is based), we were able to develop a firmware permitting driverless USB-MIDI communication between Arduino and host computer, a solution that resolved our greatest usability concern with the Arduino.

In section 2, we present our reasons for switching to the Arduino, some other platforms aimed at artistic interactivity, and frustrations with both. Section 3 details the nuts and bolts of the HIDUINO firmware. Section 4 explains the process of prototyping a controller using HIDUINO. We conclude in Section 5 with several ideas about future improvements.

NIME'11, 30 May–1 June 2010, Oslo, Norway.

2. BACKGROUND
The third iteration of the *Interface Design* class in 2009 was the first to switch to the Arduino[1]. It was during this time we became acutely aware of the power of the platform, but also its shortcomings, namely that serial data needed to be parsed and converted to a more useful format. While the complete history of using microcontrollers in the context of NIME is outside the scope of this paper, we provide a brief overview of some related projects that attempt to solve this protocol problem.

2.1 Why the Arduino
The class follows the basic ideas Perry Cook outlines in "Designing Principles for Computer Music Controllers [1, 2]," as well several requirements described by Nicola Orio *et al* in [5], including *learnability, exportability,* and *feature compatibility.* Our selection of a suitable microcontroller for the class held these concepts in mind, applying them not only to the properties *of* a musical controller but also to the elements core to their design. Based on the research presented by Scott Wilson *et al* in [12], the results of using Atmel's AVR microcontroller appeared to meet many of these design criteria.

Since the publication of that paper, many AVR-based microcontroller platforms have been released, including the increasingly popular Arduino. Our choice to move to this platform was motivated by the large community of support and open-source nature of many Arduino-based projects. In Alicia Gibb describes the Arduinos' growing reputation as an extensible platform for interactive media and goes on to say, "The design of the Arduino microcontroller caters to a non-technical audience by focusing on usability to achieve its intended goal as a platform for designers and artists [4]," supporting one of our core criteria of *learnability.* As the Arduino language is simply an abstracted form of C, we found that our *exportability* and *feature-compatibility* requirements were sufficiently satisfied by the ability to write and use low-level C libraries for more advanced projects.

2.2 On Protocol Confusion & Usability
Hans-Christoph Steiner's paper, "Firmata: Towards making microcontrollers act like extensions of the computer [9]," reveals a general problem in existing microcontroller platforms: protocols for communication. Arduino, and other similar platforms like Wiring[2] and Gainer[3], implement Virtual COM ports via USB for basic serial I/O between microcontroller and host. Since no major music application outside of Max/MSP[4] supports reading serial directly, there is a significant disconnect between controller and application.

[1] http://www.arduino.cc/
[2] http://wiring.org.co/
[3] http://gainer.cc/
[4] http://cycling74.com/

For us, this single limitation created a large usability gap in the platform which turned into the primary motivating factor behind HIDUINO.

It was necessary in the 2009 CalArts *Interface Design* class that each new musical controller required a new Arduino sketch and accompanying 'decoder' software to interpret the raw serial data and convert to music-friendly MIDI or OSC. MIDI was particularly problematic since it required the use of virtual MIDI loopback drivers (or proprietary loopback software, in the case of Windows). Considerable time was added to the development of controllers on account of the need for this middleware software. Additionally, it created a single point of failure and added additional latency between performer and application. Figure 1 illustrates an overview of a controller using this complicated method.

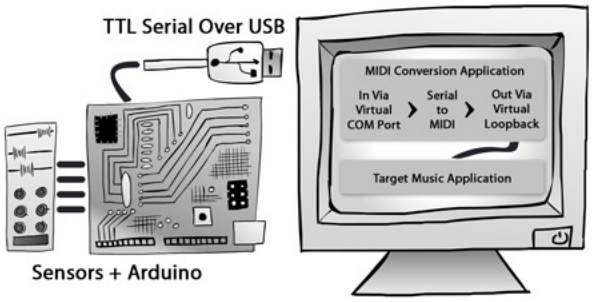

Figure 1. Controller Development Pre-HIDUINO

Our initial attempts at simplifying the process led us to look to preexisting solutions in the Arduino community. Sidestepping the need for serial and separate software, we sound several potential solutions in the form of external hardware add-ons known to the Arduino community as *shields*. We evaluated two shields, one for MIDI[5], and another for Ethernet[6].

In the case of the MIDI shield, we noted that the host computer still needed a separate MIDI interface for the older-style 5-pin DIN connection and was additionally limited by all the typical constraints of MIDI. The Ethernet shield we tested was used in conjunction with a simple OSC implementation. This combination added additional cost and extra cabling: USB is required for power, Ethernet for data. Moreover, many students wishing to use their controller with commercial software still needed an OSC to MIDI conversion app.

2.3 USB HID & the CUI

The vast majority of commercial MIDI controllers on the market implement a protocol known as USB-HID[7]. This protocol is often viewed negatively by developers, citing its implementation complexity and bloat [9]. On account of these difficulties, few have been able to implement the protocol in a working form suitable for musical controller development. However, one of the more recent and successful projects using USB-HID is the Create USB Controller (CUI) developed by Dan Overholt at UC Santa Barbara [6].

Prior to the 2009 *Interface Design* class, the CUI was the main controller platform on account of its native USB-HID support. Built on top of a Microchip PIC[8] (a competitor to the Atmel AVR), the CUI comes bundled with a generic MIDI-HID firmware which sends out 12-channels of pitch-bend data, one for each ADC pin present on the board.

In previous years, students in the class noted a few frustrations with the CUI, claiming the propriety development chain and necessity of low-level C required by Microchip complicated the process of making changes to the firmware. We also investigated the use of the micro-OSC firmware on the CUI, uOSC [7, 8]. The uOSC firmware also required separate serial to OSC software on the host. In short, it did not meet our expectation of usability.

Figure 2 illustrates the benefits of the USB-HID protocol by simplifying the number of steps present in Figure 1 and demonstrates our ideal model for a musical controller development platform.

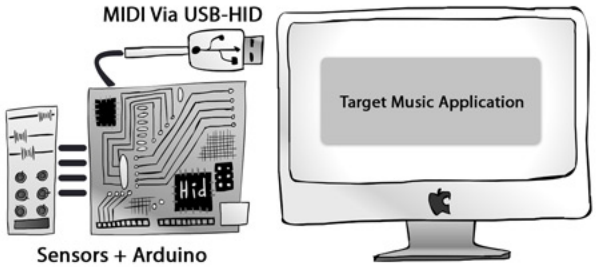

Figure 2. Simplification of protocols and software by using HID.

2.4 A New Firmware

Underling several ideas presented by Owen Vallis *et al* in his 2010 NIME presentation, "A Shift toward Iterative and Open-Source Design for Musical Interfaces [10]," the concept of HIDUINO aimed to directly attack our usability issues by removing the protocol confusion. Driven by a desire to see the Arduino act as a true USB-HID device, the primary goal of HIDUINO was to dismiss the need for custom software and remove the Arduino's dependence on the serial protocol. A secondary goal was to develop a feature-complete framework to meet the needs of both prototype and performance-ready controllers.

3. IMPLEMENTATION

Implementation of HIDUINO was aided by two recent events within the Arduino community: a redesign of the Arduino microcontroller and the support of an existing USB-HID library by the Arduino team.

3.1 2010 Arduino Redesign

The 2010 revision to the Arduino platform introduces the UNO and the Mega2560, using the Atmel ATmega328 and the ATmega2560 chips respectively. Earlier revisions were equipped with an FTDI chip permitting users to interface with the Arduino via USB. The FTDI chip presented a few challenges to familiar users, namely that it required propriety drivers on all platforms and could *only* act as a virtual serial port. The 2010 redesign omitted this chip in favor of the ATmega 8U2, the tiniest chip in Atmel's lineup that included native support for USB. The new Arduino includes a pre-loaded firmware which emulates the functionality of the older FTDI chip, but more importantly exposes the pins necessary to re-flash the 8U2 with the users' own custom firmware. This change opened up the possibility of writing a firmware that could conform to the USB-HID protocol specification and still communicate with the primary microcontroller on the Arduino. Without this redesign, the HIDUNO project would have required the production of a new shield that incorporated the 8U2 or similar ATmega USB chip.

[5] http://www.sparkfun.com/products/9595
[6] http://www.arduino.cc/en/Main/ArduinoEthernetShield
[7] http://www.usb.org/developers/hidpage/
[8] http://www.microchip.com/

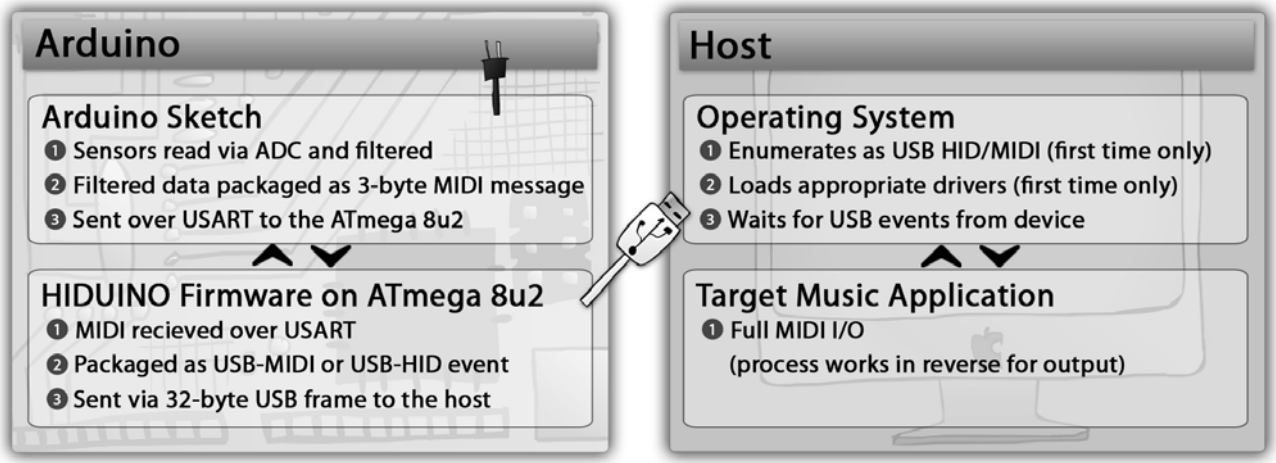

Figure 3. High-level architecture of a controller using HIDUINO

3.2 LUFA Library

In 2008 Dean Camera started the MyUSB library, an open-source Human Interface Device (HID) library for the USB-compatible line of AVR microcontrollers. Later renamed LUFA (Lightweight USB Framework for AVRs)[9], the project set out to create an elegantly-written library to demystify the USB-HID protocol. In contrast with a number of libraries written for the same purpose (detailed in the LUFA documentation)[10], the API is straightforward and not restricted to any specific AVR USB microcontroller. In addition, the library comes preloaded with descriptors for generic HID devices, including MIDI. Descriptors, as a core part of the USB protocol, instruct which drivers a host computer should use to interface with the device. Most operating systems provide built-in drivers for these USB *class-compliant* devices.

3.3 The Firmware(s)

Initial firmware programming took place late in 2010 shortly after the new Arduino designs shipped. The LUFA library already provides appropriate descriptors for USB-MIDI, thus the majority of HIDUINO code is targeted at structuring the communication between the main ATmega chip and the new 8U2. Figure 3 illustrates a full-system overview of a controller using the HIDUINO, showing the primary functions of the firmware.

The main Arduino ATmega and 8U2 chip communicate over shared transmit/receive USART[11] pins. Although the structure of the serial sent from the main chip does not matter on account of its later re-packaging as 32-byte USB-MIDI event, we decided implement a standard 3-byte MIDI protocol to standardize and simplify sending data between the chips. Once the HIDUINO firmware receives a complete 3-byte message, it is checked for validity, placed into a USB-MIDI event container, and finally pushed over USB. After developing MIDI-out, all communication functions were re-written in reverse for MIDI-in.

While we emphasize development of USB-MIDI in this publication, other HIDUINO firmwares are currently being released that give an Arduino the power to act like other common HID devices, including mice, keyboards, joysticks, game controllers, and even audio devices. Several music programming languages and frameworks currently support reading HID, though these are aimed primarily at repurposing commercial USB devices as musical controllers [13].

4. PROTOTYPING

The process of building a controller implementing HIDUINO is designed to be as straightforward as possible. In the context of the *Interface Design* class, the process of prototyping a new controller can be broken down into four steps:

1. Design
2. Signal Conditioning
3. Transitioning to HIDUINO
4. Testing

4.1 Design

The design phase exists to consider aspects of overall form, function, and effectiveness as musical controller. Students are encouraged to test and experiment with various sensors including potentiometers, soft potentiometers, force sensing resistors, buttons, accelerometers and gyros, photocells, resistive touch surfaces, proximity sensors, hall-effect sensors, and flex sensors.

4.2 Signal Conditioning

All sensors connected to the Arduino require some level of conditioning and scaling before being sent to the host. For example, accelerometers are often low passed and soft-pots are read using sample-and-hold logic. Data during this step is sent to the host using serial so it can be displayed in the serial monitor in the Arduino IDE. As a final step, sensor data is clamped to MIDI-friendly 0-127 resolution.

4.3 Transitioning to HIDUINO

After a user is content with the sensor data, the Arduino sketch is ready to implement MIDI. Using a simple Arduino library for reading and writing MIDI over serial, the data can be packaged as a note on, continuous control, or pitch bend message.

4.3.1 Flashing the firmware

The flashing process can be accomplished one of two ways depending on whether a user has access to an in system programmer (ISP). Both ATmega chips on the redesigned Arduino have exposed in circuit serial programming (ICSP) headers. In our own testing and within the class, an Atmel AVR-ISP MKII was utilized to flash firmwares directly onto the 8U2. A second option is through the use of a bootloader.

[9] http://code.google.com/p/lufa-lib/
[10] http://www.fourwalledcubicle.com/files/LUFA/Doc/101122
[11] http://arduino.cc/en/Reference/serial

The DFU bootloader[12] written by Atmel can piggyback on top of any firmware granted there is enough free flash memory. When certain pins on the exposed 8U2 headers are tripped, the chip will enter bootloader mode and then can be programmed via USB. All HIDUINO firmwares are currently built with the DFU bootloader so both methods are available. Section 5 provides a link to the HIDUINO project page which presents this entire process in greater depth (including tools and software used) in tutorial format.

In the first iteration of HIDUINO, host communication with the primary ATmega328 chip needed the virtual-serial port firmware and thus complicated the process of updating Arduino sketches as most users need to continuously switch between virtual-serial and HIDUINO. A recent build of the software combines the virtual-serial and HIDUINO firmwares into a single package so a user is able to select which firmware is loaded based on header pin configuration. This build requires the use of an ISP programmer to initially load the firmware as the combined version could not be combined with the DFU bootloader in the 8kB of space available on the 8U2.

4.4 Testing

The testing phase ensures that each sensor is correctly scaled and addressed by the right MIDI identifier.

5. SUMMARY AND FUTURE WORK

The HIDUINO project represents a significant step forward for students, musicians, and artists who desire their own custom controller. With HIDUINO, the Arduino now has the potential to become a powerful base for driverless, cross platform MIDI controllers and other HID devices. With these firmwares, DIY controllers can compete with the plug-and-play usability of commercial offerings while maintaining our core values of learnability, modularity, and flexibility for teaching and prototyping.

With respect to future work, considerable ongoing effort is being applied toward extending the quality and number of HID firmwares, including device types for joysticks, game controllers, mice, and keyboards. One of the largest student complaints about the MIDI firmware is that it is still restricted to 127[13] steps of resolution. We are investigating the possibility of implementing a part of the USB specification called CDC-ECM[14] – Ethernet Control Model. Although not currently a part of the LUFA library, CDC-ECM would allow the possibility of native Ethernet-over-USB functionality permitting the use of high-resolution protocols like OSC.

The HIDUINO project page is located online at http://mtiid.calarts.edu/research/hiduino and includes a tutorial-style guide to alter and compile the firmwares from scratch. The current SVN code repository is located on GoogleCode at http://code.google.com/p/hiduino/.

6. ACKNOWLEDGMENTS

This project would have been very difficult without the Lightweight USB Framework for AVRs (LUFA) written by Dean Camera. The authors would also like to thank Martijn Zwartjes, Jim Murphy, and the entire Arduino team and community. Special thanks to Tahnee Gehm for illustrating Figures 1, 2 and 3.

7. REFERENCES

[1] Cook, P.R. "Principles for Designing Computer Music Controllers," in *ACM SIGCHI New Interfaces for Musical Expression (NIME) Workshop* Seattle, WA, 2001.

[2] Cook, P.R. "Re-Designing Principles for Computer Music Controllers: a Case Study of SqueezeVox Maggie," in *New Interfaces for Musical Expression (NIME)* Pittsburgh, PA, 2009.

[3] D'Arcangelo, G. "Creating a Context for Musical Innovation: A NIME Curriculum," in *New Interfaces for Musical Expression (NIME)* Dublin, Ireland, 2003.

[4] Gibb, A.M. *New Media Art, Design, and the Arduino Microcontroller: A Malleable Tool.* Master's Thesis, Pratt Institute, New York, NY, 2010.

[5] Orio, N., Schnell, N., and Wanderley, M.M. "Input Devices for Musical Expression: Borrowing Tools from HCI," in *Computer Music Journal*, 26(3), MIT Press, 2002.

[6] Overholt, D. "Musical Interaction Design with the CREATE USB Interface: Teaching HCI with CUIs instead of GUIs," in *International Computer Music Conference (ICMC)* New Orleans, LA, 2006.

[7] Schmeder, A. and Freed, A. "A Low-level Embedded Service Architecture for Rapid DIY Design of Real-time Musical Instruments," in *New Interfaces for Musical Expression (NIME)* Pittsburgh, PA, 2009.

[8] Schmeder, A. and Freed, A. "uOSC: The Open Sound Control Reference Platform for Embedded Devices," in *New Interfaces for Musical Expression* (NIME) Genova, Italy, 2008.

[9] Steiner, H.C. "Firmata: Towards making microcontrollers act like extensions of the computer," in *New Interfaces for Musical Expression (NIME)* Pittsburgh, PA, 2009.

[10] Vallis, O., Hochenbaum, J., and Kapur, A. "A Shift Towards Iterative and Open-Source Design for Musical Interfaces", in *New Interfaces for Musical Expression (NIME)* Sydney, Austalia, 2010.

[11] Verplank, B., Sapp, C., and Mathews, M. "A Course on Controllers," in *ACM SIGCHI New Interfaces for Musical Expression (NIME) Workshop* Seattle, WA, 2001.

[12] Wilson, S., et al. "Microcontrollers in Music HCI Instruction", in *New Interfaces for Musical Expression (NIME)* Montreal, Canada, 2003.

[12] DFU bootloader datasheet: http://www.atmel.com/dyn/resources/prod_documents/doc7618.pdf

[13] Assuming MIDI pitch bend is not used

[14] CDC-ECM Specification: http://www.usb.org/developers/devclass_docs/CDC_EEM10.pdf