A System for Sketching in Hardware

Do-It-Yourself Interfaces for Sound and Music Computing

Overholt, Daniel

Published in:
9TH SOUND AND MUSIC COMPUTING CONFERENCE

Publication date:
2012

Document Version
Accepted author manuscript, peer reviewed version

A System for Sketching in Hardware: Do-It-Yourself Interfaces for Sound and Music Computing

Dan Overholt
Department of Architecture, Design
and Media Technology
Aalborg University, Denmark
Niels Jernes Vej 14, 3-107
dano@create.aau.dk

ABSTRACT

A system for Do-It-Yourself (DIY) interface designs focused on sound and music computing has been developed. The system is based on the Create USB Interface (CUI), which is an open source microcontroller prototyping board together with the GROVE system of interchangeable transducers. Together, these provide a malleable and fluid prototyping process of 'Sketching in Hardware' for both music and non-music interaction design ideas. The most recent version of the board is the CUI32Stem, which is designed specifically to work hand-in-hand with the GROVE elements produced by Seeed Studio, Inc. GROVE includes a growing collection of open source sensors and actuators that utilize simple 4-wire cables to connect to the CUI32Stem. The CUI32Stem itself utilizes a high-performance Microchip® PIC32 microcontroller, allowing a wide range of programmable interactions. The development of this system and its use in sound and music interaction design is described. Typical use scenarios for the system may pair the CUI32Stem with a smartphone, a normal computer, and one or more GROVE elements via wired or wireless connections.

Keywords

Music Interaction Design, Sound and Music Computing education, Microcontroller, Arduino language, StickOS BASIC, Open Sound Control, Microchip PIC32, Wireless, Zigflea, Wifi, 802.11, Bluetooth, CUI32, CUI32Stem

1. INTRODUCTION & BACKGROUND

The CUI32Stem follows in the footsteps of the author's previous circuit board designs, such as the original Create USB Interface (CUI) [8] that utilized an older (8-bit) PIC18F4553 microcontroller and the currently available CUI32 [1]. The CUI32Stem uses a modern 32-bit MCU (Microcontroller Unit) running at 80MHz. This allows it to perform much faster than the original CUI. In addition, the use of a free RTOS (Real-Time Operating System) makes the system easier to use.

The StickOS RTOS [14] was created by Rich Testardi. It includes an on-chip compiler for simple BASIC-language programs, providing ease of use for beginners or even advanced users who are interested in quick prototyping with the system. In addition, the CUI32Stem includes an Arduino-compatible bootloader, so that mid-level users are able to compile Arduino "sketches" for the CUI32Stem (of which many examples are available online). Finally, advanced users are free to compile C-language programs for the CUI32Stem using Microchip's free development environment and compiler (MPLAB X / C32). All of these programming methods are possible without having to purchase a separate programmer device, thanks to the multi-platform bootloader that is pre-installed.

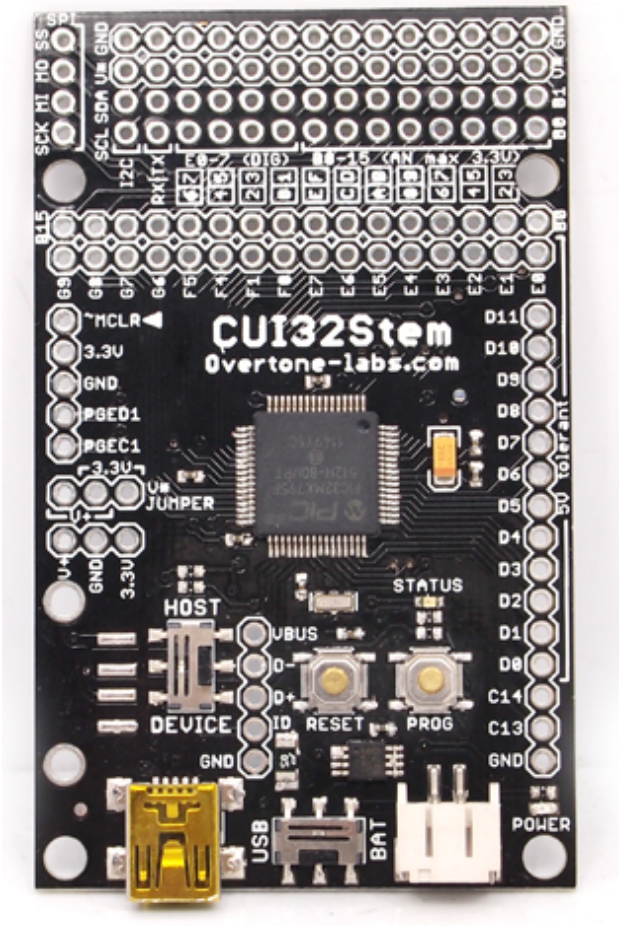

Figure 1. The CUI32Stem. 4-pin headers for SeeedStudio's GROVE system will be placed at the top, where sensors and actuators can be attached with no need for soldering (header pins are not yet installed on this board)

2. THE CUI32Stem

As shown in figure 1 above, the CUI32Stem functions primarily as a simple breakout board for the Microchip® PIC32 processor – a PIC32MX795F512H device. The ports are all conveniently labeled on the board (both top and bottom sides), saving users the hassle of referring to the datasheet for the device in most cases.

The 4-pin headers at the top of the board are designed to allow the GROVE elements (sensor and actuators) to be easily attached. The top left set of 4-pins are for the Serial Peripheral

Interface bus (SPI), while the 4-pins to the right of are for the Inter-Integrated Circuit (I^2C) bus. These are both digital bus standards that many MCUs use of in order to send and receive data with attached sensors and actuators. Next is a serial port (labeled RX / TX for Receive and Transmit pins), followed by pins E0-E7, which are for digital input or output. They are laid out in pairs, together with access to V* (which is selectable as either 3.3v or 5v, depending on the position of the jumper placed lower on the board's left side) and ground (GND). Finally, there are 16 analog input pins (B0-B15), also laid out in pairs together with V* and GND. These analog pins can also be configured as digital I/O pins, but should never have more than 3.3v applied to them. This is different from the digital 'E' pins (and all other pins on the board, including C, D, F and G), as these are 5v-tolerant when used as inputs. Interfacing with 5v-systems is simple because of this, together with the fact that most 5v-systems recognize a 3.3v output signal as a logic 'high' – if not, pins can be configured as 'open drain' outputs, and external pull-up resistors to 5v can be used.

The software environment used for programming the CUI32Stem in the Arduino language is the ChipKIT MPIDE [6] (Multi-Platform Integrated Development Environment). Ongoing work in the open source community continues to improve MPIDE, by adding support for a variety of other microcontroller boards as well. It is important to note that some of the most recent design changes to the CUI32Stem board were also made possible by the generous contribution of the open source community. For example, the board layout for the CUI32Stem has been refined by Markus Gritsch, and the Arduino-compatible bootloader was written by Rich Testardi.

2.1 The GROVE system

SeeedStudio, Inc. is a purveyor of the GROVE system for prototyping electronic interfaces. SeeedStudio is an 'open hardware facilitator' based in China, but design contributions are encouraged from all around the world (via the internet) to add to the increasing collection of GROVE elements [3]. These elements include a wide range of sensors and actuators that may be useful in various research fields. The primary interest here is in interaction design for sound and music computing. Therefore, sensors that capture human input (as opposed to environmental sensors or other types) come to the forefront. Quite a few human input sensors are available as GROVE elements. Another one of the goals of the system is to provide accessibility to as many people as possible. As such, GROVE elements (as well as the CUI32Stem itself) are not extremely expensive to purchase.

There are many similar concepts to this GROVE system, as can be seen in, for example, Teenage Engineering's 'Oplab' [7], Teague's 'Teagueduino' [13] as well as Microchip's own 'Digilent Cerebot with P-MOD' (Peripheral-Modules) system [2], all of which allow non-soldering approaches to interaction design or other prototyping systems. While the Opblab is specifically focused on music interfaces, the Teagueduino and Microchip offerings are more general-purpose. The CUI32Stem and the GROVE system can also be used as general interaction design toolkits, but the author's research focuses on interaction design for sound and music computing. Nonetheless, a somewhat generic approach facilitates teaching within many areas, and the need for a more general-purpose system arises commonly among educational programs.

3. WIRELESS AND OTHER EXAMPLES

The CUI32 has several options for wireless capabilities, three of which have been recently explored by the author: ZigFlea, Bluetooth, and WiFi. These are described only briefly below. Readers wishing for a more thorough discussion of wireless options can see [10]. A 'Serial Bluetooth' module[1] is available as part of the GROVE system. It provides a wireless link with 10 meters of range. Once paired, the Bluetooth connection works in the same way as would a standard USB-cable connected between the CUI32Stem and a computer, as explained in the CUI32 website[2], which includes downloads with examples for PureData and MaxMSP (see Figure 2). With this Bluetooth module, it is possible to remotely connect to StickOS via a terminal program to re-program the CUI32Stem wirelessly, if so desired. These examples are exactly the same when using Bluetooth or USB, and other examples also exist for interfacing the CUI32Stem to host software, such as openFrameworks[3].

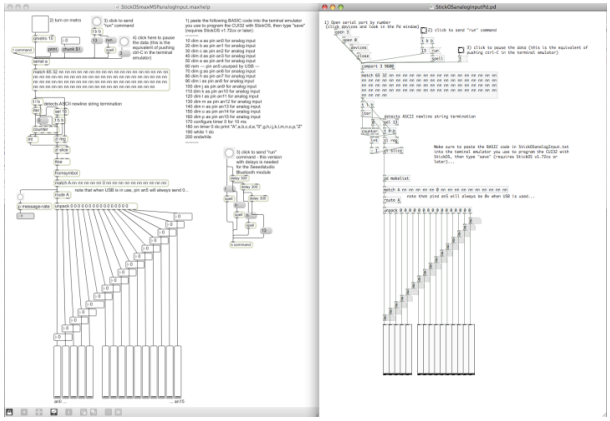

Figure 2. Example MaxMSP and Pd patches, downloadable via the Google Code website for the CUI32Stem: http://code.google.com/p/cui32/

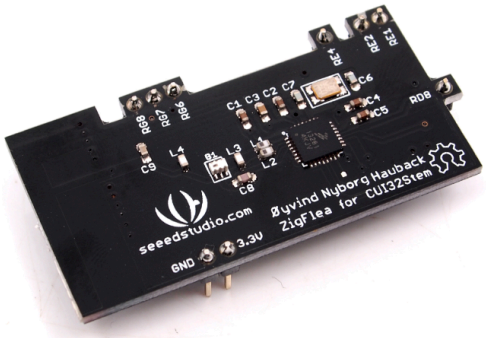

Figure 3. the ZigFlea add-on for the CUI32, developed by Øyvind Nyborg Hauback at the University of Olso, Norway, sold by www.seeedstudio.com

A 'ZigFlea' add-on board for the CUI32Stem is also available from Seeed Studio[4]. It was designed to plug onto the top of the CUI32Stem, without conflicting with any of the GROVE headers (see figure 3). A *CUI32 ZigFlea starter kit*

[1] GROVE Bluetooth-serial module, http://www.seeedstudio.com/depot/grove-serial-bluetooth-p-795.html

[2] http://overtone-labs.ning.com/profiles/blogs/cui32-analog-inputs-in-1

[3] openFrameworks, http://www.openframeworks.cc/

[4] ZigFlea, http://www.seeedstudio.com/depot/zigflea-p-1146.html

software download is available at the Google Code website[5], and includes the necessary StickOS BASIC code as well as a MaxMSP patch for receiving data from a remote node when ZigFlea add-ons are attached to both CUI32Stems. It should be noted that at the moment, the maximum data rate of ZigFlea in StickOS is limited at the moment [15], however this will be optimized in future StickOS releases.

Finally, 802.11b/g – commonly known as WiFi is an upcoming wireless option to be available as a GROVE element. The design uses a module called the WiFly (see Figure 4) that is made by Roving Networks [11]. This module can be configured to broadcast its own 'AdHoc' 802.11 base-station, or join existing WiFi networks. It allows the CUI32Stem to send raw UDP and/or TCP-based packets, and communicate easily with any software that supports Open Sound Control (OSC). One of the strengths of this approach is that the CUI32Stem can be used directly with any iOS or Android device, without having to use a laptop as a 'bridge'. It should also make it possible to develop future examples that serve simple webpages from the CUI32Stem.

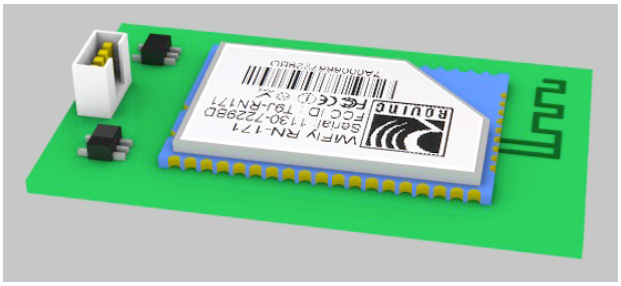

Figure 4. Rendering of the WiFly 802.11b/g radio module developed by Tobias Thyrrestrup - this will be available from Seeed Studio as the 'Serial WiFi' GROVE element

```
Connect to WiFly via a terminal program and confgire these settings:
$$$  (this puts the WiFly into configuration mode)
> set comm time 1
> set u b 115200    (or set u i 115200)
> save

On the CUI32, in StickOS (connected via USB):
configure uart 2 for 115200 baud 8 data no parity

Then, to telnet into the CUI32 from OS X use Terminal.app, example:
> telnet 169.254.1.1 2000
and type "^]" (to get telnet into command mode) then "mode character"
(this configures telnet so that ctrl-C works properly)
Now you are at the StickOS prompt, ready to make a BASIC program
```

Figure 5. The WiFly / CUI32Stem configurations for using them together to send/receive 802.11 data. This also allows to 'telnet' into the CUI32Stem and write / debug a BASIC program in StickOS interactively

For Sound and Music interaction design, the values from sensors attached to the CUI32Stem (GROVE or otherwise) can be used to control any parameters of real-time processes running on a mobile device, such as audio synthesis or effects algorithms in Pd (RjDj [12], LibPd [4]) or SuperCollider [5] (ports to iOS and Android can be found). The mapping of such controls to real-time parameter updates is of course a major task given to a composer / performer / developer of the system, as well as any haptic feedback for the user that may be controlled by the CUI32Stem. The following are specific examples for these use cases.

[5] ZigFlea starter kit for the CUI32Stem, http://code.google.com/p/cui32/downloads/list

A major advantage of using WiFi 802.11b/g over either ZigFlea or Bluetooth, is the much greater bandwidth and lower latency it offers. Two examples of sending data to an iOS device are shown below – one that sends UDP data to RjDj (Pd-vanilla running on iOS), and another that sends Open Sound Control (OSC) packets to SuperCollider running on the iOS device. The first example (sending UDP data to RjDj, which receives it via the [netreceive] object in Pd) is shown below. Declaring an analog input variable as 'debounced' causes the raw ADC value from a sensor to be averaged over 3 samples at 4ms intervals.

```
10  configure uart 2 for 115200 baud 8 data no parity
20  dim a as pin an0 for analog input debounced
30  dim b as pin an1 for analog input debounced
40  dim c as pin an2 for analog input debounced
50  dim d as pin an3 for analog input debounced
60  dim e as pin an4 for analog input debounced
70  dim g as pin an6 for analog input debounced
80  dim h as pin an7 for analog input debounced
90  dim i as pin an8 for analog input debounced
100 dim j as pin an9 for analog input debounced
110 dim k as pin an10 for analog input debounced
120 dim l as pin an11 for analog input debounced
130 dim m as pin an12 for analog input debounced
140 dim n as pin an13 for analog input debounced
150 dim o as pin an14 for analog input debounced
160 dim p as pin an15 for analog input debounced

170 sleep 100 ms  rem -- wait for WiFly to boot
180 print "$$$";  rem -- tell WiFly to enter config mode
190 sleep 300 ms  rem -- wait for it to enter config mode
200 print "open 169.254.1.3 2000"  rem -- connect to iPod Touch
210 sleep 100 ms  rem -- wait for connection to establish

220 configure timer 0 for 10 ms  rem -- interrupt update rate 100Hz
230 on timer 0 do print "A",a,b,c,d,e,"0",g,h,i,j,k,l,m,n,o,p,";"
240 while 1 do  rem -- no need to do anything in the main loop because
250 endwhile  rem -- everything is handled by the interrupt routine
```

Figure 6. StickOS BASIC program that sends the CUI32Stem's analog sensor inputs to the [netreceive] object in the RjDj app on iOS, which is running a corresponding RjDj "scene" (this patch is shown immediately below)

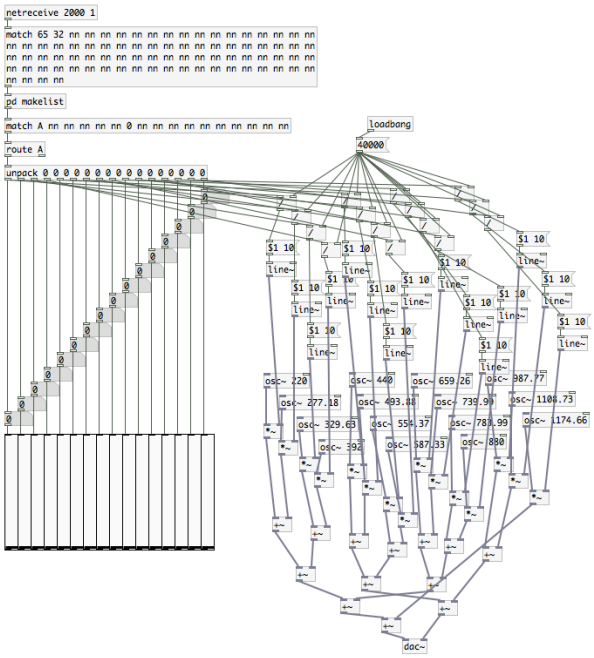

Figure 7. PureData patch used in RjDj to receive real-time analog sensor data from the CUI32. On the left side, sliders visually represent incoming sensor values, and on the right side, a simple additive synthesizer generates different timbres in response to the incoming sensor data

Some explanations are necessary for the BASIC code shown in Figure 6. On line 10, the 2nd UART (the serial port of the CUI32Stem that is connected to the WiFly) is initialized. Lines 20 through 160 are declaring variables 'a' through 'p', corresponding to the 16 analog input pins on the CUI32. Lines 170-210 create a connection between the iOS device and the CUI32Stem (the iOS / Android device or computer must already have joined the WiFly's AdHoc network). Finally, line 220 enables an internal timer in the CUI32Stem (functionally similar to the [metro] object in PureData), and configures it to cause 'interrupt' events every 10 milliseconds. Every time one of these events happens, line 230 sends the actual sensor values to RjDj, which is running the PureData patch seen in Figure 9.

The list of sensor values is always preceded with a capital "A", and appended with a semicolon. In PureData, this semicolon is needed by the [netreceive] object to signify the end of an incoming packet, and the capital "A" is simply used as an identifier to signify the beginning of the packet. Therefore, the top [match] object in Figure 9 checks for the capital "A" at the beginning of the string, so that synchronization is always maintained.

While this example is specific to RjDj (as it uses PureData internally), the same functionality can be achieved with other applications that receive Open Sound Control, with just a few modifications to the BASIC code. For example, SuperCollider requires the use of OSC-format strings in order to receive UDP data, so the addition of the proper OSC syntax (string identifiers and 4-byte boundaries) is added. The example below allows data to be received in any application that 'understands' OSC.

```
-- The StickOS BASIC code below sends the OSC string:
-- /s,iiiiiiiiiiiiiiii with all 16 of the analogInputs
-- at 500Hz (every 2ms)

configure uart 2 for 115200 baud 8 data no parity

dim s[22], j -- array[s] is used to send the OSC string
dim an[0] as pin an0 for analog input debounced
dim an[1] as pin an1 for analog input debounced
dim an[2] as pin an2 for analog input debounced
dim an[3] as pin an3 for analog input debounced
dim an[4] as pin an4 for analog input debounced
dim an[5] as pin an6 for analog input debounced
dim an[6] as pin an7 for analog input debounced
dim an[7] as pin an8 for analog input debounced
dim an[8] as pin an9 for analog input debounced
dim an[9] as pin an10 for analog input debounced
dim an[10] as pin an11 for analog input debounced
dim an[11] as pin an12 for analog input debounced
dim an[12] as pin an13 for analog input debounced
dim an[13] as pin an13 for analog input debounced
dim an[14] as pin an14 for analog input debounced
dim an[15] as pin an15 for analog input debounced
let s[0] = 796065792  rem -- OSC header "/s00" (0x2F730000)
let s[1] = 745105769  rem -- OSC header ",iii" (0x2C696969)
let s[2] = 1768515945 rem -- OSC header "iiii" (0x69696969)
let s[3] = 1768515945 rem -- OSC header "iiii" (0x69696969)
let s[4] = 1768515945 rem -- OSC header "iiii" (0x69696969)
let s[5] = 1761607680 rem -- OSC header "i000" (0x69000000)

while 1 do
  for j = 0 to 15
    let s[j+6] = an[j]
  next
  print raw s
  sleep 2 ms
endwhile
```

Figure 8. A StickOS BASIC program that sends all of the CUI32Stem's analog sensor inputs as OSC data.

This last example incorporates proper syntax elements needed at the beginning of an OSC-formatted string. The somewhat cryptic numbers in the array [s] in Figure 8 represent the string "/s,iiiiiiiiiiiiiiii" as required at the beginning of the OSC string. These are derived from the ASCII-equivalents of the individual characters in this string, and aligned to 4-byte boundaries as required by the Open Sound Control protocol

In order to use the sensor data from the CUI32Stem with interactive music programming environments, the data must be formatted into either OSC as shown here, sent serially via USB, or make use of another format that the application understands, such as MIDI, HID mouse/keyboard data, etc. The examples shown here focus on interaction design for sound and music computing, and provide corresponding MaxMSP/Pd patches that capture sensor values on the 16 analog input pins on the CUI32Stem. There are of course many different types of analog sensors that can be used with CUI32Stem, including those already in the GROVE system. In addition, simple extensions to the above examples would allow data from digital sensors (such as those which communicate via I2C or SPI, some of which are also included in the GROVE system) to be attached. Extending these examples to incorporate the control of LEDs, small motors, or other actuators for user feedback is also possible.

4. INNOVATIONS AND AVAILABILITY

The main objective of the development of the CUI32 is to pursue the author's own long-term research focused on the development of new electronically enhanced (augmented) musical instruments. One of the aims of this is to explore the potentials that such instruments have in the context of new compositions and new methods of performance. The overall goal is to add new dimensions and expressive possibilities to the capabilities of traditional electronic and/or hybrid acoustic instruments, and to explore these in contemporary music and performance. The research can be seen as a process of discovery, investigating the extension of musical instruments' expressive and performative ranges. In addition, through research-based teaching, students are exposed to the issues that arise in this research, and asked to build significant group-based projects through their semester work.

For the sake of brevity, a complete discussion of the bigger research challenges and innovations is not included here; One example of a real musical instrument created by the author, that uses the technology described herein, is the Overtone Fiddle [9]. This document instead serves as collection of useful technical information for those interested in working with the CUI32Stem, the GROVE system, and their corresponding methodologies.

The CUI32Stem is available as a single item, or bundled with an assortment of GROVE elements from Seeed Studio, Inc. At the moment, a 'CUI32Stem GROVE Dash Kit'[6] bundle has been put together (see Figure 9), and a larger bundle focused on wireless (including 2 CUI32Stems and 2 ZigFlea add-ons) is being prepared. Readers can check the SeeedStudio online wiki for more information about these bundles. More information about each of the individual GROVE elements is also available online, including schematics and related files.

In addition all of the GROVE elements can be purchased individually, if a specific design requires a certain sub or super-set of these bundled 'experimentation kits'. The bundles are simply intended to provide a convenient way of obtaining a collection of various sensors and actuators that support sound and music interaction design (performance, composition, installations, and other types of interfaces) in the context of lab, experimental, and educational use.

[6] CUI32Stem GROVE 'Dash Kit', from SeeedStudio http://www.seeedstudio.com/wiki/CUI32Stem_GROVE_Dash_Kit

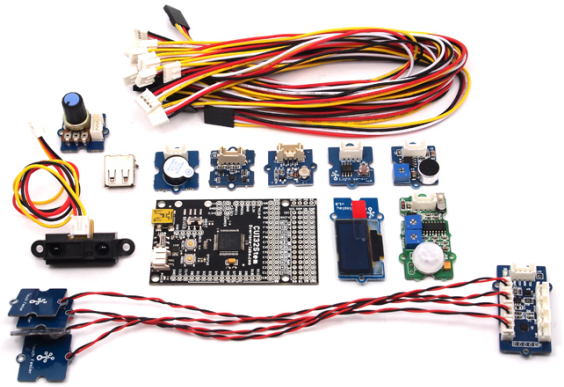

Figure 9. The CUI32Stem GROVE Dash Bundle. Note: the CUI32Stem board will ship with header pins pre-installed (not shown in the above photo) in this bundle, providing soldering-free 'sketching in hardware' methodologies

The CUI32Stem GROVE Dash bundle includes:

- 1 CUI32Stem board
- 1 USB Host connector *
- 10 Grove - Universal 4 Pin cables
- 1 Grove - I2C 3-axis Accelerometer
- 1 Grove - Rotary Angle Sensor
- 1 Grove - PIR Motion Sensor
- 1 Grove - Sound Sensor
- 1 Grove - I2C Touch Sensor (with 4 Feelers)
- 1 Grove - Collision Sensor
- 1 Grove - Piezo Buzzer
- 1 Grove - OLED Display 128x64
- 1 Grove - 80cm Infrared Proximity Sensor
- 1 Grove - Light Sensor

* Note: users must solder the USB Host connector onto their CUI32Stem if desired. It is left unpopulated because it can soldered on either the top or bottom side of the board (which has footprints for the connector on both sides). This allows users to choose the best position for their own project(s).

.

5. CONCLUSION

This system of DIY electronics for interface development – described herein with a focus on 'Sketching in Hardware' for music interaction design – was developed within research and teaching areas related to sound and music computing. It is shown in the examples provided that the use of a high-performance microcontroller and a free RTOS brings about an ease-of-use that can be good for end users, such as students and researchers who do not wish to get 'lost in the details' while first prototyping new ideas. This methodology is intended to transcend many of the 'nuts and bolts' of technological issues encountered when creating new forms of sound and music interface design prototypes, thereby leaving more time and effort to concentrate on the concept, research, and practice-based exploration of the field. While the CUI32Stem and GROVE system function as general-purpose toolkits, this paper has focused specifically on sound and music interaction design scenarios for their use.

6. ACKNOWLEDGEMENTS

The author would like to thank open source contributors Markus Gritsch, Rich Testardi, Rick Anderson, Marc McComb, Brian Schmalz, Philip Burgess, Øyvind Nyborg Hauback, Tobias Thyrrestrup, and Nana Chou.

7. REFERENCES

[1] CUI32, as sold by SparkFun Electronics, http://www.sparkfun.com/products/9645 See also: open source firmware at http://code.google.com/p/cui32/ Both accessed February 7, 2012.

[2] Digilent Cerebot and P-MODS system, http://www.microchipdirect.com/searchparts.aspx?q=cerebot&resperpage=10, accessed February 7, 2012.

[3] GROVE system, from SeeedStudio Inc., http://www.seeedstudio.com/wiki/GROVE_System#Grove_elements accessed February 7, 2012.

[4] LibPd, http://gitorious.org/pdlib/ accessed February 7, 2012.

[5] McCartney, J. Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal, 26(4), 61-68.* 2002

[6] MPIDE, from ChipKIT, http://www.chipkit.cc/wiki/index.php?title=MPIDE_Installation accessed February 7, 2012.

[7] Oplab, from Teenage Engineering, http://www.teenageengineering.com/products/oplab/ accessed February 7, 2012.

[8] Overholt, D. Musical interaction design with the CREATE USB Interface: Teaching HCI with CUIs instead of GUIs. *Proc. of the 2006 International Computer Music Conference*, New Orleans, 2006.

[9] Overholt, D. The Overtone Fiddle: an Actuated Acoustic Instrument. *Proc. of the 2011 New Intefaces for Musical Expression conference*, Oslo, Norway, 2011.

[10] Overholt, D. Musical Interaction Design with the CUI32Stem: Wireless Options and the GROVE system for prototyping new interfaces. *Proc. of the 2012 New Intefaces for Musical Expression conference*, Ann Arbor, Michigan, USA, 2012.

[11] RjDj, http://www.rjdj.me/ accessed January 29, 2012.

[12] Roving Networks (WiFly GSX module), http://rovingnetworks.com/ accessed January 29, 2012.

[13] Teagueduino, from Teague Labs, http://teagueduino.org/, accessed January 29, 2012

[14] Testardi, R. (StickOS), http://cpustick.com/stickos.htm accessed January 29, 2011.

[15] Tørresen, J., Hauback, Ø.N., Overholt, D., and Jensenius, A.R., Development and Evaluation of a ZigFlea-based Wireless Transceiver Board for CUI32. Proc of the 2012 New Interfaces for Musical Expression conference, Ann Arbor, Michigan USA 2012.