

Arduino-Audio Board

[M.Schippling -- bozotronics](#)

**v1.0a 9/7/2015(original)-9/16/15
(used to prototype Sound Control Bit)**

The Arduino-Audio board is a carrier for an Arduino Pro-Mini micro-controller module. It brings most of the Arduino I/O pins out to external connectors, either a standard 24 pin ribbon connector or LittleBits™ proprietary three pin *Bitsnaps*. The Bitsnap input connectors directly feed three Arduino ADC input pins and the outputs are driven by three Arduino PWM output pins. In this it is similar to the LittleBits Arduino module, but the Arduino-Audio board also provides two 3.5mm stereo phone jacks and a quad rail-to-rail op-amp for pre-processing audio input and filtering PWM output. It can be used for prototyping -- or short run manufacturing -- of Arduino and audio processing or detection modules with an eye to integrating them into the LittleBits environment.

Detailed Description

The Arduino-Audio board contains provision for two 3.5mm stereo audio phone-jacks, and six LittleBits (LB) proprietary I/O connectors (three each of input and output). It holds a quad op-amp, a 24 pin header for miscellaneous I/O, and an Arduino Pro-Mini daughter board. The op-amp can be configured in a number of ways, for input amplification or PWM output filtering. Audio input from the 3.5mm jacks may be amplified and filtered before being presented to Arduino ADC inputs. The three LB inputs feed other Arduino ADCs directly. Arduino digital PWM pins are buffered by the op-amps and connected to the three LB outputs. The PWM pins are also connected to LEDs used as level indicators. The Pro-Mini can run any Atmega Arduino program and can communicate with the outside world using a serial to USB FTDI adaptor. Two I/O pins on the 24 pin header can be used for I2C communication.

Sound Control Bit Description

When used for the Sound Control Bit (SCB) prototype, the board has one LB input connector, for power; and three LB output connectors, for control voltage outputs. One audio jack is used as a mono (left channel) input which is amplified and low-pass filtered by op-amp channel A. This input is converted to a digital signal and run through a Fast Hartley Transform (FHT) -- similar to, but simpler than the better known FFT -- in a library obtained from openmusiclabs.com: ([ArduinoFHT3 v9.4.12](#)). Three audio frequencies are selected from the FHT output and their volume levels are used to set the PWM rate of three digital output pins. Each PWM signal is filtered and buffered by one of the remaining op-amp channels and sent to one of the LB output connectors. Each output channel has an LED which gives a (somewhat) continuous indication of its output level.

Other Uses

There is a two pin jumper which can be used to modify the behavior of the Arduino program -- for more detail on this and other options see Arduino I/O list below.

The second audio jack J2 can be used as a second input, an output, or as a splitter to route the right-channel stereo signal elsewhere. The specific operation is set by op-amp and jumper configuration.

Op-amp channel B can be configured as either a second audio input and filter, or as a quasi-audio output -- the audio quality will not be good, but it could be acceptable in some uses. However, either of these uses will eliminate one of the filtered PWM outputs.

Note: the full board schematic and layout are somewhat confusing due to the possible configuration options for op-amp channel B and the second audio jack, J2. In SCB use channel B is wired as a buffer-follower on PWM output D11 and all the extra components (including audio jack J2) are not used. This is reflected in the ARD_SB_schematic20.png schematic image.

The LittleBits inputs are connected to three additional Arduino ADC inputs which can be used as seen fit. There is no buffering on these inputs.

The 24 pin *aux I/O* header has power and Arduino I/O signals which can be connected to the rest of the world in various ways.

The Pro-Mini card has a built-in 5v regulator which may be used instead of the LB power system. There is provision for separate power input to the card which can be routed to the Pro-Mini Vraw pin for regulation, or directly to the 5v buss.

Arduino I/O	
arduino signal	function or connection
A0	input jack J1 via op-amp channel A
A1	jumper/config to select: input jack J2 via op-amp channel B, or LB input 1 direct
A2	LB input 2 direct
A3	LB input 3 direct
A4	aux I/O pin 4
A5	aux I/O pin 6
A6	aux I/O pin 8
A7	aux I/O pin 10
D2	aux I/O pin 12
D3	aux I/O pin 14
D4	aux I/O pin 16
D5	aux I/O pin 18
D6	aux I/O pin 20
D7	aux I/O pin 22
D8	aux I/O pin 24
D9	PWM to LB output 3 via op-amp channel C
D10	PWM to LB output 2 via op-amp channel D
D11	jumper/config to select op-amp channel B for: PWM to LB output 1, or low-fi audio to output jack J2
D12	Jumper to ground for optional option selection
D13	Pro-Mini on-board LED indicator
+5v	aux I/O pins 1,5,9,13,17,21
gnd	aux I/O pins 3,7,11,15,19,23

Software

The author (schip) has modified the Arduino internal wiring_analog.c module for interrupt driven sampling of the first four (A0-3) ADC inputs (there are other modifications to the file for other uses of no interest to this project, so it's more complicated than it looks).

The interrupt code continuously cycles through the first four ADCs and puts the results into an intermediate variable array. When analogRead() is called, the contents of the appropriate array location is returned immediately, rather than initiating and waiting for a conversion result. Note that ADC4-7 are not supported in this implementation but can still be used for I2C or digital I/O.

Additionally the interrupt code buffers 64 samples of A0 data. When the buffer is full it sets the global pointer "SchipBuf0" to the beginning of the buffer, and then begins filling a second local buffer with newer samples. The user program waits for SchipBuf0 to be set, munges the data as it sees fit, and (optionally) sets SchipBuf0 to NULL so it can go back to waiting. This is a one-way semaphore as the interrupt code does not check for NULL before resetting SchipBuf0. In theory, while the user program is working on one set of 64 values from SchipBuf0 the interrupt code is filling a second buffer and they alternate back and forth.

This theory works fine as long as the user program is finished with the first sample buffer before the interrupt program presents the second buffer. As is, the ADC pre-scale is set to the longest period (128) so, with a 16MHz Atmega 328:

- A single ADC conversion takes about .110ms
- A four channel conversion cycle takes about .45ms, thus sampling at about 2222Hz
- A 64 sample conversion cycle takes about 28.8ms --
This is the user buffer update period (when SchipBUF0 is set != NULL)

The ADC pre-scale can be set lower to get higher sample rates if one does the timing analysis...This is done in wiring.c, look for "ADCSRA".

Sound Control Bit Software Specifics

The Sound Control Bit uses only one ADC input, A0. The main loop() function waits for the interrupt code to collect a set of 64 samples and set SchipBuf0. The new buffer is concatenated to the previous one to make one 128 sample array effectively ping-ponging between the two ADC interrupt buffers. As they are copied, samples are converted from 10 bit unsigned to 16 bit signed values. Then a 128 point FHT is run, the results are analyzed, and the PWM outputs are set appropriately. This updates the outputs around every 30ms. The 128 point FHT and subsequent code takes about 2.5ms to run, so the program is only using about 10% of the CPU and is loafing the rest of the time...

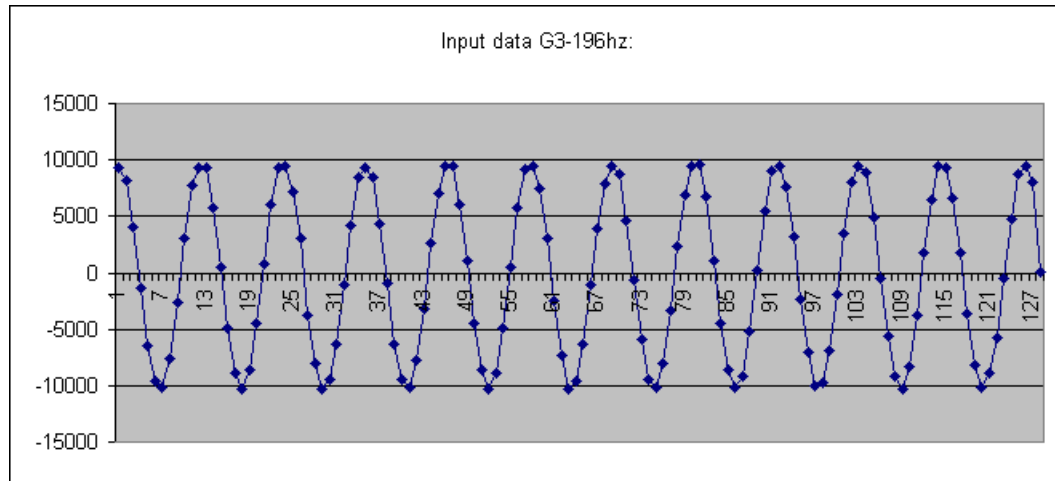
Some data dumps are available for debugging purposes. With an FTDI adapter cable connected to a PC running the Arduino IDE a serial connection at 115200 baud can be established. One of four characters can be sent to the SCB to turn on various functions. The debug characters should be sent singly with no Carriage Returns or extra chars appended:

- 1 -- print input array (128 ints)
- 2 -- print FHT output (64 bins)
- 3 -- print PWM settings only
- 0 -- shut printing off (any char except 1,2,3 will do)

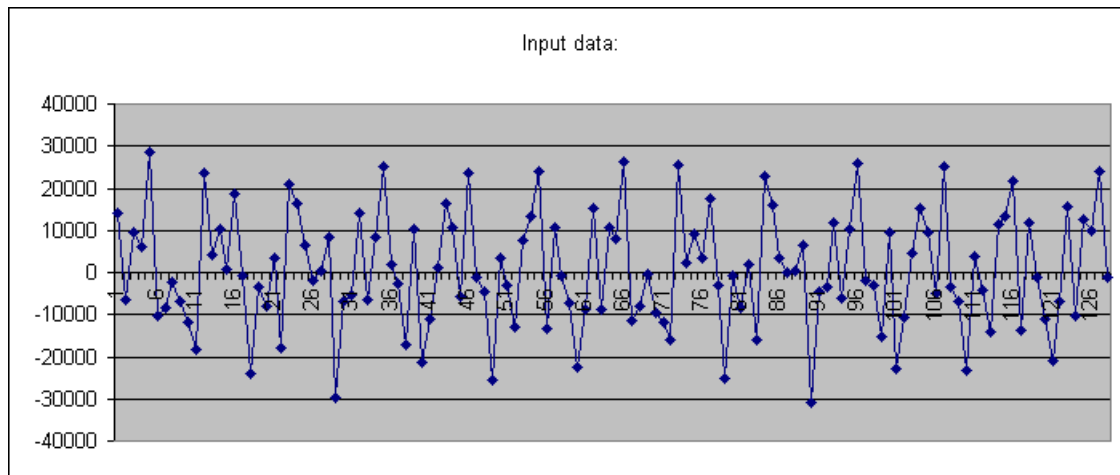
When set ON, a print dump will occur about every 7 seconds with one data value per line. For '1', the 128 sample FHT input data will be printed as signed ints between approx +/-32,000. For '2' the 64 bin FHT output will be printed as unsigned ints with values from 0-60 -- usable input values will be from about 0-25 (see discussion below) and each bin covers about 17.5Hz (at the 2222Hz sample rate). For '3' just the three PWM output values from 0-255 will be printed on a single line (this is more frequent at about once every 2 seconds). For '0', or any other garbage character, printing will be disabled. Dumping the larger input and output arrays may interfere with the

program's timing so they are throttled down to the once per 7 sec rate.

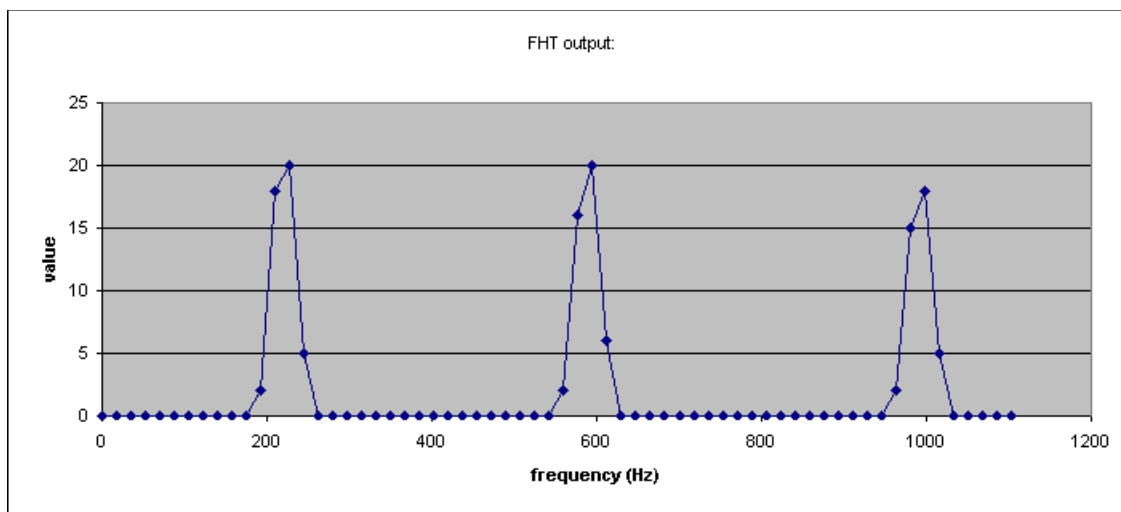
Plotting the data in Excel should show something like this:



Single low frequency audio sine input



threesins.wav sample file with all three sine waves at optimum levels



FHT output from threesins.wav
bin numbers are converted to Hz by multiplying by 17.5

Optimum Signals

The SCB has been set to react to three specific frequencies as shown in the FHT output graph above (note that bins below are numbered from 0, to match the actual FHT output array index). The notes were selected to be about equi-distant in the bin layout with frequencies that had limited common harmonics in order to maximize channel separation. The **"normal"** notes in bold are the ones to use in *normal* operation, the "motor" notes are for an experimental mode described further below.

Sound Control Tones			
output	frequency	note	FHT bin
1, motor 2.5-0v	196hz	G3	#11
1, normal 0-5v	220 hz	A3	#13
1, motor 2.5-5v	246hz	B3	#14
2, motor 2.5-0v	523hz	C5	#30
2, normal 0-5v	587 hz	D5	#34
2, motor 2.5-5v	659hz	E5	#38
3, motor 2.5-0v	932hz	Bb5	#53
3, normal 0-5v	988 hz	B5	#57
3, motor 2.5-5v	1047hz	C6	#60

The threesins.wav file contains sine waves at the three **normal** frequencies recorded at optimum levels such that the TOTAL signal is just below the maximum ADC conversion value. If it is any louder the signal will clip which adds harmonics and reduces the accuracy of the FHT when distinguishing tones. For the same reason, accuracy is diminished if the tones are not pure sine waves. The simplest voices on a synth, e.g., a flute, are good, but generating sines with your mixing program is the best.

The optimum level for each individual tone is 1/3 of the maximum or about -9db on a mix panel. When sending a single tone to the SCB it can be louder than 1/3 of the combined values but this will have no stronger effect. This is reflected in the FHT output dump mentioned above, while the actual range of a single tone may vary from 0-60, the usable range, when combined with two other tones is around 0-20.

The SCB program is set to turn on it's on-board Green LED when any signal exceeds the ideal maximum, so set your outputs to just barely flash that LED and it should be good.

Special Secret Experimental "Motor Mode"

When a jumper block is placed on the two pins between the Pro-Mini and the LB output connectors, pin 12 of the Arduino is connected to ground and the SCB program enters an experimental "Motor Mode"; YMMV when using this...

In Motor Mode when there is no relevant sound control input each output goes to 1/2 scale (2.5v). When a sound control tone is sensed the output moves above or below the middle point. Tones on either side of the normal set are used, with a slightly higher note making the output move to a higher voltage (2.5->5v) and a slightly lower note moving the output lower (2.5->0v). As usual the volume of the notes sets how much the output voltage changes, but the lower notes have an inverted response where low volume sets the output only slightly less than 1/2 scale and high volume sets the output to 0v.

So far the only Bit that might make use of this is the Hobby Servo, which sits at the middle of it's range when the input voltage is at 1/2 scale and then moves towards either extreme when input

increases or decreases from there. Continuous rotation hobby servos could be used to implement reversible motors, however our current observation is that the center-point for zero rotation is sensitively dependent on initial conditions (there's not enough dead-space between forward and backward rotation). However, this is a plausible model for how a reversible motor might be controlled, and is in fact the methodology used by the prototype stepper motor bit.

Files

Two zip files are needed:

One is the FHT code from openmusiclabs.com:

[ArduinoFHT3 v9.4.12.](#)

Download and unzip this file into your [Arduino-projects]/libraries directory where it will make an FHT directory with the code and some examples. It will also make a "reorder_table_creator" directory containing a utility for building the library's internal data tables. This can be ignored or removed.

The second file contains the hardware, software, and documentation for the Arduino-Audio Board under discussion, including this file:

[Arduino-Audio.zip.](#)

It should be unzipped into your [Arduino-projects] directory. It will make a "SoundBit" directory containing the following:

- **board/ARD_audio20.pcb** -- The board layout file
- **board/ARD_audio20.sch** -- The board schematic
(Both of these are in a proprietary format offered by ExpressPCB.com
-- for various reasons I just say NO to Eagle --
but never fear, there are png images in the doc directory.)
- **doc/Arduino-Audio.pdf** -- This file, all the documentation you get
- **doc/image*.gif** -- Images used in the documentation file
- **doc/ARD_audio_layout20.png** -- Image of the board layout
- **doc/ARD_audio_mech20.png** -- Image with the mechanical dimensions
- **doc/ARD_audio_schematic20.png** -- Image of the full board schematic
- **doc/fht_read_me.txt** -- Readme file from FHT library for reference
- **doc/SB_schematic20.png** -- Image of the actual Sound Control Bit schematic
- **doc/SBdataReal.xls** -- Example Excel debug data and plots
- **doc/threesins.wav** -- Example audio file containing all three control signals
- **SoundBit.ino** -- The actual Arduino program file to install
- **schip_analog.c** -- The code for ADC interrupts (based on wiring_analog.c v1.0.5-r2)
- **schip_analog.h** -- Header definitions and settings for schip_analog.c.
- **wiring_analog.c.original_v105r2** -- Original Arduino analog file

The interrupt modifications were done based on Arduino version 1.05r2, and the original file is included for reference. If you are not using version 1.0.5-r2 of the Arduino IDE the schip_analog.c file should be merged with the base Arduino file by comparing the base file:

[arduino-1.0.5-r2]\hardware\arduino\cores\arduino\wiring_analog.c

to wiring_analog.c.original_v105r2 and making any necessary changes. Keep copies of everything just in case...