How to make a modern drum module

By Jeremy Oden

Contents

Abstract

After more than forty years of existence, electronic drums have not evolved much compared to other electronic instruments. If professional drummers do not tend to use electronic drums on stage, that is mainly because drum modules do not emulate acoustic drums response and feel well enough. On the other hand, homemade drum modules have also existed for a long time, and they could now offer as much performance as professional modules. Thanks to recent technological advances, they have become a lot easier and less expensive to make. In this article, we will deal with the fabrication of homemade drum modules. In particular, we will show how they can achieve very high performance in terms of sound quality, dynamic range, and latency.

1 Introduction

1.1 State of the art

Drum modules, or drum brains, have been invented more than forty years ago. The first commercial electronic drum kit was the Pollard Syndrum[1]. Released by Pollard Industries in 1976, it consisted of an electric sound generator and up to four drum pads.

In 1981, the Simmons company released the Simmons SDS 5[2], which was considered to be the first viable electronic replacement for acoustic drums, although its sound quality was not very good. From that moment, drum modules have started to produce more realistic sounds. For instance, the Forat F16[3] was the first 16 bit electronic drum module, and was introduced in 1987 at a list price of $5200. It is the fastest audio triggering digital sampler ever sold, with a 0.1 ms response time.

Not long after, Roland introduced the V-Drums with its electronic drum set TD-7K in 1992, that featured a TD-7 module. The TD-7 and the less expensive TD-5 were later replaced by the TD-10 and TD-8, in 1997 and 1999, respectively. The particularity of the TD-10 is that it used mathematical models to generate its sounds using synthesizers. This was called the COSM technology. With the TD-10, its mathematical/computational sounds modeling, and the introduction of mesh heads, drum modules started to deliver good response and sound quality.

In the late nineties, drum modules were already delivering a rather good sound quality, and it did not take too long before hobbyists attempted to make their own. Homemade drum modules have started to flourish shortly after the TD-10 was released, and perhaps even before that. Most of them used microcontrollers and achieved decent results, whilst making them easy to build and affordable. For instance, in 1999 eDrums[4] was developed using a PIC16F84[5]. All the schematics, lists of components, documentations, etc. have been available on the eDrum's website since 1999. The module is pretty complete, and also provides a MIDI output.

Since then, a lot of microcontroller-based drum modules have been made by hobbyists[6], in particular using Arduino[7], Teensy[8] and STM32[9] microcontrollers. The high availability and choice of microcontrollers have contributed a lot to the development of homemade drum modules in the last fifteen years. However, we are now living in the era of single board computers (SBCs), and their availability and low price will undoubtedly contribute to a new generation of drum modules.

Until now, there have not been many serious SBC-based drum modules. Some attempts and demonstrations can be found[10, 11], but only a few usable modules exist. Although most SBCs operating systems are not real-time, there are other difficulties that slow down the development of SBC drum modules. For instance, they are harder to interface with sensors, and their sound devices are not always suitable for audiophiles.

Nevertheless, the world is changing and there is a huge potential for SBC drum modules. Recent electronic drums have very advanced built-in technologies, and perhaps we are on the verge of a new generation of drum modules. Recently, the Pearl mimic pro[12] was announced, and it is the first commercial module that offers a 7" touchscreen interface. Steven Slate, who produced one of the industry's first pro drum sample products and created the Slate Drum software (which is now the industry standard for professional drum sounds) has reacted to the Mimic pro by saying that "the Mimic Pro is a giant leap forward in drum module technology".

Unfortunately, the Mimic Pro is very expensive, and other manufacturers will probably develop comparable hardware at a similar price. To address this problem, an SBC-based drum module called [13] was created in 2015. It is an open source project, has a 7" touch screen as its user interface, and offers a low latency high quality sound rendering. Of course, the module's quality is

impacted by the less expensive hardware, but the results are still impressive, and will hopefully give hobbyists very precious guidelines on how to make a modern drum module. That is why this article aims to provide all the technical details that allowed the conception of an high quality open source drum module.

1.2 Context

In this article, we will focus on what could drastically change the way drum modules look and feel: high quality sound samples, and a pleasant user interface. Unlike the previously mentioned microcontroller-based drum modules, the open source project eXaDrums[13] aims to make a SBC module using a Raspberry Pi. In early December 2013, the hardware consisted of a Raspberry Pi model B, an analog to digital converter, and an accelerometer[1].

In May 2015 a first successful attempt to make a drum module led to the RaspiDrums[14] project. Its goal was to validate all the decisions that had been made regarding the hardware and software design. It was very efficient in terms of audio latency, and finally gave its place to a eXaDrums in September 2015.

EXaDrums takes all the best of RaspiDrums and improves it in many ways. It was designed to be used with a touchscreen, but can also be used with any kind of user interface, which makes it powerful and flexible. Its graphical user interface is shown in figure 1. As of early 2018, eXaDrums is still a work in progress but already integrates most features of a drum module.

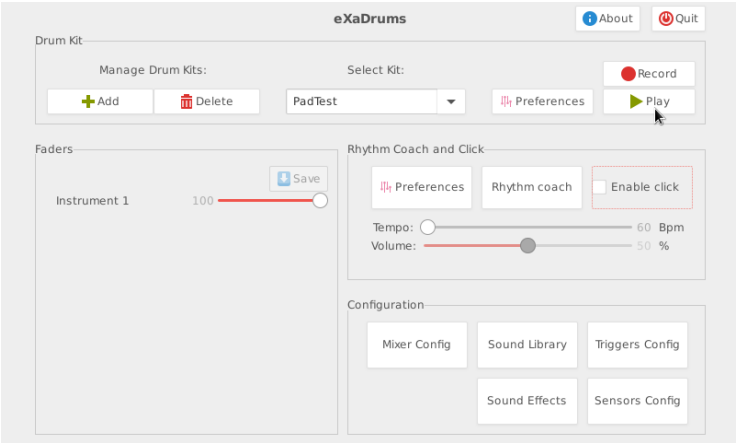


Figure 1: Graphical user interface of eXaDrums.

1.3 Our approach

Commercial drum modules are usually built around a dedicated processing unit that converts electrical signals from the sensors – or triggers – to sounds. Thus, we need a device that can read analog electrical signals, process them, generate sounds, and send them to an audio device. The next section will describe what led us to choose a Raspberry Pi to fulfill the role of the processing unit, despite its lack of analog inputs.

[1]The reason why an accelerometer was chosen will be discussed later.

Throughout the following sections, we are going to describe how to build a SBC-based drum module.

2 Hardware and sound

As mentioned in the introduction, the heart of our module will be a Raspberry Pi. Before we get into more detail, it is worth mentioning a few things that distinguished the Raspberry Pi from its competitors.

2.1 Single board computers

Single board computers have become widely available since the launch of the BeagleBoard-xM in 2010. Only a few single board computers have been released between 2010 and 2011, but, from 2012 on everything has accelerated, and we now see a dozen new SBCs every year. As a consequence, there are hundreds of SBCs, but they are not all good choices to make a drum module.

The projects this article is based on have started in 2013. At the time, the most popular SBCs were the Raspberry Pi, the BeagleBone Black, and the Odroid-XU. Although the Odroid and BeagleBone Black were a lot more powerful than the Raspberry Pi and both had analog to digital converters (ADC), those two boards were more expensive than the Raspberry Pi. Their ADCs also supported a maximum voltage of 1.8V, which was not enough as the first prototypes used accelerometers that had 3.3V or 5 V outputs.

What really made a difference were price, software support, and the user community. In 2013, the Raspberry Pi's operating system Raspbian was already stable and the user community very responsive. Moreover, the first project that has been developed (which was RaspiDrums) did not need a graphical user interface, so the Raspberry Pi had enough power to run it. The only drawback was the built-in sound card, which was not good enough to make a decent drum module.

2.2 Sound cards and latency

2.2.1 Latency

One issue when using a Raspberry Pi to make a drum module is its sound card. Because it uses a pulse width modulation (PWM) technique, it does not deliver a good sound quality nor a low latency. In order to solve that problem, two solutions are possible: to use a USB sound card, or to use an audio board. The problem with the latter is that it needs to be plugged in the GPIO port, which is unfortunately already used by the ADC.

A USB card seems to be a reasonable choice, as some of them can deliver a pretty good sound quality. The real question is how is the latency impacting the overall system's performance? First of all, in the case of a drum module, the latency is defined as the time interval between the moment a drum stick hits a drum pad and the moment a speaker outputs the corresponding sound.

For now, we will focus on the sound card's latency only. A sound card is essentially a digital to analog converter (DAC). It takes in some data, and converts it to an analog signal that, once sent to a speaker, produces a sound.

Let's consider a stereo sound device. The sound data has to contain both a right and a left channel. Each channel has to receive data from the audio system. We will denote every element of that data "L" and "R", respectively for the left and right channels. A group of samples "L" and "R" is called a frame, see figure 2.

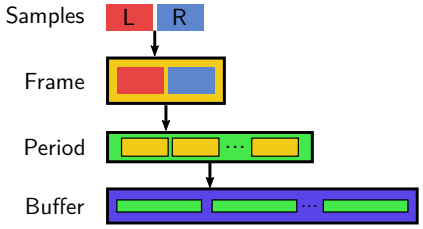


Figure 2: Illustration of the data contained in an audio buffer.

A succession of frames forms a period, and the audio buffer consists of a number of periods of the same size. The audio driver constantly accesses the buffer, but it can do so only one period at a time, otherwise data could be written to a period as it is being read. This is why the minimum number of periods has to be at least two, so that in theory, the driver never tries to read the period that is currently being written data to. Every time a period is read by the sound card, it is divided into frames, and then into samples, which are separated to be routed to their respective channels (right and left in the case of a stereo system).

Throughout this article, we will enumerate all the possible sources of latency of the system. There are two main sources of latency when using a soundcard: the buffer's duration, and the hardware's latency. The latter is often due to the filtering process that is used to reduce the output noise of the soundcard.

Professional drum modules usually are very low latency systems. As we saw in the introduction, the lowest latency is 0.1 ms for the Forat F16 drum module, but it remains crucial to determine what is the maximum latency that is acceptable for professional use? The table 1 lists some commercial drum modules and their corresponding latencies, i.e. the time it has taken the module to process the signal minus the time it took for the direct signal to reach the interface. Research on the subject tends to show that drummers agree that the

Drum module	**Latency**
Roland TD-30	3 ms
Roland TM2	4 ms
ATV aD5	4 ms
Yamaha DTX 700	5 ms
Pearl r.e.d.box	9 ms

Table 1: Commercial drum modules latencies. Source: vdrums.com (Module Latency Posted from Allan Leibowitz).

latency should not reach 10 ms, and ideally remain around 7 ms or less. Our target is then clear: the system needs to have a latency that stays below 7 ms.

2.2.2 Audio sampling

Now that we have seen how the data is transferred from the buffer to the speaker, we will show where this data comes from. We consider only one of the two channels of a stereo system. Each sample of data is a set of bytes and is referred as the resolution of the sound device. For instance, a 16 bit resolution sound sample contains two bytes per sample as one byte contains 8 bits. At a 48,000 Hz sampling rate, one second of sound data contains 48,000 samples, so 96,000 bytes. The time between two samples is thus 1/48,000 seconds, which is approximately $T = 20.83\,\mu$s. As shown on the figure 3, the sampling time determines the time interval between each sample.

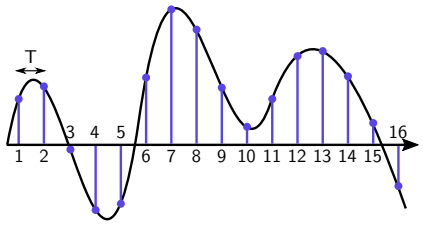


Figure 3: Sampled audio data, with a sampling time T.

The data transfer between software and hardware takes time, which is why the buffer size matters so much. Moreover, an audio device requires a minimum amount of samples to work properly. Common values are 64, 96, or 128 samples. Let's assume that a audio device needs a buffer of 96 samples. That corresponds to a latency of 2 ms. This is why the most important thing to keep in mind before choosing a sound card is to check the minimum buffer size in samples, and convert it in milliseconds to calculate its latency.

3 Interfacing with sensors

3.1 Different types of sensors

As mentioned before, there are many sources of latency, and the response time of a sensor is one of them. Common electronic drum kits use piezoelectric sensors, as they have a very short response time, usually less than a tenth of a millisecond. Other types of sensors can be used, though. For instance, accelerometers can reliably be used as drum triggers, as they provide a measure of the drum head's acceleration in real time [15].

Unlike piezoelectric sensors, accelerometers need a power source, and present a longer response time, as they usually need an output low-pass filter, often limiting their bandwidth to 1 kHz or less, which corresponds to a millisecond of latency or more. However, their main advantage is that their output voltage is very well controlled, and usually varies between 0V and 3.3V (or 5V depending on their supply voltage). Some even have a digital output, which could be very useful if we were to interface them directly with a Raspberry Pi.

It has been shown in [15] that drum heads acceleration can reach several thousands of m/s^2, which corresponds to more than a hundred Newtons. Fortunately, foam can reduce those values and thus allow the use of

standard accelerometers. Some accelerometers can measure an acceleration of $\pm 500\,g$, that is about $\pm 5000\,m/s^2$. For instance, the output of an ADXL001 accelerometer[16] goes from 0V (-500 g) to 3.3V (+500 g).

3.2 Interfacing sensors with an ADC

At least two different types of sensors can be used: piezoelectric sensors and accelerometers. On the one hand, accelerometers offer a very well controlled output voltage and the possibility of a digital output. On the other hand, piezoelectric sensors are very cheap and fast. They are both good solutions, but piezoelectric sensors are going to be our main focus for the rest of this article.

Because our drum module needs to handle more than one sensor, it is preferable to use an external ADC. Moreover, the bandwidths of the sensor and the ADC need to be adapted to the drum head's vibrations. Measurements tend to show that a 1 kHz bandwidth is enough to detect peaks and their intensities correctly[15].

Assuming that we need to connect eight sensors to the drum module, if we were to get ten data points every milliseconds, we would need an ADC that can get at least 80 ksps. Were the unit ksps means kilo samples per second, and represents the number of samples that we get in one second. Note that the vocabulary is very similar to the terms we used to describe the way sound cards work. That is completely normal: a sound card is a DAC, it does the reverse operation of an ADC. In short, a drum module uses an ADC to digitize values, process them, and then it uses a DAC to convert them to sounds.

Like an DAC, an ADC has two important properties: a sampling rate and a resolution. We already know that the sampling rate needs to be at least 80 ksps, but what about the resolution? If a sound card delivers a good sound quality with a resolution of 16 bits, a similar value would certainly be sufficient to ensure a good analog to digital conversion. In 2010 an extension to the MIDI standard introduced a high resolution velocity prefix in order to provide an additional 7 bits of precision for the velocity value. The reason for that was that 7 bits (or 127 possible values) was not enough for most musicians. We will assume here that 10 bits or 12 bits are decent values for what we intend to do.

Now that we now the requirements, we need to choose an ADC chip among probably hundreds of them. Fortunately, some people have already made boards to connect various chips to the Raspberry Pi. For instance, the RaspIO Analog Zero[17] uses a MCP3008[18] and is compatible with the Raspberry Pi's GPIO port. The MCP3008 has a resolution of 10 bits, and a maximum sampling rate of 200 ksps. The RaspIO Analog Zero can also be used with a MCP3208, which provides a 12 bits resolution, and a sampling rate of 100 ksps. Both the MCP3008 and MCP3208 communicate over the SPI bus, and the Raspberry Pi can handle two SPI devices simultaneously.

3.3 Piezoelectric sensor conditioning

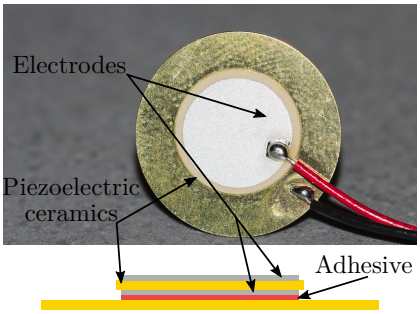
Interfacing with accelerometers is relatively easy, as their output voltage is well controlled and compatible with the MCP3008. However, commercial drum kits use piezoelectric sensors, and their output voltage has very different properties. We are going to show how to map the output voltage of a piezoelectric sensor to the MCP3008's input voltage range.

3.3.1 Piezoelectric sensor model and output voltage scaling

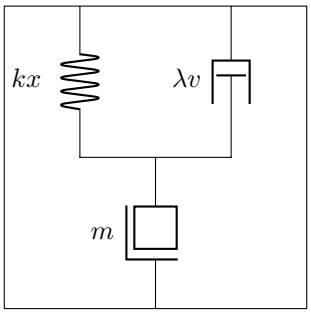
The output voltage of a piezoelectric sensor alternates between negative and positive peaks that can reach more than 20 V. Unfortunately, those values are too high for an analog to digital converter like the MCP3008. In order to solve that problem, it is necessary to understand how a piezoelectric sensor works.

A piezoelectric element is made of a piezoelectric ceramics and a metal plate held together with adhesive, Fig. 4a. Electrodes are placed on both sides of the piezoelectric ceramics to ensure electrical conduction. Exposure to mechanical strain causes the material to develop an electric field.

In our case, the piezoelectric element behaves like an accelerometer. The metal plate is bound to the drum head, and acts as a seismic mass. As the drum head undergoes an acceleration, the sensor behaves like a spring-mass-damper system, Fig. 4b.



(a) Composition of a piezoelectric sensor.



(b) Mechanical model of a piezoelectric sensor.

Figure 4: Piezoelectric sensor composition and mechanical model.

Such a system naturally exhibits a resonance, which can be observed in the electrical response of piezoelectric transducers, and also explains how they can be used as buzzers to generate sound when driven by an adequate input voltage. When used as an accelerometer, the resonance frequency of the system defines the upper frequency limit of the acceleration.

Using mechanical-electrical analogies, we can transpose the behavior of a piezoelectric accelerometer into the electrical domain. In the spring-mass-damper system the mass and spring exchange kinetic and elastic energy, respectively. Because of the damper, the energy decreases over time: the energy is not conserved. The same phenomenon happens in an RLC circuit, as the inductor and capacitor both exchange energy, and the resistor leaks some of it.

The table 2 shows that there is a direct analogy between current and velocity, and force and voltage. The latter tells us that if a piezoelectric element is subject to a strain, the resulting forces are equivalent to a voltage source in the electric domain.

The electrical model of a piezoelectric sensor is thus a voltage source associated to a RLC circuit. The model can be refined by adding a capacitor C_0 in parallel with the resistor, to take into account the static capacitance of the piezoceramics element, as shown by Fig. 5.

It is important to note that this circuit is very similar to a voltage divider, where the two resistors would be replaced by impedances Z_1 and Z_2, where

Spring-mass-damper system	**RLC circuit**
Kinetic Energy: $U_k = \frac{1}{2}mv^2$	Inductor Energy: $U_L = \frac{1}{2}Li^2$
Elastic Energy: $U_e = \frac{1}{2}kx^2$	Capacitor Energy: $U_C = \frac{1}{2}\frac{1}{C}q^2$
Frictional force: $F_f = \lambda v$	Voltage drop: $U_R = R\,i$

Table 2: Mechanical-electrical analogies show that x and $v = \frac{dx}{dt}$ are equivalent to q and $i = \frac{dq}{dt}$ in the electrical domain.

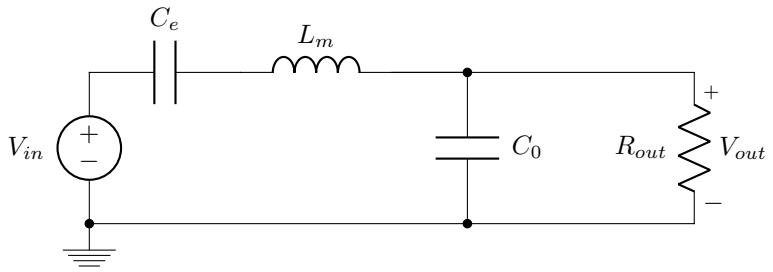


Figure 5: Piezoelectric sensor electric model.

Z_1 is given by the impedance of C_e and L_m, and Z_2 comes from the parallel association of C_0 and R_{out}:

$$Z_1 = \frac{1}{\jmath C_e \omega} + \jmath L_m \omega \tag{1a}$$

$$Z_2 = \frac{\jmath R_{out} C_0 \omega}{1 + \jmath R_{out} C_0 \omega}. \tag{1b}$$

The transfer function of the piezoelectric sensor is thus given by[2]

$$T(\omega) = \frac{Z_2}{Z_1 + Z_2} = \frac{1}{1 + \frac{Z_1}{Z_2}}, \tag{2}$$

which means that adding a load in parallel with R_{out} would essentially decrease the value of Z_2 for all frequencies, and would in turn reduce the gain $|T(\omega)|$. So if we add a potentiometer in parallel with Z_2, we will be able to scale down the output voltage of the piezoelectric sensor, as shown by Fig. 6.

The potentiometer must have a rather high total resistance, otherwise it will decrease the output voltage to very low values, and modify the response of the sensor. As we can see, it is not very hard to scale down the output voltage of a piezoelectric sensor. The potentiometer's wiper can be used to increase or decrease the output voltage's range at will. However, the resulting voltage still oscillates between positive and negative values, and it needs to be mapped to a positive voltage that is compatible with the ADC's input.

[2]Note that $T(\omega) = \frac{Z_2}{Z_1+Z_2} = \frac{\jmath R_{out} C_e \omega}{1 + \jmath R_{out}(C_e + C_0)\omega - L_m C_e \omega^2 - L_m C_e R_{out} C_0 \omega^3}$.

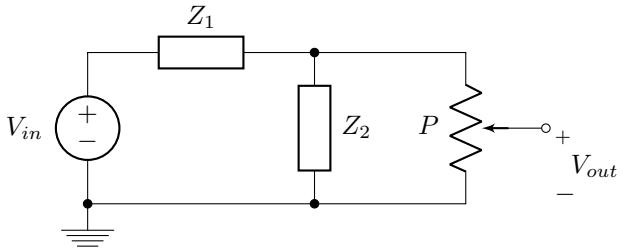


Figure 6: Piezoelectric sensor model with load.

3.3.2 Voltage mapping circuit

In this section, we will show how to map the output voltage of a piezoelectric sensor to the input voltage of an analog to digital converter. To that end, we need to know the properties of the output voltage of our sensor. Since we can attenuate that voltage, we are going to assume that its minimum and maximum values are -10 V and +10 V, respectively. As a consequence, we need to design a circuit that maps those voltages to 0 V and 3.3 V.

A simple mixer, just like an audio mixer, would be a perfect solution. In a two-input mixer configuration, the input voltages are added with different weights depending on their respective resistors values. In our case, the inputs would be the output voltage of the sensor and a DC voltage in order to generate the offset that we need. This configuration is presented in Fig. 7. The two inputs

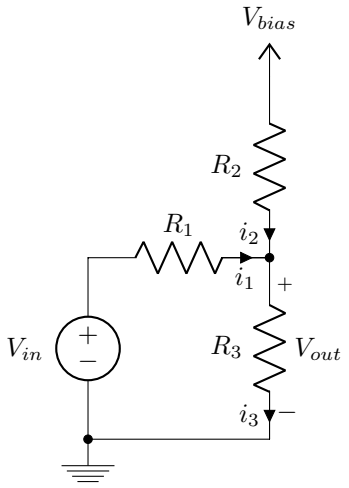


Figure 7: Three-resistor voltage mapping circuit.

are denoted V_{in} and V_{bias}, respectively. They are both connected to different resistors, and the output voltage V_{out} is connected to a third resistor, in order to convert the current i_3 into a voltage.

The output voltage can be derived using Kirchhoff's law:

$$i_3 = i_1 + i_2 \Rightarrow \frac{V_{out}}{R_3} = \frac{V_{in} - V_{out}}{R_1} + \frac{V_{bias} - V_{out}}{R_2}, \tag{3}$$

which leads to the following equation:

$$V_{out} \cdot \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} \right) = \frac{V_{in}}{R_1} + \frac{V_{bias}}{R_2}. \tag{4}$$

thus, V_{out} can be expressed as:

$$V_{out} = \alpha V_{in} + \beta, \tag{5}$$

which is exactly what we need, and allows us to map the sensor's output voltage to the ADC's input voltage by carefully choosing α and β. To determine the values of the three resistors, we are going to introduce a new constant that will help simplify the equations and solve them:

$$\kappa = 1/R_1 + 1/R_2 + 1/R_3. \tag{6}$$

Then, using equations 4, 5, and 6, we get:

$$\begin{cases} \alpha = \dfrac{1}{\kappa R_1} & \text{(7a)} \\ \beta = \dfrac{V_{bias}}{\kappa R_2}. & \text{(7b)} \end{cases}$$

The first order derivative of equation 5 gives us the expressions of α and β with respect to V_{in} and V_{out}:

$$\begin{cases} \alpha = \dfrac{\Delta V_{out}}{\Delta V_{in}} & \text{(8a)} \\ \beta = V_{out} - \alpha V_{in}, & \text{(8b)} \end{cases}$$

where $\Delta V_{out}/\Delta V_{in}$ represents the slope of the $V_{out} = f(V_{in})$ function. Using equations 7, we get:

$$\begin{cases} \kappa R_1 = \dfrac{\Delta V_{in}}{\Delta V_{out}} & \text{(9a)} \\ \kappa R_2 = \dfrac{V_{bias}}{V_{out} - \dfrac{V_{in}}{\kappa R_1}}. & \text{(9b)} \end{cases}$$

In order to complete this system of equations, we can use Eq. 6, which gives us: $1 = 1/(\kappa R_1) + 1/(\kappa R_2) + 1/(\kappa R_3)$, and leads to:

$$\begin{cases} \kappa R_1 = \dfrac{\Delta V_{in}}{\Delta V_{out}} & \text{(10a)} \\ \kappa R_2 = \dfrac{V_{bias}}{V_{out} - \dfrac{V_{in}}{\kappa R_1}} & \text{(10b)} \\ \kappa R_3 = \dfrac{1}{1 - \dfrac{1}{\kappa R_1} - \dfrac{1}{\kappa R_2}}. & \text{(10c)} \end{cases}$$

Unfortunately, κ remains unknown. However, the above equation shows that any value of κ gives us a solution. As a consequence, we can choose a value R_0 for R_1, and express R_2 and R_3 as functions of R_1, κR_1, κR_2, and κR_3:

$$\begin{cases} R_1 = R_0 & \text{(11a)} \\ R_2 = R_1 \dfrac{\kappa R_2}{\kappa R_1} & \text{(11b)} \\ R_3 = R_1 \dfrac{\kappa R_3}{\kappa R_1}. & \text{(11c)} \end{cases}$$

We use those last two equations to see how we can map a voltage that goes from -10 V to 10 V to the 0–3.3 V range. We have: $\Delta V_{out} = 3.3\,\text{V}$, and $\Delta V_{in} = 20\,\text{V}$, and κR_2 can be solved using any values of V_{in} and V_{out} can take, for instance: $V_{in} = 0\,\text{V}$, and $V_{out} = 1.65\,\text{V}$. Now we need to choose a suitable voltage for V_{bias}. Since we use a Raspberry Pi, it can either be 3.3 or 5 V. In this example, we are going to chose $V_{bias} = 5\,\text{V}$, which gives us:

$$\begin{cases} \kappa R_1 = \dfrac{20}{3.3} \approx 6.06 & \text{(12a)} \\ \kappa R_2 = \dfrac{5}{1.65} \approx 3.03 & \text{(12b)} \\ \kappa R_3 = \dfrac{1}{1 - 3.3/20 - 1.65/5} \approx 1.98. & \text{(12c)} \end{cases}$$

If we choose $R_1 = R_0 = 20\,\text{k}\Omega$, then we get:

$$\begin{cases} R_1 = R_0 = 20\,\text{k}\Omega & \text{(13a)} \\ R_2 = 20\,\text{k}\Omega \times \dfrac{3.03}{6.06} = 10\,\text{k}\Omega & \text{(13b)} \\ R_3 = 20\,\text{k}\Omega \times \dfrac{1.98}{6.06} \approx 6.5\,\text{k}\Omega. & \text{(13c)} \end{cases}$$

Although the calculated values of R_1 and R_2 are standard, that is not the case for R_3. The closest standard value for R_3 is 6.8 kΩ, which gives us an acceptable output voltage that varies between 0 V and 3.37 V when V_{in} varies from -10 V to +10 V.

3.3.3 Universal piezoelectric sensor voltage mapping circuit

Now that we have a way to map a sensor's voltage to our ADC's input specifications, we need to put all the pieces together to design our piezoelectric sensor conditioning circuit. The schematic of what we need is detailed by the figure 8. We first use a potentiometer P_{in} to make sure that the output voltage of the sensors fits into a range of ±10V. Then, in order to keep the load seen by the sensor as high as possible, we use an operational amplifier as a voltage follower. Its output is connected to the three-resistor voltage mapping circuit, which is itself connected to the analog to digital converter. Note that depending on the ADC's input impedance it might be necessary to add another follower between the three resistors and the ADC. This is not required if we use an MCP3008.

It is very important to note that the operational amplifier must be connected to a symmetrical power supply that delivers at least ±10 V. We will see how this can be achieved in the next section.

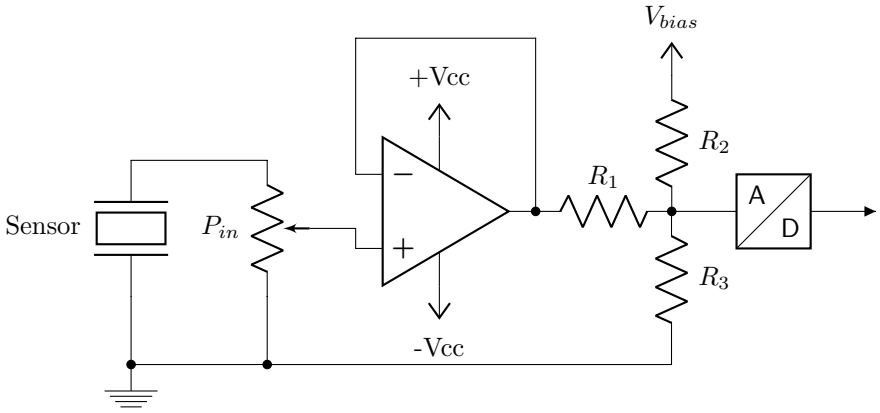


Figure 8: Piezoelectric sensor conditioning circuit.

3.3.4 Practical results

In order to validate the previous circuit, we are going to make it and compare its output to the piezoelectric sensor's output. To that end, we will use a 300 kΩ potentiometer, a TL081 operational amplifier, and three resistors: $R_1 = 20\,\text{k}\Omega$, $R_2 = 10\,\text{k}\Omega$, and $R_3 = 6.8\,\text{k}\Omega$. Although good results have been obtained with a Roland TD4-KP pad and a Roland PDX-100 snare with a mesh head, only the TD4-KP pad's response will be shown here.

First of all, it is important to choose a very high total resistance for the potentiometer, which is why we opted for 300 kΩ, but the higher the value, the better. The experimental setup is as follows: an oscilloscope probe is connected to the potentiometer's wiper in order to make sure that the voltage does not exceed 10 V in absolute value. Once the potentiometer is tuned so that this condition is fullfiled, it is not touched again.

As mentioned before, the operational amplifier requires a symmetric power supply. Since we do not have such a power supply yet, it would be best to generate it using the 5 V supply that is already used for the Raspberry Pi. In order to do that, a DC/DC converter has been used to convert the 5 V supply to a symmetric ±15 V source. A TMV 2-0515DHI has been chosen, but any similar converter can be used. Its ground is common for the +15 V and the -15 V outputs, and has been connected to the 5 V input supply's ground.

The output of the operational amplifier is wired to R_1, and a probe has been connected to the output of the three-resistor mapping circuit. As shown on figure 9, the curves (amplifier's input in blue, and three-resistor circuit's output in blue) have exactly the same shape. The difference between them is their minimum and maximum values. As per design, the output voltage of the piezoelectric sensor is scaled down to the ±10 V range. We can verify that, as we observe a minimum value of -9 V, and a maximum of 2.8 V (yellow line). As expected, the signal at the input of the ADC does not exceed 3.3 V and it does not go below 0 V either. From the blue curve, we can calculate the average value of the ADC's input voltage. We get 1.73 V which appears to be a little bit more than the expected 1.65 V. That is due to the 5 V power supply of the Raspberry Pi whose measured voltage is in fact 5.25 V.

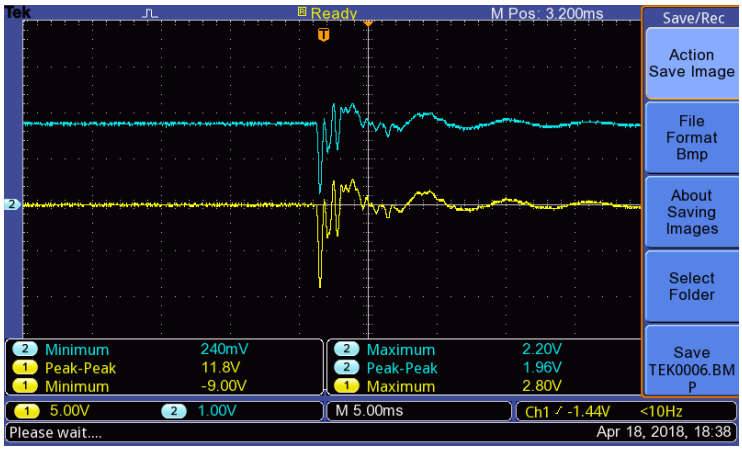


Figure 9: Input voltage of the operational amplifier (yellow) and the output voltage of the three-resistor mapping cricuit (blue).

That last remark enlightens the fact that there should be a numerical high-pass filter in the software that computes the ADC's data. That is vital as it protects the system from triggering against voltage variations due to the variable current draw of the Raspberry Pi. Also note that once the average voltage has been removed from the signal, only its absolute value matters as far as triggering goes.

It is worth mentioning that only a handful of commercial drum module ensure compatibility with other brand's pads. So it is a pretty unique feature for a drum module to be compatible with any drum pad.

Note that different components can be used to make this circuit. In particular, using an ICL7660 (or alternatively a MAX1044 or MAX660) to invert the 5 V supply appears to be an attractive solution, as it is a very efficient chip. Of course a compatible operational amplifier needs to be used. We recommend the ICL7621. Moreover, the resistors values have to be re-calculated in order to map a ±5 V to 0–3.3 V. To that end, we provide the Matlab/Octave source code that allowed to calculate the previous in the appendix A. This gives us the same value for the three resistors.

3.4 Sensors and latency

Unlike accelereometers, piezoelectric sensors' response is a lot shorter than a millisecond, but that does not mean that they induce no latency. Even though their response is fast, there are more than one source of latency that can explain why the digitized sensor's output may be delayed with respect to the actual drum stick hit. First of all, there is the drum head (or pad) response time, then the sensor's response, followed by the analog to digital conversion. The main contribution is probably the drum head's response time, as the sensor's output response and conversion are pretty fast.

But those are not really the bottleneck, as the slowest part is the peak detection algorithm. The algorithm itself is not slow at all, but it needs to detect the maximum velocity of the drum head, which means that it needs to

read the signal until the highest peak is detected. The software first detects if the signal goes above a certain threshold that is low, but higher than the noise. From that instant, the software needs to read the data for a longer time than the time it takes to reach the maximum velocity. To that end, a trigger scan window has to be defined. Drum modules usually let the user change that value from half a millisecond to a few milliseconds.

In practice, this setting depends on the mesh heads or pads used, but a value of 1 ms to 4 ms gives good results. This shows that, despite the very fast hardware response, the software inevitably introduces a latency of one to a few milliseconds, which is non-negligible.

4 Making the system

4.1 Hardware

The hardware of the drum module consists of what has been described before: an ADC and its adaptation board, a Raspberry Pi, and a sound card. The system that we present here is exactly what has been used to make the eXaDrums module. It is intended to be a fully working standalone drum module, and requires the following components:

- A Raspberry Pi 2B (or a more powerful version).
- A low latency USB sound card.
- The official Raspberry Pi 7" touchscreen.
- An enclosure.
- An analog to digital converter board.
- A voltage mapping board.

Of course, an SD card and a power supply are also needed. All of the above can be found pretty easily. We recommend a DesignSpark enclosure, as everything, apart from the voltage mapping board, fits pretty nicely in it. Note that the whole Raspberry Pi and enclosure kit can be found on the U.K. RS website, under the reference 124-4206 for £99.99, as shown on the figure 10.

Figure 10: Raspberry Pi 3 Premium Display Kit – RS U.K.

Assembling the kit is pretty straightforward. The difficult part is to take care of the interface between the sensors and the ADC. As we have seen before, an voltage mapping board needs to be made, in order to make sure that the signal coming from the triggers stays compatible with the ADC. The circuit has been detailed in the previous section, and can be extended to a eight- or sixteen-input board.

4.2 Operating system and software

Although it is recommended to use the latest version of Raspbian (Raspbian Stretch is recommended), most Linux distributions can run eXaDrums. Most USB sound cards work out of the box, and as long the SPI communications are enabled, reading values from the ADC will not be a problem.

The software is completely open source and free. It is written in C++, and as long as the right compiler is used, it is very easy to compile. For flexibility reasons, the software comes in two parts: an executable and a shared library.

All of the source code is available on Github, at the addresses given by references [19] and [20]. Instructions, dependencies and all the documentation are available on Github. Hopefully that will help the reader develop new powerful drum modules software.

Nonetheless, it is important to remember that every part of the system adds a little amount of latency, the software is no exception to that rule. Some of the latency is induced by the software itself, for instance the sensor's output peak detection, the sound processing, etc.

However, the rest of the latency comes from the operating system (OS) itself, as the official Raspbian distribution is not a real time OS. It means that the OS exhibts an inherent latency, which is moreover not constant. On a Raspberry Pi, this can lead to an increase of a few microseconds up to half a millisecond or slightly more. Adding that contribution to the software latency results in about one millisecond of software and OS latency, which is not a big issue.

5 Practical results

Software and hardware have been continuously tested during development of eXaDrums, and some benchmarks have given great results. We discuss here some of those tests and their results, and show how they confirm that our system is as good as some professional drum modules.

5.1 Latency measurements

Throughout this article, we have talked about the different possible sources of latency. By latency, we do not only mean the audio device's buffer duration, but the total latency, defined as the time it takes between the peak vibration of the drum head and the rise of the sound signal.

In order to measure the actual system's latency, an oscilloscope has been connected to both the piezoelectric sensor's output (CH1), and the sound card's output (CH2). The drum was hit many times and the delay between the two signals measured for each hit. The figure 11 shows a screen capture of the oscilloscope for one of those hits.

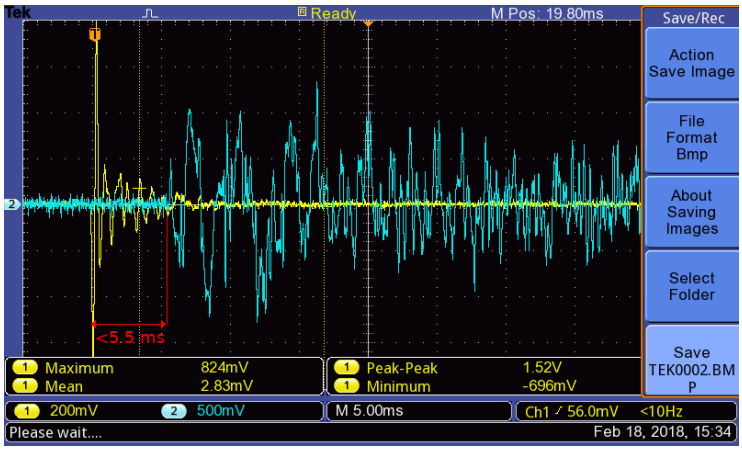


Figure 11: Delay between the peak intensity of the piezoelectric sensor's output (yellow) and the rise of the sound at the output of the sound card (blue).

As we can see, the latency does not exceed 6 ms (the time scale is 5 ms/div). Those measurements were performed on a non overclocked Raspberry Pi 3B, with a 2 ms audio buffer duration, and revealed a minimum latency of 5 ms and a maximum of 6 ms. The repeatability of that test was quite remarkable, as the only variable source of latency comes from the OS itself and does not vary more than a few hundreds microseconds.

This test was performed with a JustBoom DAC HAT to reduce the latency to its minimum. In contrast to a soundcard, this allows a direct transfer of sounds from the Raspberry Pi to a DAC. The reason for that is that it is possible to determine quite precisely the latency that comes from the hardware[21]. The next section will show how this helped to identify the latency contributions.

5.2 Latency contributions

As previously mentioned, there are four major sources of latency:

- The soundcard latency, see section 2.2.
- The sensors latency (hardware and software), see section 3.4.
- The operating system and software's latency, see section 4.2.

From the previous experiments, we know that the total average latency is about 5.5 ms. During the test, the trigger scan time has been set to 2 ms, as well the audio buffer's length. These account for most of the latency, but there still are 1.5 ms that are missing.

The remaining latency comes from the software, the operating system, and the hardware. We separate the hardware contribution from the OS and software one, which leads to the chart presented in figure 12. The operating system's and software latency is as we expected: a constant and a variable contribution with an average value of 1 ms.

The hardware latency however is theoretically constant, and according to the previous measurements, has a value of about 0.5 ms. Because a DAC was used

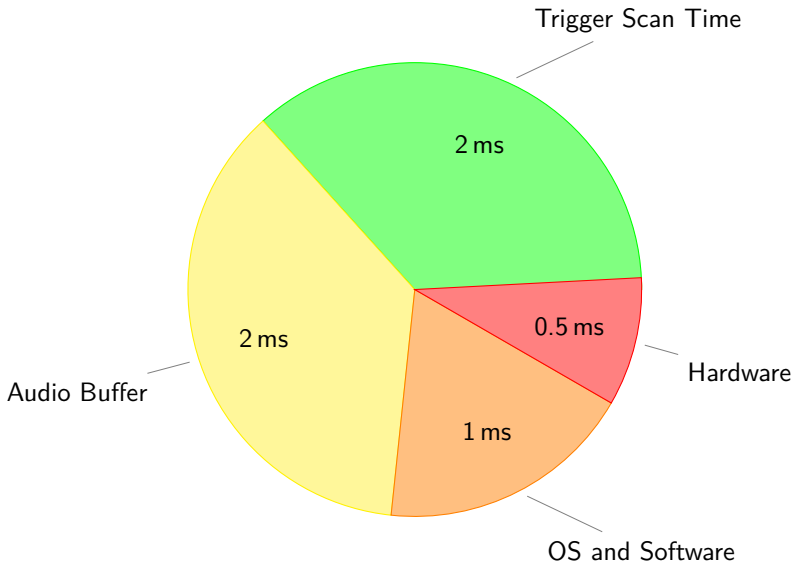


Figure 12: Latency contributions.

during the tests, it is possible to determine precisely where this latency originates from. The JustBoom DAC uses a TI PCM5122 DAC, followed by a TPA6133A2 headphone amplifier. Although the amplifier's latency can be ignored, that is not the case for the DAC. The PCM5122 uses an internal 8× oversampling digital filter that introduces a latency of $20/f_s$, where f_s is the audio signal's sampling frequency. In our case, the sampling frequency was 48,000 kHz, so the filter's latency is about 417 μs. That is in very good agreement with what the measurements, as we estimated the hardware latency to about 500 μs.

5.3 Responsiveness

It is very hard to quantify how a drummer feels about playing on a particular drum kit. This is why, during the following tests, the drum module was connected to a Roland TD4-KP drum set[22]. The advantage of the TD4 module is that it has a single connector to the sensors, and it is a common DB-25. It is thus easy to connect an eXaDrums module to it.

The tests were conducted with three pads: the hi-hat, the snare drum, and the bass drum. The outstanding responsiveness of the module was quite remarkable and playing drums with an eXaDrums module felt exactly like playing with a TD4 module, if not better. No latency could be perceived, and ghost notes were perfectly rendered, which is the proof of a really good sensors conditioning ADC's specifications.

5.4 Rhythm coach

EXaDrums also provides a metronome and a rhythm coach, to help drummers improve their skills, or just to warm up. The rhythm coach is still in development, but some common features are already available. It is coupled to the

metronome and can measure the time difference between the click and the time a stick hits a drum pad. We will call that time difference the jitter.

The most important thing about the rhythm coach is to be sure that it is almost perfectly synchronized to the metronome's sound. However, due to the induced sound card's latency, it is very difficult to fulfill that condition. An efficient synchronization mechanism has been integrated to the module, in order to solve that problem.

To validate the efficiency of the synchronization, it was very important to measure the time difference between what the rhythm coach indicates, and the actual jitter of the drummer. To that end, an oscilloscope has been connected to the piezoelectric sensor on CH1, and to the output of the sound card on CH2. The metronome was enabled, and the rhythm coach was running. Every time the drum pad was hit, the delay that was given by the coach was recorded[3], and it was also measured with the oscilloscope. The two values were compared and the test was repeated several times. The results vary, and we observed a difference of up to 1 ms between the value that was read on the oscilloscope and what the rhythm coach gave us. This was expected, as the jitter comes from the OS and software's variable latency contribution.

We believe that a drum module should do more than just mimic acoustic drums. Tools like rhythms coaches, metronome, training program, etc. really make a huge difference. In the future, we hope to see very complete drum module that exploit all the new technologies that are available.

6 Conclusion

We have presented a very modern kind of homemade drum module. As hobbyists, it is our role to keep up with commercial products, and evolve as new technologies become available to us. This is why eXaDrums only uses recent hardware such as the Raspberry Pi 3. Compared to microntrollers, the available memory makes SBC drum modules much more powerful and configurable, as the user can save a virtually unlimited number of drum kits and sounds samples. Despite some challenges, we have shown that this solution is viable and led to very good results in terms of latency, sound quality, and user interface.

The user experience has been the most important development criteria. With a modern drum module, it must be possible to download drum sounds samples and customize every instrument of each custom drum kit. All those features need to be packed in a flexible and robust software.

Moreover, the rhythm coach that has been developed for eXaDrums shows that drum modules can offer way more than just good acoustic drums replicas. For instance, they can include a personal trainer of some sort and help drummers learn or teach drums to beginners. Drummers need to be able to keep track of their performance and progression, plan new custom training sessions, and see how they can improve their skills.

The future is about data and great user interfaces because those are two most important things to the users. Drum modules have to evolve in that way and provide better graphical user interfaces, but also better learning and training tools.

[3]The value is not available through the GUI, but can be obtained using the API.

A Voltage mapping: how to choose the resistors

The following source code has been used to determine the resistors' values in the voltage mapping circuit. Three voltages need to be provided: the maximum value of the input voltage, the maximum value of the desired output voltage, and the voltage bias. The value of the resistor R_0 has to be provided too. Results are stored in the variables A, B, and C, and correspond to the values of the resistors R_1, R_2, and R_3, in ohms, respectively.

```
VinMax = 5;          % Peak value of the input voltage
VoutMax = 3.3;       % Maximum desired output voltage
Vbias = 5;           % Bias voltage (either 3.3V or 5V)
R0 = 100e3;          % Chosen value for R0 in ohms

% Calculations

DVin = 2*VinMax;
DVout = VoutMax;

Vin = -VinMax;
Vout = 0;

AL = DVin / DVout;
BL = Vbias / (Vout - Vin/AL);
CL = AL*BL / (AL*BL - AL - BL);

% Results

A = R0               % R1 = R0
B = R0 / AL * BL     % R2
C = R0 / AL * CL     % R3
```

References

[1] Pollard Syndrum *Wikipedia*, https://en.wikipedia.org/wiki/Pollard_Syndrum.

[2] Simmons SDS 5 *Wikipedia*, https://en.wikipedia.org/wiki/Simmons_SDS-V.

[3] Forat F16, *Wikipedia*, https://en.wikipedia.org/wiki/Forat_F16.

[4] eDrum *Admir Salahovic*, http://www.edrum.info.

[5] Pic Microcontroller *Wikipedia*, https://en.wikipedia.org/wiki/PIC_microcontroller.

[6] Alen Šiljak *Open Source Drum Module*, https://sites.google.com/view/open-source-drum-module.

[7] Massimo Bernava *MicroDrum*, http://microdrum.altervista.org.

[8] Wyatt Olson *Drum Maser*, http://drummaster.digitalcave.ca.

[9] Dmitri Skachkov *MegaDrum*, https://www.megadrum.info.

[10] HydroDynamo Inventor *Instructables* http://www.instructables.com/id/Raspberry-Pi-Drum-Kit

[11] David Pride *Raspberrypi.org* https://www.raspberrypi.org/blog/raspberry-pi-air-drum-kit

[12] Pearl Drums *Pearldrum.com* http://pearldrum.com/products/kits/electronics/mimic-pro-module

[13] Jeremy Oden *eXaDrums* https://hackaday.io/project/9350

[14] Jeremy Oden *RaspiDrums* https://hackaday.io/project/7499

[15] Wagner, A., *Analysis of drumbeats—interaction between Drummer, Drumstick and Instrument*, KTH Computer Science and Communication, 2006.

[16] O'reilly, R., Khenkin, A. and Harney, K. *Sonic nirvana: Using mems accelerometers as acoustic pickups in musical instruments.* Analog Dialogue 43

[17] Alex Eames *RasPIO Analog Zero* http://rasp.io/analogzero

[18] Microchip *MCP3008* http://www.microchip.com/wwwproducts/en/MCP3008

[19] Jeremy Oden *libeXaDrums on Github* https://github.com/SpintroniK/libeXaDrums

[20] Jeremy Oden *eXaDrums on Github* https://github.com/SpintroniK/eXaDrums

[21] Matthew Wright and Ryan J. Cassidy and Michael Zbyszynski *Audio and Gesture Latency Measurements on Linux and OSX*, IMC, 2004

[22] Roland *Roland.com* https://www.roland.com/us/products/v-drums_portable_td-4kp