The XY Drum

Anonymous MIT student

December 10, 2009

Abstract

Standard electronic drum pads are equipped with one piezoelectric sensor that converts stick strikes to electronic pulses which are measured and used to trigger pre-recorded samples or other sounds. Advanced models can measure the amplitude of this pulse and adjust the volume of the resultant sample or even select a different sample entirely. The state of the art electronic drums are equipped with several 'trigger zones' on each head that allow for two to four different samples to be mapped to different parts of the drum. The XY Drum aims to offer a new level flexibility to electronic drum designs by providing high resolution x,y strike position for each trigger event. Furthermore, the XY Drum uses this information to enrich the interface of electronic rhythm, affording an entirely new playing experience. This paper motivates a fundamental approach to capturing x-y information using several piezoelectric sensors simultaneously and demonstrates its viability. Furthermore, an early prototype of the XY drum is described. However, further obstacles in the analog design of the system must be overcome before this prototype will be able to successfully generate the desired x-y strike positions.

1 The XY Drum

The XY Drum is a nonprogrammable electronic drum head that captures stick strike velocity and x,y position on the pad. While information alone encapsulates a rich rhythmic interface, users are left to decide how best to use this information. Simple synthesis firmware for generating four sinusoidal voices are provided, but the mapping between these oscillators and

the trigger events are intentionally left blank so that users can experiment with how to use this new interface. This mapping is specified via a simple software interface that is uploaded to the XY Drum over USB. The classic use case, although this has not yet been demonstrated, is to map frequency on the vertical axis and decay on the horizontal. Thus pure tones at certain frequencies and with certain envelopes can be produced by striking the pad at varying positions. Early data suggests that users should have no problem triggering several notes simultaneously. This is possible because no player is capable of producing truly simultaneously strick strikes. Incident impulses are measured with a phase resolution of 20MHz (wave peaks can be differentiated as long as they are 50ns apart). Furthermore, positioning information depends on 1ms timing differential between wave peaks, which is about the time it takes for sound to travel across the pad. Thus, unless a player can play simultaneous strokes accurate to the millisecond, the system should gracefully handle the gesture as two distinct strikes.

2 Measuring the Timing Delta

The first step towards building the XY Drum was to verify that timing information could be used to triangulate stick strike position. More complex methods of measuring this information are possible. For example, field interference effects (technique used by most tablet displays) would be effective, but would require special sticks with inductive tips. Resistive arrays are also a possibility, and this approach is backed by several patents for x,y sensitive drum pads (none have come to commercial fruition). This mechanism is mechanically unsuitable, however, as the resistive top layer that does the sensing tends to ruin the desired strike dynamics that players prefer. Amplitude triangulation is ruled out because it depends on the pad dampening wave pulses as they transmit across the surface. This dampening effect would make it impossible to recover velocity information about the strike, since amplitude would necessarily depend on strike position. Measuring the timing delta between wave peaks is subject to non of the above issues. However, this approach requires high frequency sampling not available on most electronic drum interfaces. In fact, most drum triggers are samples at the audio rate of 44KHz, or sometimes slower, since that already exceeds the accuracy of the player. In high quality drum pads such as those produced by Roland, the piezoelectric sensor is coupled to the drum head with a thin metal interface.

This decrease any dampening, but sound travels very quickly through this material. Depending on the metal, the speed of sound is usually near 3000 m/s. Thus wave pulses take about 3us to travel across a 10cm pad. While it is technically possible to capture 100 million samples per second (MsPs), the associated electronics cost would be too high (more than 200$). In a plastic 'rock-band' style drum, sound travels substantially slower. Experimental results show that wave pulses take a full 1ms to travel across the pad. Unfortunately, this comes at a cost of substantial dampening, making velocity retrieval difficult. However, this timing delta can be trivially measured to produce ¡ 1mm precision in x,y triangulation, which is far less than players can accurately differentiate. Rather than sample the piezoelectric sensors and search for wave peaks, the signal is amplified so that any tap will rail to the gain ceiling of 3.3V. This means that an ADC is not needed to measure strike position, as this 3.3V can be used to trigger a digital interrupt on most micro controllers. Velocity measurements are impossible in this configuration, but can be easily obtained by sampling any of the 3 triggers (or all three) and measuring the peak amplitude of impulses before they have been amplified.

3 The Hardware

The piezoelectric sensors source the data pipeline, measuring stick strikes AC pulses. In order to generate x,y information, there are actually three identical data pipelines sourced from three separate sensors. From the sensors, the signal is rectified and amplified and then passed into an STM32F103 Microcontroller. This ARM Cortex M-3 Processor was chosen because it has the requisite speed to measure the timing deltas while simultaneously synthesizing sound or replaying samples. Furthermore, the simple programming interface to this device makes it perfect to allow users to reprogram the XY Drum to suit their sonic needs. The processor generates an acoustic waveform that is modulated using PWM. This signal is then low pass filtered and amplified before being passed to an integrated loudspeaker. In this way, the XY Drum can operate as a freestanding device (once programmed) with no need to be attached to further hardware to be used as a live percussive instrument.

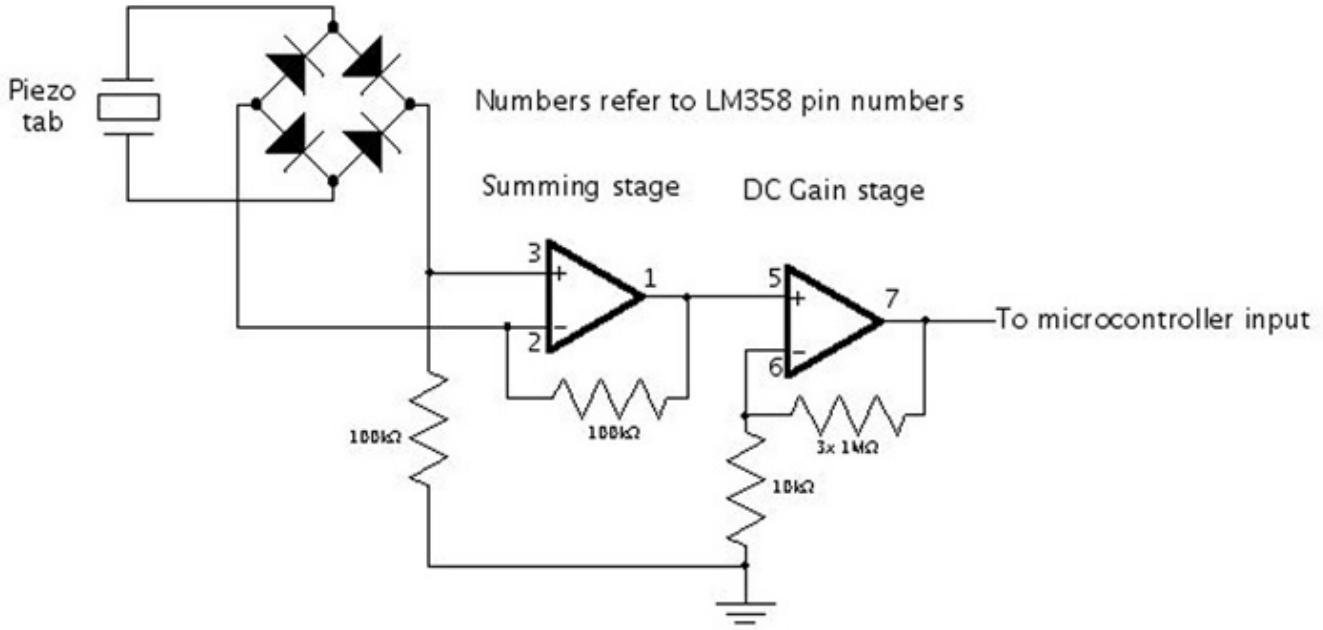
3.1 Conditioning the Signal

The piezoelectric sensor generates AC signals, and depending on the physical dynamics of drum pad there are no guarantees that the first wave pulse will be measured as positive or negative. Thus the first stage in the signal pipeline is a full-wave rectifier. This circuit leaves positive pulses untouched while multiplying negative signals by -1. This filter comes at a cost, however, reducing the peak voltage by .7V as required by the operation of the diodes. Since both the positive and negative terminals of the sensor are technically being measured as part of the signal the sensor is decoupled from ground. This make the long leads in the circuit extremely sensitive to periodic noise, namely the 60Hz electrical background noise common in most environments. Filtering out this 60Hz noise while maintaining the original signal proved difficult and remains one of the principle roadblocks to a successful prototype. In order to make the drum sensitive to even light taps, this rectified signal must then be amplified. Ordinarily, light taps produce a .1V swing, while hard strikes can produce wave peaks over 2V. The 60Hz background noise hovers at around .05V in amplitude, which allots extremely narrow margins between amplifying the .1V taps such that they exceed the digital transition voltage (around 2V) without amplifying the 60Hz noise as well. The microcontroller will usually respond to input voltages above 2V, so the goal of amplification is to achieve a precise gain of 200 so that .1V taps result in a trigger, but .05V noise does not. In fact, this precision was never achieved, and the current state of the XY Drum requires strong stick strikes in order to cause a trigger event.

3.2 The Maple Microcontroller

The Maple Microcontroller is effectively a usb programmable ARM Cortex M-3 processor with a lightweight library that exposes much of the processors functionality. Two core features of this processor are used. The first is the 50MHz GPIO bus, that is configured to trigger separate interrupts whenever one of the three input channels rise above 2V. In practice, these GPIO pins are only accurate to 20MHz, although this did not adversely affect the performance of measurement. Second, the integrated PWM peripheral is used in order to generate the output signal without the need for a DAC. PWM is common practice that encodes data in the duty cycle of a fixed frequency square wave. In the XY Drum, the output PWM channel is configured to

Figure 1: Recitifcation and Amplification



produce a square wave at 1MHz, where the duty cycle can vary from 0% to 100% in 255 discrete steps. Generating this signal only requires one instruction per microsecond, since once the PWM channel is loaded with a duty cycle, it generates the square wave without use of the processor. In practice, this resource is more than sufficient to create high quality sine tones up to and beyond the range of human hearing. The channel is, however, a little audibly noisy due to the high frequency components of the carrier square wave, and this noise is filtered out with a simple RC Low Pass filter.

4 Software

The software is composed of three principle components. The external interrupts, the main state machine, and the signal multiplexer. Each of these simple structures are described below.

4.1 Detecting Position With Interrupts

Interrupts are software routines that are separate from the main loop of the program. These routines must be "triggered" by some external event. Once triggered, the processor immediately stops executing whatever routine is currently active, branches to the interrupt, and then returns to the routine that was interrupted. Interrupt routines and main loop routines communicated via shared global variables. Each of the three conditioned piezoelectric sensors drive identical but independent external interrupt channels. The psuedocode for these interrupt service routines are as follows:

```
global int x,y;
global bool trigger;
exti_isr_sensor0() {
           disableAllInterrupts();
           int count_sensor0=0;
           int count_sensor1=0;
           int count_sensor2=0;
           int count=0;

           while (1) {
             count++;
             if (readPin(sensor1)) {
```

```
            count_sensor1=count;
            break;
          } else if (readPin(sensor2)) {
            count_sensor2=count;
            break;
          }
        }

        if (count_sensor1 != 0) {
           while (1) {
             count++;
             if (readPin(sensor2)) {
               count_sensor2=count;
               break;
             }
        } else {
           while (1) {
             count++;
             if (readPin(sensor1)) {
               count_sensor1=count;
               break;
             }
        }

        /* count_sensor0 is always 0, since we interrupted on it */
        x = triangulateX(count_sensor0,count_sensor1,count_sensor2);
        y = triangulateY(count_sensor0,count_sensor1,count_sensor2);
        trigger=TRUE;

        enableAllInterrupts();
}
```

4.2 Generating Sound with PWM and Wavetables

In order to avoid unnecessary overhead in the size and complexity of the embedded executable, common programming libraries are omitted, such as libc. This means that convience functions like sin and exp are simply not available. To counter this, 4096 individual samples of $sin(t)$ and e^{-x} were

generated in python, the values were scaled to vary from $[0, 255]$ in both cases. These values were formatted in appropriate ASCII text and simply copy-pasted into the source code for the XY Drum. The wavetable for sin captures a single period, and the wavetable for e^{-x} was manually windowed to capture a decay from 254 down to 1. In this way, $sin(x)$ (or e^{-x}) can be computed via:

```
u8 sin(u32 x) {
   x = x%4096;
   return sin_table[x];
}

u8 exp(u32 x) {
   if (x>4096) {return 0;}
   return exp_table[x];
}
```

4.3 Coding a Drum

With these components, the simple drum state machine can be constructed. In order to minimize processing overhead, only four voices are provided, each defined by an amplitude, a frequency, and a decay rate. This limitation easily satisfies the processing constraints of the microcontroller. In fact, the limits were chosen conservatively. However, since the PWM signal is generated by a separate processing peripheral and uses only 1 instruction per microsecond, and the ISR is only triggered on very infrequent (compared to the processor speed) strike events, nearly all of the 72 million instructions per second that are available are unused. Thus a four voice limitation is extremely conservative. In fact, there are enough left over cycles to mix together dozens of samples simultaneously or add simple dsp filtering (like reverb or delay) to the signal. These flourishes are beyond the scope of this project, however.

```
void main() {
   u16 voices_time[4];
   u8  voices_freq[4];
   u8  voices_decay[4];
   u8  voices_amplitude[4]; /* not used */
```

```
    u32 signal; /* the signal is really only
                8 bits, so we have LOTS of
                room for computational overhead */

    cvi=0;      /* current voice index */
    while (1) {
      if (trigger) {
        trigger=FALSE;
        voices_time[cvi]  = 1;
        voices_freq[cvi]  = y;
        voices_decay[cvi] = x;
        if (cvi++ > 3) {cvi=0;}
      }

      signal = 0;
      count = 0;
      for (int i=0; i<4; i++) {
        if (voices_time[i] != 0) {
          signal += sin(voices_time[i]*voices_freq[i])*
                    exp(voices_time[i]*voices_decay[i]);
          count++;
          if (voices_decay[i]*(voices_time[i]++) > 4096) {
            voices_time[i] = 0;
          }
        }
      }
      signal = signal/(255*count); /* normalize out that
                                   exp() goes to 255, and
                                   we have 'count' voices */
      setPwm(output,signal);
    }
}
```

There are a few features in the above pseudocode worth discussing. The first is that voices who have run to the end of their decay ($deay * voicesTime$) are no longer added to the signal and are not incremented. They are dead, only to be revived by future trigger events. Also note that this main loop is not of static runtime! Depending on the conditionals, this loops may have

fewer or greater number of instructions (for example, how many voices are currently active). This means $setPwm()$ will be called at a higher frequency, and voicesTime will be similarly incremented at a higher rate. Thus, the more voices are active the lower all the frequencies are! In experimenting with this code, all sorts of interesting bugs can be uncovered by their interesting (sometimes desirable) sonic properties, which certainly adds to the fun of playing with the XY Drum. In the above example, the signal is given enough headroom to avoid any clipping or rollover effects. However, this is another source of endless variation, that can add unpredictable noise and other sounds into the signal. Many of these rollover errors are extremely interesting, if not always reproducible. All of these bugs can be eliminated, if desired. For example, the timing variation can be eliminated by scheduling this signal generation routine as a fixed-time interrupt. So that this logic only run every 1MHz or 100KHz, as opposed to "as fast as possible".

5 Putting it all Together

Without the proper materials, a rugged frame and CAD designed, laser cut, computer milled components were simply unavailabe. The original prototype was build from wood and tape. Glue was not used so that components could be dissasembled later. Unfortunately, the XY Drum is currently dissasembled for further development and only these "simulated" pictures are available.

6 The Result

The principle obstacle still standing is the proper conditioning and amplification of the sensor signal. This can likely be fixed via a tune filter to remove the 60Hz background noise from the signal. However, the amplitude of the noise is simply too near to the amplitude of the signal we want to capture to make this gain stage a trivial one. So far, all attempts to make the readings more sensitive using gain have resulted in a signal that is either only responds to strong stick strikes or is pretty much continuously firing the external interrupts due to amplified noise. Furtermore, the piezo sensors themselves have an extremely large parasitic capacitance, which tends to slowly bias the amplifier over time such that even when the gain is properly tuned, the signal quickly fades to the positive rail and fires the interrupt for no reason.

This effect isnt totally understood, however, a sophistocated gain circuit and some properly placed capacitors and inductors to match the impedance of the sensor should probably fix the problem. Also remaining is the brunt of the triangulation routines. While quadrant detection has been demonstrated with great accuracy, precise x,y measurements have not yet been obtained. This defficiency should be easily corrected with added software and it is not the result of any hardware problems. However, in order to obtain accurate timing delta's between wave peaks, the signal must be properly rectified, which reduces the signal by .7V and therefore requires more gain. It is precisely the gain stage that has been proving difficult to implement correctly, thus indirectly hindering the progress of the x,y triangulation mechanism. The ability of the STM32 to generate high quality audio was expected, but it was still a surprisingly successful result. Carelessness in handling timing variability of program logic as well as managing variable overflow and wrap around caused some very interesting, often desirable sonic effects. Overall, the initial synthesizer was unpredictable but effective. However, these problems have since been corrected and pleasing pure tones in four part harmony have been achieved. The addition of a low pass filter on the output stage further reduced the noise on the output.

6.1 The Future

Once a suitible gain circuit is identified for signal conditioning, the final steps towards an accurate and sensitive x,y drum trigger pad are minimal. From there, future improvements will almost certainly include:

1. Velocity sensitivity via sampling the analog values of the sensors
2. A new chasis, designed to harness the components and organize the wiring rather than the ad-hoc duct time design currently employed
3. The ability to upload samples and map them to locations on the trigger pad
4. The ability to map strike location to filter parameters like reverb or eq
5. A physical interface to select between operating modes, this might include a set of buttons and a small OLED screen

6. DIY pictures and instructions to go along with the project. Unfortunately timing and resource constraints prevented adequate visual documentation of the project. If you have seen a white IPhone (with camera), please return it to ajmeyer@mit.edu

21M.380 Music and Technology (Contemporary History and Aesthetics)

Fall 2009