A Sample Accurate Triggering System for Pd and Max/MSP

Eric Lyon
Sonic Arts Research Centre
School of Music and Sonic Arts
Queen's University Belfast
e.lyon@qub.ac.uk

Abstract

A system of externals for Pd and Max/MSP is described that uses click triggers for sample-accurate timing. These externals interoperate and can also be used to control existing Pd and Max/MSP externals that are not sample-accurate through conversion from clicks to bangs.

1 Introduction

In the world of experimental electronic music, regular pulsation has often been frowned upon. During an exchange of ideas between Karlheinz Stockhausen and several younger electronic musicians (Witts 1995) Stockausen observed, "I heard the piece Aphex Twin of Richard James carefully: I think it would be very helpful if he listens to my work Song Of The Youth, which is electronic music, and a young boy's voice singing with himself. Because he would then immediately stop with all these post-African repetitions, and he would look for changing tempi and changing rhythms, and he would not allow to repeat any rhythm if it were varied to some extent and if it did not have a direction in its sequence of variations." Richard D. James responded from a different perspective, "I didn't agree with him. I thought he should listen to a couple of tracks of mine: "Didgeridoo", then he'd stop making abstract, random patterns you can't dance to."

1.1 The Need for Better Timing

The same year this interview was published, I attempted to use a pre-MSP version of Max to control a drum machine I had built in Kyma. This experiment also resulted in "random patterns you can't dance to," since the Max event scheduler at the time was good enough for certain kinds of algorithmic music, yet not stable enough for techno music. Irrespective of aesthetic issues, the precision required for creating techno music can reveal limitations in some of our favorite computer music systems that we might not have otherwise noticed. Ten years later, the event schedulers for both Max/MSP and Pd are much more stable, and are quite usable for some forms of music based on regular pulsation. However, their performance is still subject to variability based on factors such as the signal vector size and competition from control-level events. Furthermore, the scheduling systems of Max/MSP and Pd differ such that the timing behavior of similar patches can perform quite differently between the two systems. The Max/MSP event scheduler is prone to permanently drift from a sample accurate measurement of timing. The underlying Pd event scheduler is sub-sample-accurate using 64-bit floating point numbers to represent time, though apparently at the cost of a higher likelihood of interruption of the audio scheduler, resulting in audible glitches. In both systems temporal accuracy of control-level events can drift freely within the space of a signal vector.

1.2 The Problem with Small Time Deviations

Even when the amount of deviation from sample accuracy is not clearly noticeable at a rhythmic level, it may still have undesirable musical effects. For example, a pulsation may feel not quite right when there are a few 10s of milliseconds of inaccuracy in the timing from beat to beat. Smaller inaccuracies, though rhythmically acceptable, can still cause problems when sequencing sounds with sharp transients, since changes in alignment on the order of a couple of milliseconds will create different comb filtering effects as the transients slightly realign on successive attacks. This artifact is particularly insidious since many users might not think to trace a spectral effect to a system timing flaw. This and some other artifacts have already been discussed in the context of MIDI bandwidth limitations (Moore 1988). Thus low level temporal indeterminacy subtly degrades not only the sense of machine-generated precision required by techno, but also its seeming opposite, namely the human performance expressivity which is the main concern of Moore's paper.

2 Sketch of a Solution

One way to sidestep the abovementioned problems is to implement trigger scheduling at the sample level, rather than at the event level at which bangs perform. This scheduling must be built into every external that is to benefit from sample accurate triggering. In order to be of much use, such externals must be able to easily synchronize with each other. I have developed a system of externals based on click triggers. The trigger signal contains a non-zero value at every sample where a trigger is to be sent, and zeros at all other samples. The value of the click trigger can convey one additional piece of information to its receiver, such as desired amplitude.

3 Sample Accurate Metronomes

The centerpiece of the system is an external that coordinates multiple metronomes. It is called `samm~` (for sample accurate multiple metronomes). The first argument to `samm~` is the tempo in BPM, followed by a series of beat divisors that each define the metronome speed for a corresponding outlet. For example, the arguments 120 1 2 3 7 would activate four outlets, all beating at 120 BPM, the first at a quarter note, the second at an eighth note, the third at an eighth note triplet and the fourth at a sixteenth note septuplet. Any of these parameters can have fractional components, and the beat divisors may be less than 1.0, resulting in beat durations greater than a quarter note. A click trigger from `samm~` is always a sample with the value 1.0. The tempo can be varied during performance while preserving proportional relations among all beat streams.

3.1 Alternative Metronome Specifications

For convenience, several different methods are provided for specifying metronome tempi. A new set of beat divisors may be specified with the message ”divbeats.” Beat durations may be specified directly in milliseconds with the message ”msbeats.” Beats may be specified in samples (useful if tempi need to be built around a soundfile in terms of its length in samples) with the message ”sampbeats.” Finally beat durations may be specified with ratio pairs (with the denominator representing a division of a whole note) using the message ”ratiobeats.” The message ”ratiobeats 1 4 3 16 5 28” specifies relative beat durations of respectively a quarter note, a dotted eighth note and five septuplets. Fractions may be employed to represent more complex ratios, though it is probably simpler in that case to represent such ratios with decimal numbers and use the ”divbeats” message.

4 Pattern Articulation

The beat streams from `samm~` can be patterned into arbitrary rhythms with another external called `mask~`. This external stores a sequence of numbers, which are sent out in cyclic series in response to click triggers. An initial series is given as a set of arguments to `mask~`. For example, the arguments 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 could serve to define a rhythmic pattern for a single instrument in a drum machine, in this case perhaps a kick drum. Since zeros cannot trigger an attack, any zero in a `mask~` pattern will convert an incoming beat to a rest. Since any non-zero number can serve as a trigger, the attacks need not all have value ”1” but could specify different amplitudes instead. Multiple `mask~` instances could control different parameters of the same event, all sample-synched. Since `mask~` patterns can be of any size (up to 1024 members), different sized `mask~` patterns will cycle in and out of phase with each other, which is desirable in a poly-metric scheme. It is also possible for two `mask~` patterns of the same size to be out of sync with each other if, for example, one `mask~` was created later in the design of a given patch. This loss of sync is usually not desirable. Thus, `mask~` provides an option whereby the input is interpreted not as triggers, but rather as index numbers used to iterate through the `mask~` pattern. Using the same indexing clicks (generated from another `mask~` of course) guarantees that all patterns so controlled remain locked in phase. Any `mask~` external can hold a large number of different patterns which may be stored and recalled during performance.

4.1 Sample Accurate Synthesizers

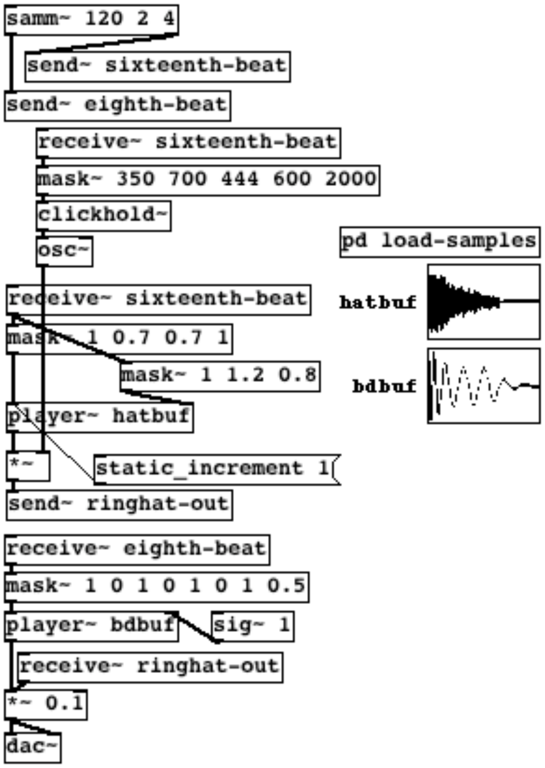
Sample accurate externals are provided for sound production through both sampling and synthesis. `adsr~`, an external that is already part of my Web-published Max/MSP external set LyonPotpourri (Lyon 2003) is an ADSR envelope generator. I have retrofitted `adsr~` to respond to click triggers, interpreting the value of the click as the overall amplitude of the envelope. Any software synthesis algorithm that uses `adsr~` as an envelope can now be triggered with sample accuracy.

4.2 Sample Accurate Samplers

A sample playback external called `player~` is provided, which plays back a sample stored in a buffer (for Max/MSP) or an array (for Pd). The two parameters to `player~` are amplitude and playback increment, sent as signals to the first and second inlets respectively. Amplitude is always a click trigger. Under normal conditions, playback increment is a signal that can be manipulated during performance. An alternative static increment mode is provided, called by the 'static_increment' message, in which the playback increment is also received as a click trigger, which persists throughout the playback instance without variation.

4.3 Polyphonic Nature of player˜

An inconvenient feature of groove˜, tabosc4˜, et. al. is that if a note is currently active when a new playback trigger arrives, the current note is instantly truncated which often creates discontinuities. In player˜, all currently active notes continue playing to the end of their buffers, even as new attack triggers arrive. This is much more convenient than having to create a poly structure for every member of a drum machine. This is also the reason for the static_increment mode. In static_increment mode multiple instances of playback can proceed at different playback increments, which is quite handy for creating polyphony from a single sample. By default, player˜ provides a maximum of 8 simultaneous playback instances, which has been found to be sufficient in most cases. This maximum can also be specified with an optional creation argument to player˜.



```
samm~ 120 2 4
send~ sixteenth-beat
send~ eighth-beat
receive~ sixteenth-beat
mask~ 350 700 444 600 2000
clickhold~
osc~
pd load-samples
receive~ sixteenth-beat
mask~ 1 0.7 0.7 1
mask~ 1 1.2 0.8
player~ hatbuf
*~
static_increment 1
send~ ringhat-out
receive~ eighth-beat
mask~ 1 0 1 0 1 0 1 0.5
player~ bdbuf
sig~ 1
receive~ ringhat-out
*~ 0.1
dac~
```

Figure 1: A two voice drum machine.

5 Putting the Pieces Together - a Simple Drum Machine

Now let's look at an example of how the externals described thus far can be combined. (See Figure 1.) A samm˜ unit with a tempo of 120 BPM creates two beat streams, the first dividing the quarter by two (eighth-notes) and the second dividing the quarter by four (sixteenth-notes). Two player˜ objects play samples stored in two arrays. The bdbuf player˜ takes its metronome from the eighth-note beat stream. Its attack/amplitude pattern is stored in the mask˜ object directly above it. The increment is fixed at 1.0, taken from a sig˜ object. The output is scaled and sent to the DACs.

5.1 Polyrhythmic Sequencing

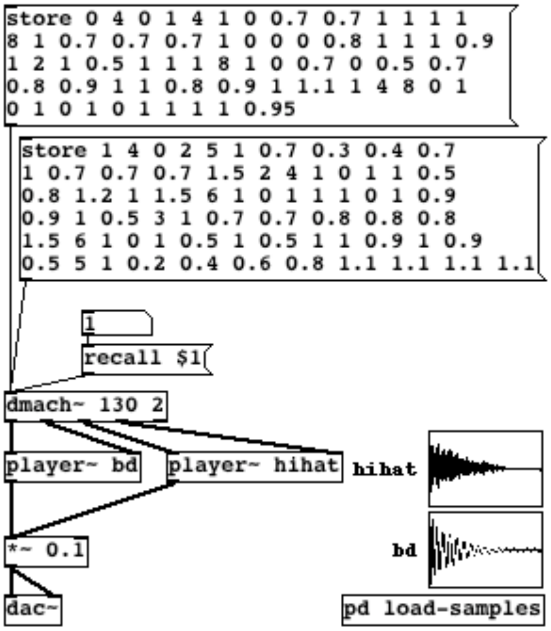
The hihat structure is slightly more complicated than that of the bass drum. The beat stream is sixteenth-notes in all cases. The duration of the attack/amplitude pattern is one beat, rather than the four beats of the bass drum pattern. But a second pattern with a periodicity of three sixteenth-notes controls the increment from a second mask˜ object routed to the second (increment) inlet of the hatbuf player˜. A third rhythmic level is added as the hihat output is ring-modulated by the sine wave output of an osc˜ object. The osc˜ is controlled by a frequency pattern with a periodicity of five sixteenth-notes. A custom object, clickhold˜is inserted between the mask˜ and the osc˜ to sample and hold each click as it comes in, resulting in the sustained frequency signal required by osc˜. As the three different patterns go in and out of phase with each other, a composite 15-beat pattern emerges. More complex polyrhythmic arrangements are easily imagined, especially to control parameters of rich synthesis algorithms, rather than the relatively few parameters of sample playback.

5.2 You Call *That* a Drum Machine?

It is quite clear that the Pd patch shown in Figure 1 does not look anything like a conventional drum machine. Of course it is possible to use some graphical objects to wire up an interface that that looks more like a drum machine and serves as a front end, generating patterns for the mask˜ objects. But this sort of tidy front end would also limit our possibilities. The very looseness of the scheme of distributed mask˜ objects suggest more fluid ways of thinking about drum patterns, and manipulating them during performance.

6 dmach˜ - an Integrated Drum Machine External

The combined use of samm˜ and mask˜ can create arbitrarily complex rhythms. However certain kinds of rhythms are somewhat inconvenient to specify under this model. Consider a 4/4 bar pattern where the first beat is divided into 16th notes, the second beat into triplets, and the last two beats divided into eighth-note quintuplets. A representation of this pattern requires three different beat streams and three different mask˜ objects. In order to address this problem, a proof-of-concept external called dmach˜ has been designed. dmach˜ contains an internal clock, and stores user-specified patterns. The patterns are sent as outlet pairs; dmach˜ is designed to send both attack patterns and increment patterns. These patterns can be recalled at will during the performance. The current pattern is looped until such time as a new one is recalled. Patterns are stored with the 'store' message and recalled with the 'recall' message. The current pattern is played to completion before a new pattern is loaded, thus guaranteeing a sample-synced performance. The last outlet of dmach˜ sends a click at the start of each pattern playback, making it easy for the user to build a sequencer for stored patterns.



```
store 0 4 0 1 4 1 0 0.7 0.7 1 1 1 1
8 1 0.7 0.7 0.7 1 0 0 0 0.8 1 1 1 0.9
1 2 1 0.5 1 1 1 8 1 0 0.7 0 0.5 0.7
0.8 0.9 1 1 0.8 0.9 1 1.1 1 4 8 0 1
0 1 0 1 0 1 1 1 1 0.95
```

```
store 1 4 0 2 5 1 0.7 0.3 0.4 0.7
1 0.7 0.7 0.7 1.5 2 4 1 0 1 1 0.5
0.8 1.2 1 1.5 6 1 0 1 1 1 0 1 0.9
0.9 1 0.5 3 1 0.7 0.7 0.8 0.8 0.8
1.5 6 1 0 1 0.5 1 0.5 1 1 0.9 1 0.9
0.5 5 1 0.2 0.4 0.6 0.8 1.1 1.1 1.1 1.1
```

```
1
recall $1
dmach~ 130 2
player~ bd
player~ hihat
*~ 0.1
dac~
pd load-samples
```

Figure 2: A dmach˜ two voice drum machine.

6.1 Pattern Specification for dmach˜

Pattern specification in dmach˜ allows for arbitrary bar sizes and arbitrary subdivisions within the bar. Patterns are entered into dmach˜ with the 'store' message. The first two parameters are the pattern number and the number of beats in the bar. The pattern number is an integer which will be used to recall the pattern. The number of beats is defined as the number of quarter notes in a bar. A specification of 4 creates a 4/4 bar. A 3/8 bar would be specified with a bar duration of 1.5. Following are a series of beat patterns, each one targeted toward a different instrument. The first parameter is the designated instrument. (Instrument 0 has its beat stream come out of the first outlet, and its increment stream out of the second outlet of dmach˜.) Next is a beat duration representing a segment of the bar. If the entire beat pattern shares a single subdivision of the beat, then this segment would simply be the same as the bar duration. However since an arbitrary number of segments can be specified (on condition that they eventually add up to precisely the bar duration), an arbitrary rhythmic subdivision of the bar is possible, which could be quite complex. Following the segment duration is the segment subdivision factor. This subdivision must be an integer. Following this is an attack pattern that must have the same number of elements as the subdivision factor just specified. Following the attack pattern is the increment pattern. The increment pattern has the same number of elements as non-zero attacks in the attack pattern. In other words, increments are specified only for attacks, not for rests. Additional segments are identically specified until the duration of the bar is filled. This process is repeated for every instrumental beat/increment stream required for the pattern. Data is only required for instruments that play in a given pattern. A pattern with no instrument data functions as a bar of rest.

6.2 Usability of dmach˜

As mentioned above, dmach˜ is a proof-of-concept external. Given the intricacy of the data format, it is recommended that a preprocessor be used to generate the data from a more user-friendly interface than typing raw data by hand. It is interesting to consider what might be a suitable graphical interface to design patterns for dmach˜ though such considerations are beyond the scope of this paper. The complexity of pattern specification for dmach˜, while potentially burdensome, is also necessary in order to obtain full rhythmic flexibility. Indeed this flexibility goes considerably beyond what is possible with most commercial drum machines. However as mentioned above, the user can be buffered from this complexity with a suitable interface, at the cost of some loss of flexibility in pattern design. As can be seen in Figure 2, the data complexity is localized in the 'store' messages, so the patching structure for wiring up a drum machine in Pd with dmach~ is somewhat simpler than in the earlier example with multiple mask~ objects.

6.3 Relative Inflexibility of dmach~

While it is convenient to bind increment patterns to attack patterns in dmach~ this arrangement is somewhat inflexible. The user might prefer to not control increment, or to control increment out of sync (or even randomly) in which case the additional outlets for increment become superfluous, as does the burden of specifying increment pattern in the 'store' messages. On the other hand, one might well wish to simultaneously control parameters other than or in addition to increment, such as pan location, filter parameters, ring modulation frequency or multiple synthesis parameters, if a software synthesizer is being driven by a particular dmach~ beat stream.

6.3.1 Increasing Flexibility for dmach~

A simple method to increase the flexibility of dmach~ would use the second beat stream outlet to send attack index numbers which could then be used to control multiple mask~ objects. This would give full flexibility, though the pattern data would in most cases be spread over multuple mask~ objects. In some cases this could be an advantage since individual data streams could be changed independently during performance. A more complicated solution would allow the user to specify the structure of a given dmach~ object through its creation arguments, such that a given beat pattern could have an arbitrary number of outlets in addition to its attack pattern outlet. This would keep all the pattern data in a single 'store' message. However the complexity of maintaining data in this form, along with the possibility of eventually bumping up against the hard limit on the number of atoms allowed in a Max/MSP or Pd message box, might make this solution unwieldy in practice. A sufficiently flexible graphical interface that could create and manage the data in the 'store' messages with arbitrary structure, might make this approach worth pursuing. As mentioned above, dmach~ is still a prototype object, which is not yet ready for prime-time. Nonetheless dmach~ does raise interesting questions about structuring control data within the sample accurate triggering system under discussion.

7 Interoperation with Non Sample Accurate Externals

Many useful externals exist in Pd and Max/MSP which do not currently provide a sample accurate response to triggers. In order to utilize such externals in the system presented here, they must be triggered with a bang synchronized to the incoming click trigger. In Max/MSP this can be done with the edge~ external which sends a bang on detecting a change from zero to a non-zero value in an incoming signal. Since edge~ is only available for Max/MSP, a Pd external called click2bang~ has been designed to send out a bang in response to an incoming click. The bang can only be accurate to within the size of the signal vector. However the receiver can be isolated in a sub-patch with a signal vector size set to 1 by the block~ object, forcing that part of the patch back to sample accuracy.

8 Clicks and Continuity

While click triggers are conceptually simple and thus easy to work with in designing patches, they do have one disadvantage. Once sent there is no further information on the progress of a triggered process until the next click is received. For the types of data discussed here this is not a problem. However certain continuous processes such as filter sweeps might need to be correlated to the progress of a given time span. For most practical purposes a line or line~ object triggered by an appropriate message (itself triggered by a click2bang~) will suffice. However it would be fairly easy to design an external that outputs precise continuous data in response to click triggers. We might call such an external clickline~ which would receive a target value and the duration over which to interpolate from a stored initial value to the target value, with the trigger and other input data sent as clicks.

9 Conclusions and Future Work

The click trigger model has proved easy to implement, useful for designing rhythmic patches in Pd and Max/MSP and enables a degree of timing precision for rhythmic events that is not generally practical for Pd and Max/MSP. I plan to incorporate this model into future externals where appropriate. There has been increased interest in sample-accurate timing within the Max/MSP community . Some more recent Max/MSP object such as sfplay~ and techno~ employ sample accurate triggering, albeit using more complicated methods than the click triggering system described here. It would be nice to see a unified sample-accurate triggering system employed to encompass the many Pd and Max/MSP externals that could benefit from it, such as tabplay~ and groove~. Third party developers of externals might also find this model useful for any of their objects that involve triggering. All of the work described here is based on steady pulses. It would be interesting to develop metronomes that implement arbitrary tempo curves, which would also output click triggers. This would allow for sample-accurate exploration of a very different class of rhythms. The sample accurate trigger streams from `dmach~` could also be intercepted by another external that imposes expressive timing curves or rubato to explore different performance articulations of the same rhythmic pattern.

10 Acknowledgements

An earlier version of this paper was presented at the 2006 Linux Audio Conference. Frank Barknecht provided useful comments as reviewer for that conference. Peter Castine provided helpful information on numerical accuracy in C. Comments from the anonymous ICMC reviewers lead to further clarifications.

References

Lyon, E. (2003). *LyonPotpourri - a Collection of Max/MSP Externals*. http://arcana.dartmouth.edu/~eric/MAX.

Moore, F. R. (1988). The dysfunctions of MIDI. *Computer Music Journal 12*(1), 19–28.

Witts, D. (1995). Advice to clever children/advice from clever children. *The Wire 141*, 33–35.