Teletype concepts and thoughts

teletype

1 August 10, 2020, 10:05pm

there is an interesting paradox about developing firmware for something like teletype - i spend most of my time developing, not using new features. this is not always the case, and there are, of course, features that i develop precisely because it's something i want to use. but even with these i feel like i'm only scratching the surface of what's possible - which is the very nature of teletype, since there are so many ways to use it. considering these possibilities is an important part of development - use cases inform a feature's design. and here lies the paradox: i think of many different concepts but realize only a few. this thread is an attempt to document them so that they don't get lost - and hopefully serve as inspiration and a starting point for discussions. i'll also share some random thoughts on teletype and some tips & tricks.

50 Likes

2 August 10, 2020, 10:06pm

concept: teletype as a module expander / controller

if you asked people what they consider the main use for teletype to be, most would probably say sequencing. and indeed, teletype is arguably the ultimate modular tool for creating sequencers and automata. using it as an expander for another module seems like underutilizing it - but i think it's one of the most exciting aspects of teletype - it allows you to create your own custom expander / controller for a module.

let's use the param knob to control multiple things. typically, to process continuous changes (like the CV input) we want to use a fast metro script:

```
#I
M 25
```

```
#M
L 1 4: CV I PARAM
```

the init script sets the metro rate to 25ms, and the metro script is used to poll the param knob value and pass it to CV outputs. not super exciting, since we could just mult one of the CV outputs. let's change the metro to this:

```
#M
CV 1 PARAM
CV 2 SCALE 0 V 10 V 10 0 PARAM
CV 3 SCALE 0 V 10 0 V 5 PARAM
CV 4 SCALE 0 V 10 V 5 0 PARAM
```

now both CV 1 and CV 2 will output voltage between 0 and 10V (`V 10`), but CV 2 reaction to the knob is inverted - it outputs 10V when the knob is all the way down, and 0 when it's all the way up. additionally, CV 3 and 4 use 0…5V range. you can use any positive range between 0…10V, so you could scale the full knob range to, say, 2V…4V: `SCALE 0 V 10 V 2 V 4 PARAM` . first 2 parameters for `SCALE` are the range you're scaling from (for PARAM knob it's `0` … `V 10`), parameters 3 and 4 are the target range and the last parameter is the value you're scaling.

with this simple script you can now control 4 parameters at once with one knob. say, you could use it with mimeophon to increase halo and repeats while simultaneously decreasing color.

this is one of those things that is difficult to do in modular but incredibly easy to do with teletype. let's use mimeophon again as an example. it has inputs for flip and hold which means we can automate them. let's say we want to clock mimeophon from teletype and we want flip and hold to get engaged on beat 2 and 4 respectively:

```
       1234
clock: ----
flip:  -#--
hold:  ---#
```

mimeophon uses triggers for flip and hold, so what we need to do is: send a trigger to flip on clock pulse 2 and 3 and send a trigger to hold on clock pulse 4 and 1. here is the script:

```
#M
TR.P 1
D + D 1
IF > D 4: D 1
IF OR EQ D 2 EQ D 3: TR.P 2
IF OR EQ D 1 EQ D 4: TR.P 3
```

`D` is the beat counter, trigger output 1 is the clock, trigger output 2 should be connected to flip input, trigger output 3 to hold.
ideas to try: adding some probability and random: `IF TOSS: TR.P 2` etc

this is an example of how we can build a synchronized action - now let’s try and create a complex gesture. something like this:

- turn flip on for 1 second
- increase halo gradually during that second
- in the 2nd half of that second also turn hold on

here is the script (and a good example of how easy it is to just do things literally with teletype):

```
#1
TR.P 2
DEL 500: TR.P 3
DEL 1000: TR.P 2; TR.P 3
CV.SET 1 V 10
CV.SLEW 1 1000; CV 1 0
```

`CV.SET` is a good op to use when employing CV slewing - it ignores the current slew setting and sets the voltage immediately.

we now have a complex gesture mapped to trigger input 1 - which we can trigger either manually from keyboard, or by connecting a trigger source to trigger input 1. and we still have 7 scripts left (8 with metro) to program other gestures. we could trigger these gestures from grid. add these lines to the init script:

```
M.ACT 0
G.BTX 1 0 6 2 2 0 4 9 8 1
```

first, stop the metro script, then create 8 grid buttons and assign them to the metro script. and in the metro script we simply do this to trigger the appropriate script:

```
#M
IF G.BTNV: SCRIPT G.BTNI
```

i won’t go into details on grid ops here - check the if you want to learn more.

what else can we do? we have 128 buttons and are only using 8, since we are limited to 8 scripts / gestures. but we could add some variations by utilizing all 128 buttons like this:

```
#I
G.BTX 1 0 0 1 1 0 0 9 16 8

#M
IF ! G.BTNV: BREAK
A G.BTNX
SCRIPT + 1 G.BTNY
```

now any button in row 1 will trigger script 1, any button in row 2 - script 2 etc. but depending on where in the row you press, you’ll get a different value for A (it’ll be set to the X coordinate of the button pressed, so between 0 and 15). let’s modify script 1 as follows:

```
#1
J SCALE 0 15 100 2000 A
TR.P 2
DEL J: TR.P 3
DEL * J 2: TR.P 2; TR.P 3
CV.SET 1 V 10
CV.SLEW 1 * J 2; CV 1 0
```

so now even though you have all 16 buttons in row 1 trigger the same gesture (script 1), the response time will depend on where in the row you press - this is done by using the X coordinate to calculate the delay time.

now if we add scripts 2-8 we will have an expander for mimeophon that is capable of executing complex gestures which can be triggered by other modules or manually. to replicate just one gesture we used here would require an envelope generator, a clock divider and some logic modules. teletype replaces all of that - and in a way that allows you to easily modify and recall easily.

@scanner_darkly_

i've started a thread on my thoughts and ideas on how to use teletype (link in...

to summarize: teletype is a great way to extend functionality of pretty much any module as it allows you to easily program complex actions with precision. not just that, but you can also use it to map external controllers (MIDI or grid) - which is a good topic in itself and one i will cover in a future post.

ideas to try: teletype as an expander doesn’t have to be one way - you can program gestures and trigger some of them from the module you’re using with teletype. for instance, you could create a magneto expander and then use the magneto clock outputs to trigger scripts 1-4 for some synced gestures that would automatically follow the clock.

49 Likes

---

3 August 11, 2020, 12:06am

I think the closest to an “expander” concept I’ve come with Teletype is an FM ratio selector for Hertz Donut mk2. I set it up so I could set the modulation oscillator to follow mode but turn down its frequency fully, and just patch CV1 from Teletype and select the ratio with the Param knob.

I had to determine the values for the pattern experimentally… which was easy enough to do, using a fader from 16n to set the CV and copy it to A, so I could watch it in real time. I could have gotten fancy and saved the value to a pattern with a trigger, but I decided to type it manually 🙂

…

Another thing I like to do is set a range of pitch values on faders 8 through 16, and use the M script to read fader 7 to select which of those 8 to use for pitch at any given time (quantized or not according to taste).

9 Likes

---

4 August 11, 2020, 1:46am

What an incredibly exciting thread. Following with great interest and want to also thank you so much for the range of tireless work that you’ve already completed on the platform.

7 Likes

---

5 August 14, 2020, 3:52am

concept: preset morpher

inspired by the macro control script above. quick recap: we can use the param knob to control the CV outputs in a more continuous fashion by using the metro script with a high refresh rate, like 25ms: `M`

`25` . you can experiment with even higher rates by using the `M!` op as `M` caps it at 25 - just expect that teletype might start acting a little weird.

this was the script from the previous post, showing that you can map the knob range to different output ranges by using `SCALE` :

```
#M

CV 1 PARAM
CV 2 SCALE 0 V 10 V 10 0 PARAM
CV 3 SCALE 0 V 10 0 V 5 PARAM
CV 4 SCALE 0 V 10 V 5 0 PARAM
```

it's a super simple scene but what we have is essentially a 4 CV preset morpher (which makes it sound more exciting). you have one set of voltages when the knob is down all the way, another when it's all the way up, and turning the knob morphs between them. what would make it really usable though if instead of typing voltage values in we could actually use the knob itself to choose them.

let's add the ability to record snapshots. it's simpler than it sounds. first, we need some way to indicate whether the knob is being used for recording or for morphing. let's use variable `A` - value of 0 can mean we're morphing, value 1 - recording CV 1 etc. first, we need to add this line to the metro script:

```
#M

IF A: SCRIPT 8; BREAK
CV 1 PARAM
CV 2 SCALE 0 V 10 V 10 0 PARAM
CV 3 SCALE 0 V 10 0 V 5 PARAM
CV 4 SCALE 0 V 10 V 5 0 PARAM
```

since any positive value is evaluated as "true", then when we are recording (`A` is 1…4) it will execute script 8 (which is what we'll use for recording) and stop.

one more thing: we need to store 8 values - 2 snapshots of 4 CVs. let's store snapshot 1 in pattern bank 1 and snapshot 2 in pattern bank 2. we also need a variable that determines which snapshot we are recording, let's use `B` (we'll worry about where to set it later). here is script 8:

```
#8
PN B A PARAM
CV A PARAM
```

the first line stores the knob value, the second line passes it to the corresponding CV output so we can hear the result.

now, we need a way to trigger recording. let's use scripts 1-4 to turn recording on, one for each CV, then we can use keyboard to toggle recording. since we need to record 2 snapshots, we could do it this way: executing script 1 enables recording for snapshot 1, executing it again enables recording for snapshot 2, executing it again stops recording and enables morphing again:

```
#1
IF AND A EQ B 1: B 2; BREAK
IF AND A EQ B 2: A 0; BREAK
A 1
B 1
```

first line will only get executed when `A` is not zero (so, we are recording already) and `B` is `1` (we are recording snapshot 1), so we switch to snapshot 2. second line will only get executed when we are recording snapshot 2, so set `A` to 0 to stop recording. and if we get past that, we're not recording, so we enable it.

we can now copy this script to scripts 2-4 (remember, you can select several lines by holding `SHIFT` and copy them at once). then modify `A 1` to `A 2` in script 2 etc. here is a neat trick we could use so we wouldn't even need to edit scripts after duplicating them: `A SCRIPT`. `SCRIPT` op returns the current script number, so it'll do exactly what we need!

switch to pattern view before you record snapshots - this way you can see the values being updated as you turn the knob. it would be also neat to see which value is being recorded at any given time - let's use for it! we'll need to make it visible when we start recording and set its coordinates to the proper pattern cell, and then hide it once recording is done. unfortunately, adding it to the existing script pushes us over the line limit, so we have to modify the script a bit and use another trick - if you have a really long `IF` condition, store it in a variable and use the variable instead:

```
J AND A EQ B 1
IF J: B 2; @X B; BREAK
J AND A EQ B 2
IF J: A 0; @SHOW 0; BREAK
A SCRIPT; B 1
@X B; @Y A; @SHOW 1
```

we can also employ the `A SCRIPT` trick, so you can just copy this whole script to scripts 2-4 without modifying anything (and we could save ourselves time and make it more readable by moving the whole thing to script 5, changing `A SCRIPT` to `A Z` and then scripts 1-4 could simply be `Z SCRIPT; SCRIPT 5`).

finally, we need to update the metro script to use the new snapshots instead of the fixed values. for each of the 4 CVs we need to scale the current param value from 0…10 V range to the min/max range saved in the pattern banks. this would be something like this:

```
CV 1 SCALE 0 V 10 PN 1 1 PN 2 1 PARAM
```

this is definitely too long! let's use aliases, and also take advantage of using a loop. modify the metro script as follows and add script 7:

```
#M
IF A: SCRIPT 8; BREAK
L 1 4: SCRIPT 7

#7
```

```
J SCL 0 V 10 PN 1 I PN 2 I PRM
CV I J
```

if you tried it at this point and are wondering why the values are in the middle and start from row 2 - that’s because pattern banks and indexes are 0-based. doesn’t really matter here and simple enough to subtract 1 when needed.

final thing: remember to set the metro rate in the init script, and we need to start with proper values for `A` and `B`:

```
#I
A 0
B 1
M 25
```

oh, and would be useful to be able to reset all values at once. easy! let’s use script 5:

```
#4
L 1 4: PN 1 I 0; PN 2 I 0
```

@scanner_darkly_

continuing with my #teletype concept series: a preset morpher - record 2...

so here it is, simple enough but pretty useful. and as always, with teletype this can be just a starting point for further exploration. here are some ideas: with something like telexo or er-301 we could control more than 4 values (just have to be careful not to have too many - remember, we are still updating each one of them at 25ms or faster). instead of the knob you could use faderbank to record 16 values at once. and what about storing and morphing between 4 snapshots instead of 2?

and here is what a more complex variation of this idea might look like: morphing faders grid scene

@scanner_darkly_

even more fun to control 16 parameters of single voice and put morph position...

18 Likes

6 August 14, 2020, 4:48am

Thank you for these. I’ve found a whit whale of a module recently that was what I had got the TT to interface with. These use cases are incredible real world examples that I am SO EXCITED for. I will be revisiting the original studies, then taking these and the grid studies on as a sort of “applied extensive studies.”

Your work on this module has been so inspiring and I can’t wait to begin applying these concepts in my experimentation.

Thanks so much! (Also, I always forget… what is the ! used for again… 😅)

1 Like

---

7 August 14, 2020, 5:51am

> kasselvania:
>
> what is the ! used for

Negation, in teletype evaluates to true on non-zero, including negative values

EDIT: Exactly 100% wrong. Should have written (hope I don’t mess it up again):

Negation, in teletype this means 0 becomes 1, non-zero becomes 0

1 Like

---

8 August 14, 2020, 1:21pm

Okay, so I reread the line; basically, if a button pressed equals 0, then it breaks that line, meaning it doesn’t run the rest of the scene…
Is that correct?

```
#I
G.BTX 1 0 0 1 1 0 0 9 16 8

#M
IF ! G.BTNV: BREAK
A G.BTNX
SCRIPT + 1 G.BTNY
```

I’m reading Study 6 (which I would like to start going back through this again this weekend) I see the competitors and understand. Thanks!

---

9 August 14, 2020, 2:21pm

Incredibly interesting and helpful, thank you for posting these!

> kasselvania:
>
> I’ve found a whit whale of a module recently that was what I had got the TT to interface with.

Just curious, what’s the module and how are you using it with TT? 😄

1 Like

---

10 August 14, 2020, 2:28pm

I feel like this is one of the more basic answers, but the module is an ER-301 😃

They’re just so hard to come by I almost felt bad bragging about it. It’s why I had wanted a TT all along. Having the ability to trigger so many things with simple scripts is a big big plus for me. It allows me to do a lot with a selective few modules.

With the TT and ER-301 connection, I’m dying to try getting the arc involved, having Max patches output i2c signals instead of midi through crow.

1 Like

---

11 August 14, 2020, 4:02pm

> a773:
>
> Negation, in teletype evaluates to true on non-zero, including negative values

this doesn’t sound correct - it will evaluate to false on non zero values.

`! x` will return 1 (equivalent to “true”) if `x` is 0, and 0 (equivalent to “false”) for any other value. used with `IF`:

```
IF ! x: ...
```

it means: only execute this line if `x` is 0.
this line:

```
IF ! G.BTNV: BREAK
```

is typically used with momentary buttons since the script is called both on button press and release. so if only want to take action on button press, you check the last pressed button value (`G.BTNV`) - if it’s a release, the value will be 0, so `! G.BTNV` will evaluate to true and it will break.

> kasselvania:
>
> I've found a whit whale of a module

for a second i thought you mean the actual module 🙂

3 Likes

---

12 August 14, 2020, 4:28pm

> scanner_darkly:
>
> this doesn't sound correct - it will evaluate to false on non zero values.

How stupid can I be???

You're obviously 100% correct, my brain must have been… I don't know what.

Let's do some simple example lines, that will all fire TR 1 (hope I don't mess it up this time as well):

```
TR 1
IF 1: TR 1
IF ! 0: TR 1
```

1 Like

---

13 August 15, 2020, 12:56am

forgot to update the metro script at the end - updated my post!

---

14 September 7, 2020, 9:51pm

concept: patch/preset manager

imagine a micro system that is fully repatchable from teletype. change your whole patch by pressing a key. teletype as the heart of a system: this is the concept i want to explore in this post. using teletype as a preset manager might seem like underutilizing what it's capable of, but i think it's the exact opposite, and a good example of how teletype can "glue" a whole system together.

let’s look at preset management first.

what kind of presets can we store? the most obvious example is modules that have no presets but have CV inputs for the parameters you want to store. then it’s as simple as using teletype CV outputs to control those (or using trigger outputs to control gate inputs). with teletype alone you can control 4 CVs and 4 gates. if you need more than that, you could use ansible or telexo to gain another 4 CV and 4 trigger outputs.

the downside to this is that you have to use teletype to change these parameters (or use the knobs and then approximate the knob values with your script). you can address this by using the teletype knob and recording snapshots as shown in the . still not great, since you have to record each value separately, but this could be addressed by using a midi controller and the .

this might still seem fiddly but here is what you gain: you can control CVs more precisely, you can switch multiple CVs at once and you can have multiple banks of values. something like pressure points can be used in this manner and gives you 4 sets of 3 CVs - with teletype you can record and recall **64 sets of 4 CVs** (using pattern banks) - and that’s just one scene. and you can do other interesting things, like morphing between different snapshots (i will refer to the again).

---

some modules have their own preset management, and in many cases teletype can be used to control those as well. trilogy and ansible come to mind first - whitewhale, earthsea, meadowphysics, ansible can all store presets, and you can select them with ops:

whitewhale: `WW.PRESET x` and `WW.PATTERN x`
earthsea: `ES.PRESET x` and `ES.PATTERN x`
meadowphysics: `MP.PRESET x`

ansible `ANS.APP x` to select an app
kria app: `KR.PRE x` and `KR.PAT x`
mp app: `MP.PRE x`
es app: no ansible versions, use the earthsea ops above
cycles app: `CY.PRE x`
levels app: `LV.PRE x`

with ansible ops you can also get the current preset and pattern (where applicable) if you don’t supply a parameter.

---

other modules / firmwares that allow you to select presets via i2c:

orca: `OR.BANK x` and `OR.PRESET x`
disting ex: `EX.PRE x`
matrixarchate: `MA.PGM x`

not all i2c enabled modules support preset selection via i2c (you can’t do it on er-301, for instance). not all need it - for something like telexo and just friends it’s just a matter of using individual ops to get them into the desired state (using something like `JF.MODE` and `TO.ENV.ACT`).

you can also control various select bus devices - rene mk2, voltage block etc. you will need to use expert sleepers disting ex module for that, and then use ______________ _ and refer to the select bus specification. and if you add a midi breakout, you can also send MIDI program change messages, so you could even control presets on various MIDI devices!

---

now, what about patch management?

to control an actual patch we'll need to employ additional modules - VCAs and switches. you could plug an LFO into several VCAs and send it to multiple destinations, for instance, and use teletype CV outs to control which VCAs are open, thus controlling the patch. voltage controllable switches, similar idea: you could use teletype trigger outs with something like ___ ______ _____________ or CV outs with ___ ___ _____ __________________. there are many similar modules to choose from.

you could also implement a patch matrix. you have several options: use something like ______________ and 3 telexo modules (which will give you 12 CV outputs, and you have 4 on teletype, so 16 altogether) to control its nodes. not the most practical option perhaps! you could also use disting ex's matrix mixer algorithm which gives you a 6x4 matrix which you can control directly via i2c. finally, ______________ will give you a 16x8 matrix, also directly controllable via i2c - just be aware that it has several quirks that might make it not suitable for your purpose (see the thread for details). grid interface would work really well for controlling a patch matrix.

---

ok, this one is for completeness sake - you could also store patch notes in a scene description 🙂

---

as you can see, you can recall many different things via teletype, which makes it a really powerful preset manager. there are many ways you could go about it too.

you could simply have each scene set up various parameters and load specific presets on other modules, and then just load scenes to switch patches. you could store multiple snapshots (both CVs and preset numbers) in one scene using pattern banks.

you could do something more interesting - like the already mentioned morphing between different snapshots (this won't work for loading presets on other modules, of course). another idea: meta controller. build a grid interface to control presets on other modules all in one place. or take it further and build a meta sequencer that sequences presets (just take into account that it might take some time to load a preset, depending on the module).

or how about this: **imagine a grid scene that is set up like a dj mixer**. left side selects presets on, say, ansible and earthsea, right side selects presets on disting ex and voltage block. so 4 chains, each one with its own sequencer and sound source. teletype CVs control VCAs so you can control volume of each chain independently. you could program a crossfader too. then you could build a performance where you select a new preset while it's playing the other side, switch parts gradually, combine them in different ways, all played from a grid.

---

i would really love to see more discussion in this thread too - please share your own thoughts and ideas or if you got inspired and implemented your own variation of any of the concepts described here i would love to learn about it!

22 Likes

Split this topic 15 October 11, 2020, 5:46pm

2 posts were merged into an existing topic:

16 October 11, 2020, 6:17pm

I've had this drafted here since this topic started, wasn't sure if you wanted other people to share or if this was more of a diary/log type thing—been really enjoying your ideas thanks for all your work and for sharing!

teletype as nested sequencers

in my smallish case, teletype is really my only sequencer-like-object. I'm working on creating a basic scene which I can use as a starting point when creating songs or performances, rather than building from scratch every time. I don't really have anything sending gates or triggers in to Teletype, so I wanted to be able to have complex rhythmic control over internally triggering scripts, that was the starting place…

This is basically what I have so far—it's quite basic and I'm sure I will build on it and it may become slightly more complex. The benefit of it is that I can pull up a rhythm or sequence quickly. The more complex the basic building block is the less open ended the building becomes, so it probably wont get too much more complex… I am not a super proficient Teletyper, but I hope this can show that even luddites like myself can learn a couple key tricks and get the tool they need. Big shoutout to .

1. I wanted a tempo knob and I found myself not using param much so I put it to use 🙂

```
#I
PARAM.SCALE 60 240

#M
M BPM PARAM
L 0 7: $ 8
P.N 0; P.NEXT
```

2. Working backwards sort of, but if you understand bitwise ops you can see here I'm using script 8 to trigger scripts 1-7 with binary reading of numbers in pattern 0.

```
#8
X BGET P P.I I
IF X: $ + 1 I
```

3. For simplicity I will often have scripts 1-4 controlling triggers 1-4, usually with one trigger controlling a bass drum, one a high hat, and two going to two inputs on a sampler. Heres what it might look like (Sometimes I have TR.P 4 controlled by metro as like a base clock to send to just friends or the samplers grain sync. Then script 4-7 control CV sequencing.)

```
#1
TR.P 1

#2
TR.P 2
DEL CHAOS: TR.P 2 (just for fun random hihats)

#3
TR.P 3

#4
TR.P 4

#P0
9 (bass drum + sample 2)
0
2 (hihat)
0
1 (bd)
4 (sample 1)
2 (hh)
4 (sample 1)
```

4. Lastly I'll use the CV outs or i2c to control melody. Using P0 I'll trigger scripts 5-7 which will advance patterns 1-3. I feel like I may be getting too in the weeds now because even the stuff I wrote just above ^^ is beyond my "base scene". Once muting gets working properly (Alt-F1, for example, so that it muted from internal triggering as well as external—hopefully on the next teletype update, fingers crossed) then I may leave some of this prewritten, we'll see. I also have toyed with using ________ to manually trigger scenes 4-7 to get the melodies more off the grid, but I have had no success with this—it seems like it will acknowledge triggers from elsewhere over the bridged connection, but I can't turn a high gate into triggers by tapping, perhaps even my fastest taps are too long of a gate (although with Just Friends sending triggers it seems like my teletype normally acknowledges two triggers—gate high and gate low—so I'm still a bit puzzled).

5. Further possibilities: I haven't had JF hooked up to i2c recently, but the possibility of using binary to control Just Friends in Geode mode or even just to control the triggering of envelopes instead of just triggers has certainly crossed my mind and I may be exploring this in the near future. Multiple bitwise/binary patterns maybe? With something like that, trigger outputs would be freed up for other uses… Basically any deeper i2c control I have yet to really incorporate into this scheme, I have some w/s which may get involved here too…

I hope this seems helpful and on topic to someone 😛 As you can tell I'm still working on this, and it's developing alongside my experiments with a Ciani inspired permanent patch which I've written about already _____ __ of _____. Part of the appeal of a base patch is also that I will always know — CV 1 controls the bassdrum voice on Mangrove 1, CV 2 and 3 are controlling Mangrove 2. X triggers are doing Y, param is tempo. In building an instrument (which is my goal) I want to easily decipher what the brain is doing at any given time in order to be able to perform using only the teletype keyboard and some knob turning.

PS since writing this draft a couple months ago has made me want to do a version of this base script using binary ops to patch up a little acid machine—a four voice drum machine with some mod channels where you can slowly change individual steps over time… it seems like a perfect job for teletype, and I'm sure there's more complicated and sophisticated ways to do it, but keeping it simple appeals to me… 🙂

20 Likes

17 October 11, 2020, 7:11pm

this is great, and please feel free to post here - it's an open topic / discussion!

your scene in a way is the essence of teletype - it's a device that orchestrates events based on triggers, but unlike conventional sequencers, instead of just generating a CV/gate you can have more complex logic being executed on each step.

an alternative way to set up this kind of a scene is to just trigger all scripts on each metro but then use something like `PROB` or `EVERY` to control individual script execution. the benefit of doing it your way is being able to control all 8 scripts from a single pattern value (a fun thing to do would be generating random pattern values).

why this kind of a scene is great is because you can do many things in each script. you could add ratcheting by doing something like `DEL.R 4 / M 4: TR.P 1` . or add a simple `CV 4 RND V 10` and then use it to modulate a filter or a delay, so you have a sample&hold that will change CV 4 value each time a drum is triggered.

re: allflesh not triggering teletype - you can use gates to trigger scripts, likely it's just not hitting the threshold.

5 Likes

18 October 11, 2020, 8:08pm

Thank you, fantastic!
Two main take aways for me:

1. Should use the param knob more!
2. Should learn Bitwise OPs!

The way you store the sequence with multiple scripts triggered per step is just genius!!

5 Likes

19 October 11, 2020, 8:10pm

Same here! Gonna dive into Bitwise ops! Thanks for your post!

1 Like

20 October 18, 2020, 10:41pm

omg insane system really diggin it ! 😍

21 October 18, 2020, 11:35pm

This was really inspiring and provided several friendly jumping off points for me, thanks.

2 Likes

24 January 20, 2021, 12:25am

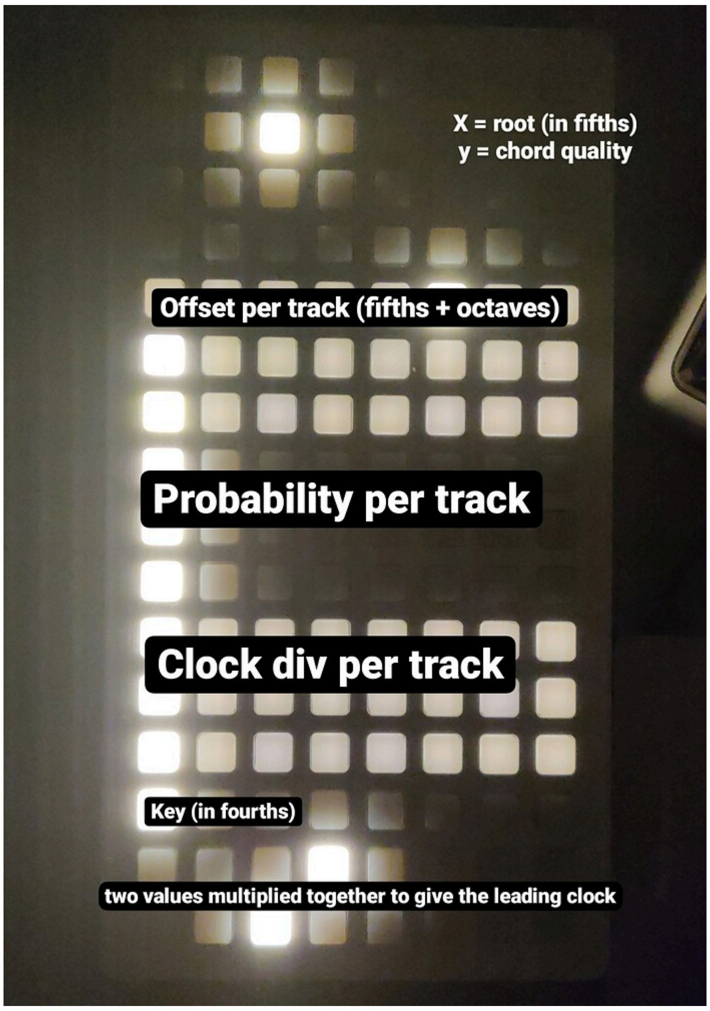
Concept: Kria Companion

I've been working extensively with Kria over the past year or so, and have nailed down a pretty good flow with a certain set of external controls, typically through 16n. However, I've recently been playing more "hands on knobs" because of certain normalizations within modules (DPO wavefolder), or else because many parameters I'd want to tweak by hand are already receiving modulation from elsewhere and I don't have 16 free VCA/CV mixing channels lying around. This led me to neglect the 16n, since I'd be constantly riding knobs rather than faders and, in my last bigger patch, I only used three of the sixteen faders (and not particularly elegantly). So, I thought, let's get rid of the 16n altogether and *maybe even teletype while we're at it*.

This is, essentially, a "performance page" for kria tailored to my specific goals that makes heavy use of grid ops for teletype, and that I hope to transfer to satellite soon.

I'd like to say this only took three evenings, coming from lots of tt experience but 0 other coding besides, and the bulk of it was actually done this afternoon after reading about button groups and realizing all of the problems I was facing had already been solved.

I did a little sketch of the button lay out for the parameters I wanted and then set about creating SO MANY groups.



```
I
G.GBX 0 1 0 0 1 1 1 0 5 4 8
G.GBX 1 33 4 0 1 1 1 3 6 1 8
G.GBX 2 41 5 0 1 1 1 3 6 1 8
G.GBX 3 49 6 0 1 1 1 3 6 1 8
G.GBX 4 57 7 0 1 1 1 0 6 1 8
$ 8; $ 7

8
G.GBX 5 65 8 0 1 1 1 0 6 1 8
G.GBX 6 73 9 0 1 1 1 0 6 1 8
G.GBX 7 81 10 0 1 1 1 3 6 1 8
G.GBX 8 89 11 0 1 1 1 3 6 1 8
G.GBX 9 97 12 0 1 1 1 3 6 1 8
TR.TIME 4 11

7
J 105; K 113; D 121
G.GBX 10 J 13 0 1 1 1 0 6 1 8
```

```
G.GBX 11 K 14 0 1 1 1 0 6 1 8
G.GBX 12 D 15 0 1 1 1 0 6 1 8
```

So that gives us 13 groups of vertically arranged buttons, with a sorta striped pattern to help guide the eye, and with the leftmost group containing four columns of buttons rather than just one column per group. Those `GROUP 0` buttons also trigger a different script from the rest of the buttons.

The first four columns I'd like to control harmonic content, with kria scale on the y axis and offset in fifths along the x axis (so that, left to right, you get I, V, II, VI, with highly limited four-note scales interior to Kria, based in part on the rings/plaits chord mode, mapped vertically, most consonant on top, most dissonant on bottom). These buttons all trigger script 5:

```
5
KR.SCALE + 8 G.BTNY
G.BTN.SW G.BTNI
A + * 5 G.LED 13 8 * 7 G.BTNX
```

This first component of A (`* 5 G.LED 13 8`) will be explained later. Otherwise this script just literally "change kria scale to Y (+ 8 cuz that's where custom kria scales sit for me)" and "create variable A which is equal to X as expressed in fifths (+ some other fuckery)" This A is a component of all of my three "note-triggering" scripts.

The next three are per-channel offsets that scale up in fifths and octaves. This is where the key trick of the script comes out: storing values in grid row 8 via brightness (hence, out of reach). In fact, that's *all script 6 does at all*. Ninety of our buttons just say "turn off the other buttons in this row and set the brightness of a non-existent square to this integer value so we can MATH it later."

```
6
G.BTN.SW G.BTNI
G.LED G.BTNX 8 - 7 G.BTNY
```

The next three are per-channel probability faders for controlling note density.

The next three are per-channel clock dividers.

The next one is a "key" designator, which acts as a master offset in fourths (this is that first component of our variable A).

Finally we have a pair of values which are multed together to give the clock base. I like having two. different controls as one can get clean divisions at a glance or deliberately increase / decrease in a fashion that does not contain an obvious division of the prior state (by changing both values simultaneously).

Each of these groups just triggers script 6, but here's how they get used, ultimately:

```
1
J / QT KR.CV 1 N 1 N 1
K / QT KR.CV 1 V 1 V 1
CV.OFF 1 PN 0 G.LED 4 8
```

```
PROB - 100 * 14 G.LED 7 8: BRK
CV 1 + V K N WRP + A J 0 11
TR.P 1
```

Set local variables J + K to nice clean semitone and octave values, respectively. Set the lowest possible note to some number from this list:

```
P0
0
7
12
19
24
31
36
43
```

corresponding to the position of the fader on column four. If column seven is set very low, probably stop here. But maybe don't. If not, trigger a note that folds together the proper key and chord offset (on top of the octave / fifth "master offset") and wraps, maintaining the integrity of any octaving coming from kria.

The next two scripts are duplicates of this, incremented by one in key locations:

```
2
J / QT KR.CV 2 N 1 N 1
K / QT KR.CV 2 V 1 V 1
CV.OFF 2 N PN 0 G.LED 5 8
PROB - 100 * 14 G.LED 8 8: BRK
CV 2 + V K N WRP + A J 0 11

3
J / QT KR.CV 3 N 1 N 1
K / QT KR.CV 3 V 1 V 1
CV.OFF 3 N PN 0 G.LED 6 8
PROB - 100 * 14 G.LED 9 8: BRK
CV 3 + V K N WRP + A J 0 11
```

And then in the metro script…

```
EVERY + 1 G.LED 10 8: KR.CLK 1
EVERY + 1 G.LED 11 8: KR.CLK 2
EVERY + 1 G.LED 12 8: KR.CLK 3
J * G.LED 14 8 G.LED 15 8
M * J 30; TR.P 4
```

I'm posting all this in part to force myself to audit my own work (found two mistakes!!!), and in part to encourage people to take advantage of 1) this awesome thread 2) FUCKING AWESOME GRIDOPS!!! and 3) satellite.

The last part is to ask a little aaaaaaaaaadvice. Are there any overtly dumb things going on here? Something super clunky for no reason? I want to turn this into a satellite script and migrate TT out of my main case (because this is all I've been using TT for for a looooooong time), but want to make sure this is smooth sailing first. I know Satellite only has two trigger ins – could I just set it to poll Kria's trigger outputs in the metro like so: "IF KR.TR 1: $1", or will this upset the I2C gods? Finally: I'd like to create a per-note keymap for the leftmost quadrant of the grid to highlight "safe" harmonic choices and aid in learning the moves from a set of ~12 possible next moves each rather than 32 – is there a good way to do this in 1 script, possibly storing data in the tracker / unused grid buttons / both? That leap still isn't happening for me…

Also let me know if there are any questions about this. The thought of a custom Kria page is… Amazing. Then thinking "custom page for ANYTHING" is just… Too much.

14 Likes

---

25 January 20, 2021, 8:11am

This looks really fantastic and gives me another push to revisit grid ops (which so far I've found a little on the wrong side of the planned coding / live coding divide for my tastes). I'm very keen on the idea of using Teletype as a metasequencer for kria (and also, perhaps, the other way around)! Can I ask, do you have two grids? Or do you have a reliable way of switching the grid between the two modules. I've tried various cables and switches but it's always ended up being a bit clunky and/or noisy.

2 Likes

---

26 January 20, 2021, 11:37am

> SimonKirby:
>
> gives me another push to revisit grid ops

!!!

has worked for me but ymmv. I've found tt and grid noise to be magically disappearing / reappearing type things, varying due to which wall it's plugged into, how many LEDs are lit, etc.

I will say that this particular page I use *instead of* grid control of Kria, not *in addition to*. My sequences are "prepared" in an extra-especial fashion to work well with this particular set of manipulations.

2 Likes

---

27 January 22, 2021, 4:20am

sorry, finally found the time to reply - this is totally amazing!! the scene itself and the detailed breakdown both.

this is such a great example of how you can essentially add your own features to kria, earthsea and any other i2c controllable apps - all the way to creating your own interface for them! i love the fact that you didn't just expand kria's interface but replaced it with your own. and yeah, a good way to use satellite as this is something you likely to build once and then maybe only tweak occasionally (also a reminder for me to fix the bug in satellite).

---

in terms of the script itself not much to add, really! it's very straightforward and shows that you don't really need a complicated script. glad you took advantage of groups, it would be difficult to achieve same functionality in a limited script space.

> yams:
>
> I want to turn this into a satellite script and migrate TT out of my main case

you'll need something to provide power/pullups for i2c - do you have anything on your i2c bus that would have that? if you're just planning to use ansible/kria + ansible/satellite you'll need an i2c backpack or something similar (txb would also work).

> yams:
>
> could I just set it to poll Kria's trigger outputs in the metro like so: "IF KR.TR 1: $1", or will this upset the I2C gods?

should be fine!

> yams:
>
> Finally: I'd like to create a per-note keymap for the leftmost quadrant of the grid to highlight "safe" harmonic choices and aid in learning the moves from a set of ~12 possible next moves each rather than 32 – is there a good way to do this in 1 script, possibly storing data in the tracker / unused grid buttons / both?

yeah i would say use patterns to store coordinates or something like that and then set them in a loop. and then use `G.LED` with brightness level set to `-2` - that's a special value that will "brighten" whatever is underneath, including buttons.

clever trick with storing values in unused LEDs! you're limited to range 0-15 though. i'd use fine faders instead with `G.FDR.V` ops. you can store 64 values this way.

have you seen the awesome `ANS.G` ops @csboling added a while ago? you're not limited to app specific i2c ops - you could emulate *any* grid press or a combination of presses.

1 Like

28 January 22, 2021, 5:01am

“This is great; recode everything” – the best kind of feedback!

Really though: switching to faders is gonna help save a ton of space (in subtle ways)! Lots of other gold here, too 🙂

I’ve never done anything with loops but will hopefully start tinkering tonight after the rebuild. Expanding harmonic vocabulary was the real catalyst to move to this (vs a single button that changed keys in fourths) and it’s a little difficult just hunting and pecking for transitions lol.

I was thinking to put Crow + JF on the I2C bus. Crow should gimme the pullups needed, right?

1 Like

---

29 January 22, 2021, 1:48pm

> yams:
>
> Crow should gimme the pullups needed, right?

Yes, as long as you haven’t disabled the pull-ups in your user script (for whatever reason).
JF running 4.0 firmware also provides some weak pull-ups.

2 Likes

---

30 June 13, 2021, 9:55pm

concept: simple performance patch

this is a re-recorded version of the patch i did for flash crash (the original performance was shaky due to my nerves, the recording seem to have some timing issues and the screens aren’t very readable). i initially considered doing something unusual, like creating some special ops or hacking the firmware in some other manner, but what i really wanted to demonstrate is that you don’t need to use a lot of ops or complicated techniques to get something going.

there is also nothing particularly special about 2 teletypes - could probably do the same with one but it made it easier to separate voices. the left teletype controls the left disting ex via i2c, which is used as a drum sample player (sd triggers algorithm). it’s further processed by mimeophone. the right teletype controls telexo / just friends / another disting ex (running augustus loop delay).

the crossfader is mainly used to mix drums and just friends output. i initially planned to do something interesting by controlling teletypes with it by patching the crossfader CV to teletype inputs and using that voltage to control various script aspects, similar idea to octatrack, but ended up not using it that much.

here is the breakdown of what's happening:

00:09
i start by initializing the synth mode on JF. i also wanted to show you can get so much variation by simply playing the same 6 notes (you'll see below i partially failed to do that even in this re-recorded patch).

```
JF.MODE
L 1 6: JF.NOTE N P I V 5
```

i populate patch bank 0 with 6 notes and send them to JF with the above loop. from now on the only pitch change that will happen will be through transposition.

00:40
create a counter using variable `T` to simply step through the 6 notes:

```
T % + 1 T 6; J + T 1
JF.NOTE N P T V 5
```

first mistake here, `J` should be used instead of `T` on the 2nd line. `T % + 1 T 6` means: "add 1 to `T`, then wrap it back to 0…5 range using the modulo operation". since i'm using pattern values 1-6, i needed to shift this range to 1…6, which is what `J` was for.

01:26
add some JF modulation using telexo LFO:

```
TO.OSC.CYC 2 15000
TO.CV 2 V 5
```

this will start an LFO on output 2 with cycle of 15sec and -5…+5V voltage range.

01:48
create a super simple trigger sequencer with grid ops:

```
G.BTX 0 0 0 1 1 1 0 0 16 8
```

this creates a block of 1x1 grid buttons in a 16x8 block - 8 tracks with 16 steps each. adding this to the metro to highlight the active step:

```
T % + 1 T 16
G.CLR; G.REC T 0 1 6 -2 -2
```

tracks 1-6 will be used to trigger drums 1-6 on the left disting:

`L 1 6: SCRIPT 7`

and this is script 7:

```
J + T * - I 1 16
IF G.BTN.V J: EX.TV I PARAM
```

`G.BTN.V` will tell us if a grid button is pressed or not. since the buttons started with id 0 (1st parameter to `G.BTX`) and they are given ids that increase left to right then top to bottom, we can calculate the button number for the current track (which is stored in variable `I` when script 7 is called from the metro) and the current step (stored in variable `T`) by using the formula above.

if the button is pressed, `G.BTN.V J` will return 1, so the `IF` statement executes. `EX.TV` is a disting op that triggers the drum voice passed as the 1st parameter. the 2nd parameter is loudness, and by using `PARAM` i can use the teletype knob to control drum volume.

03:53
i also want to use tracks 6-8 to send triggers to teletype #2, so i copy script 7 to script 6 and modify it to send triggers: `IF G.BTN.V J: TR.P - I 5` (we have to subtract 5 so that tracks 6-8 translate into trigger outputs 1-3).

i patch teletype #1 trigger output 1 to trigger input 1 on teletype #2. i stop the metronome on the 2nd teletype, and instead call the metro script from script 1. i also clock the augustus loop algorithm on the right disting with `EX.AL.CLK`.

i also connect trigger output 2 to trigger input 2. now script 2 on teletype #2 is also sequenced from the trigger sequencer. i will use it later to transpose the sequence.

06:05
instead of stepping through individual notes i change it to a chord stab:

`L 1 6: JF.VTR I V 5`

`JF.VTR` has 2 useful aspects - it triggers JF voices using the last pitch received, but you can still change the volume of each voice.

06:33
syncing mimeophone to teletype #1 metronome by using trigger output 4

07:05
adding further variation by transposing the whole JF chord:

```
C % + 1 C 4
JF.SHIFT N PN 1 C
```

JF makes this kind of thing very easy!

08:04
i spend a couple of minutes here trying to set up a telexo voice as the bass. first, enable envelope and choose a waveform:

```
TO.ENV.ACT 1 1
TO.OSC.WAVE 1 2400
```

then trigger the voice and set the volume using the param knob:

```
TO.ENV.TRIG 1
TO.CV / PARAM 10
```

there is no sound because the current pitch is 0 - which is very low.

10:24
the bass finally drops! always remember to shift by several octaves up for telexo. i use the same notes i used for transposing the chord for the bass.

11:06
drums are pretty static - let’s add some variation on each 8th step. in metro:

`EVERY 8: SCRIPT 4`

which is simply a way to call script 5 in a loop (i only want to modify drums 2-5):

`L 2 5: SCRIPT 5`

and script 5:

```
J + 1 * I 3
EX.P + 7 I RRND J + J 5
```

this needs some unpacking. i prepared drum samples specifically for this patch by collecting 6 samples for each of 6 voices and naming them so that kick would be files 0-5, snare - files 6-11 etc etc. `J` will give me the number of the 1st sample in the group, and `RRND J + J 5` will select a random sample from within that group (except that i made another mistake here and `I` should be multiplied by 6). i then use `EX.P` op to change assigned sample to the randomly selected one (side note: changing folders is not instantaneous, so i wouldn’t try to change it live, but changing individual samples seems to be fine!).

12:15
add random modulation for mimeophone - i typically use `CV RND V 10` as a super quick way to generate stepped random voltage. `CV.SLEW` could also be very useful here.

12:47
further JF variation - instead of playing a chord, i change it to play a random note. there is a mistake in the line - instead of `JF.VTR RRND 1 6 V 2 V 7` it should be `JF.VTR RRND 1 6 RRND V 2 V 7`, so that in addition to playing a random note it should also use a random volume for it, but since i didn't include `RRND` it just takes `V 2` as the volume parameter, resulting in me trying to figure out why JF is so quiet all of a sudden.

13:44
add another random stepped modulation for JF

14:31
one more random stepped modulation - but for this one i use script 3, which is triggered from the trigger sequencer track #8 - this is a good way to sync modulation changes.

15:48
mute bass drum by changing the loop that triggers drums.

16:55
change back from individual notes to a chord stab. i'm still missing `RRND` in front of `V 2 V 7` here, which is a bummer because one of the cool things to do with JF is to play the same chord stab but vary volume of individual notes - you get really nice organic changes.

18:12
i finally do something with the crossfader - simply modulating the position with a random voltage generated by teletype, which adds more variation to the drum track.

---

that's it! the only remaining thing to mention is that just friends is an incredible sound source - which is bad for practicing, because you get lost in all the sweet spots!

24 Likes

---

31 June 13, 2021, 10:31pm

Amazing, thank you for taking the time to record and share all this! It's really incredible that you're getting this out of such a small system — all those modules are really pulling their weight! Love the look of the video too 🙂

The way you were thinking about using the crossfader octatrack style is really interesting — using the param-knob or cv in to teletype (in the absence of a cool crossfader module 😛) as a way of sort of “preset shifting” or something, the way people use the octatrack crossfader, seems like something with a lot of performance potential that is under-explored (maybe just by me! lol).

2 Likes

---

32 June 13, 2021, 11:19pm

yeah, i was trying to think of a simple mechanism to do that… you are essentially using a value (crossfader position or CV or param knob) to scale between 2 sets of values. one thought was to use pattern banks, one bank for left side values and another bank for right side values, and have a script that would do something like `PN 2 I SCALE PN 0 I PN 1 0 V 10 PARAM`.

another option is to scale any constants you have in the code, assuming zero is appropriate as the minimum value - let’s say you have `DEL.X 15: ...` and you want to scale that 15 based on the param knob. you could do something like `DEL.X / PARAM / V 10 15` but it’s really awkward. maybe we need a dedicated op for this, so you could do something like `DEL.X SC 15 PARAM:`

2 Likes

---

33 June 13, 2021, 11:43pm

ahhhh i’m such a sucker for modulating delay arps like this. thank you for sharing and all the explanations. i’m watching and learning now!

2 Likes

---

34 August 11, 2021, 10:08pm

Here’s a wee demo of what im tinkering away with at the moment.
A lot like from @midouest , This Teletype scene uses grid ops to play Just Friends in poly mode with Grid, with dual pyramid interface and massive boiling sun.

Big thanks to all on the discord that have been helping me to try and add features and share their insight.

Current features
standard 12tet keyboard output or Quantised output
Hidden Sun Keys 1 & 2 activate two additional generative melodies, with `param` mapped to their amplitude

Once i’ve refined things more I’ll get all the code up for others to play around with it. Any features youd like to see included? basically 12 hidden buttons for mapping to things like `JF.RUN`, Triggers, CV outputs, metro etc.

Anyhoo, hope you enjoy the vid

15 Likes

35 October 13, 2022, 12:44am

This looks great - did you ever upload the code? I would be interested in having a play

36 October 13, 2022, 9:24am

ahh Sorry Jason, I’m afraid this is abandonware that got lost between firmware updates!
As a consolation, If you’ve not seen them already, @modularbeat made ***Minim*** and @Erekutoronikku made ***Gentle Antagonism*** which both look like excellent TT Gridstruments.

5 Likes

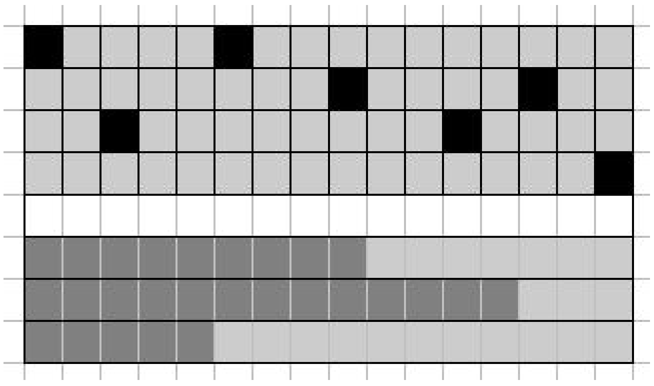
37 October 13, 2022, 9:45am

no worries, thanks for getting back - will check out those suggestions 🙂

3 Likes

38 December 24, 2022, 12:43am

i wanted to do something to try the which allows you to have 4 mutable braids under i2c control. i’ve been using my old braids as a percussion voice a lot, so a seemed like a good idea. but one of the coolest features of braids is the many models available, so why not make an elektron style plock-able trigger sequencer where you can select a different model for each step? and in addition to the model selection we could control other parameters too, like timbre and colour. something like this:



the top 4 rows represent the trigger sequencer steps - we just need 4 since there are 4 braids oscillators. the bottom 3 rows are taken by the faders we will use to edit step parameters. let's put this in the init script and execute it by pressing F10:

```
G.BTX 0 0 0 1 1 0 0 1 16 4
G.FDX 0 0 5 16 1 0 3 4 1 3
```

you can check that it looks correct using the grid visualizer - go to live screen and press `alt-g` . let's unpack:

```
G.BTX 0 0 0 1 1 0 0 1 16 4
      i x y w h t l s c  r
```

`G.BTX` creates a group of buttons:

- `i` is the index for the first button, we can start with 0
- `x` / `y` are the coordinates for the first button, we start with the top left corner
- `w` / `h` are the width/height for each button, so 1x1
- `t` is the button type, 0 is momentary
- `l` is the brightness level for not pressed buttons
- `s` is the script assigned for button press/release
- `c` is the number of columns
- `r` is the number of rows - so 4 rows of 16 columns

```
G.FDX 0 0 5 16 1 0 3 4 1 3
      i x y w  h t l s c r
```

`G.FDX` creates a group of faders - most parameters are the same as above, so we are saying: create 3 rows of faders (one in each column) starting with index 0, starting in 5th row and that are 16 LEDs wide.

typically, we would use latching buttons for the trigger sequencer portion, so why momentary buttons here? that's because i want faders to only be shown while a trigger button is pressed - this means we need to react to both button presses and releases, and only momentary buttons trigger a script on both. this complicates things a little as we can't use the button states to tell us which steps are on/off, so we will need to store that in a pattern bank. conveniently, there are 4 banks, so we can dedicate a bank to each one of the 4 oscillators.

we assigned script 1 to buttons, let’s add this line to it:

```
IF G.BTNV: SCRIPT 2; BREAK
```

now script 2 will get triggered when a button is pressed, and we can use the rest of script 1 to treat a button release. whenever a button is pressed, we can flip the state for the corresponding step. let’s add this to script 2:

```
A % G.BTNI 16; B / G.BTNI 16
PN B A ! PN B A
```

`G.BTNI` is the index of the button that triggered the script. since we told it to start with index 0, and indexes are incremented left-right, row by row, dividing the index by 16 gives us the row number and the remainder of that division (mod) gives us the step number (0..15). we can store them in variables `A` and `B` in case we need them later (spoiler: we will). then we simply flip the value of the pattern bank corresponding to the current row and current step.

let’s do the cool bit with the faders only shown when a button is pressed. we can use the “enable” op, in script 2:

```
L 0 2: G.FDR.EN I 1
```

this will enable (show) all 3 faders. now we should hide them in script 1 (when a button is released), add this:

```
L 0 2: G.FDR.EN I 0
```

and we should add this to the init script so that when the scene is loaded, the faders are hidden until a button is pressed.

now, let’s implement the sequencer part. switch to metro script:

```
M SCL 0 V 10 500 50 PARAM
x % + 1 X 16
G.CLR
G.REC x 0 1 4 -2 -2
L 0 3: Y I; SCRIPT 8
```

- the 1st line will use the param knob to control the metro rate
- the 2nd line increments `X` on each step and wraps it so that it stays in 0..15 range - this is our current step
- 3rd and 4th lines show a vertical line indicating the current step
- 5th line calls script 8 for each of the rows

script 8 is where we will check if a note should be triggered:

```
IF EZ PN Y X: BREAK
EX.VOX + Y 1 V 2 V 5
```

if the pattern bank `Y` value for to step `X` is zero, we exit, otherwise we play a note on the corresponding oscillator. since we use momentary buttons we also need to show steps that are on, this will need another script as we need to loop through each step, so we add `L 0 15: SCRIPT 7` to the beginning of script 8, and set script 7 to this:

```
IF EZ PN Y I: BREAK
G.LED I Y 15
```

basically: go through every step for row `Y` and set the LED brightness to 15 if the step is on (remember, we clear all LEDs in metro with `G.CLR`). at this point we should have a working trigger sequencer that even shows faders when a button is pressed - except the faders don't do anything yet. let's add that.

faders will be used to set 3 parameters for each of the 16 steps, 48 values total, and coincidentally we have exactly 48 values remaining unused in each pattern bank. faders are assigned to script 4, so this is where we will set the pattern values when a fader is pressed:

```
J + A + 16 * G.FDRI 16
PN B J G.FDRN
```

if you recall, script 2 sets `A` to the currently pressed button's column and `B` to the button's row (which corresponds to the pattern bank, so `PN B`). the first line calculates the pattern index: we start with step 16 (since steps 0..15 are used for the step states), and we additionally increment by 16 for each fader:

- steps 0..15: step states
- steps 16..31: 1st parameter value controlled by fader #0
- steps 32..47: 2nd parameter value controlled by fader #1
- steps 48..63: 3rd parameter value controlled by fader #2

we also need to initialize faders with the current values when a button is pressed. let's replace `L 0 2: G.FDR.EN I 1` line in script 2 with `L 0 2: SCRIPT 3` - we will call script 3 to do the work for each fader (set the fader value and enable it):

```
J + A + 16 * I 16
G.FDR.N I PN B J
G.FDR.EN I 1
```

now the final piece: use these values to control the actual oscillator parameters. this will also require a script, so let's add `SCRIPT 6` to script 8 right before we play a note, and in script 6 we set the parameters:

```
K * 13 Y; I PN Y + X 16
IF I: EX.P + 7 K I
I PN Y + X 32
IF I: EX.P + 8 K * I 64
```

```
I PN Y + X 48
IF I: EX.P + 9 K * I 64
```

it looks more complicated that it really is: disting ex dedicates parameters 7-19 to oscillator 1, 20-32 to oscillator 2 etc. so `K` gives us the parameter offset based on which oscillator we want to control (which corresponds to the row stored in `Y`). `I` is then set to the pattern value, and if it's not zero, we set the required oscillator parameter. parameter 7 is the model, 8 is the timbre and 9 is the colour - the last 2 have range of 0..1024, and since we used coarse faders, their range is 0..15, so we multiply by 64 (for model it just uses the actual value so it can only choose the first 16 models - but you could use fine faders to control a bigger range). why compare a value to zero? this way we can treat the value of zero as a special value for "don't change the current values".

this is it! except once you play with it you realize: if you press a step to simply change parameters, it will disable the step once the button is released so you have to re-enable it. what if we changed it so that steps are only disabled if the faders weren't touched? let's use variable `Z` to track that. first, remove `PN B A ! PN B A` line from script 2 and change it to:

```
IF PN B A: Z 1
ELSE: PN B A 1; Z 0
```

script 2 gets triggered when a button is pressed, so this means: if the current step state is already "on", set `Z` to 1 to indicate we will need to set the state to "off" when the button is released - unless it's currently off, in which case we set the step to "on" and `Z` to zero (meaning - don't turn off this step when the button is released). and let's add `Z 0` to script 4 which gets triggered when a fader is pressed. finally, add this to the end of script 1 (which gets executed upon a button release):

```
IF Z: PN B A 0
```

which translates into: if a button was pressed, and the step was already on, and no faders were touched - turn the step off when the button is released.

final touch: it would be nice to have a script to erase the current state and reset the oscillator parameters. script 5 is still empty and can be used for that:

```
L 0 255: PN / I 64 % I 64 0
L 0 2: EX.P + I 7 0
L 0 2: EX.P + I 20 0
L 0 2: EX.P + I 33 0
L 0 2: EX.P + I 46 0
```

perhaps not the most practical script - but a good example of what can be done with grid ops, and it could be optimized to add more functionality.

complete script: (1.8 KB)

29 Likes