ASCTard010 – Tapped Out

Tapped out is a pattern sequencer for the ardcore with some unique features. You will need the ardcore expander to get the most out of this sketch as it allows you to have 10 independent patterns playing at once. To get started you should familiarize yourself with the control layout as it is a functionally dense sketch.

Tapped Out features:

- 10 independent outputs with changeable speed and patterns
- 256 default patterns to choose from and you can upload your own
- Pattern shuffle
- Cross platform standalone app to create custom pattern files
- Sync/reset input (with 2 flavours of sync)
- Knob Value 'Pickup'
- Easy to configure user settings for extra control
- Presets saved when powered down

Control Layout:

A[0]: Output Channel
Regardless of layer this selects the output channel the other controls are addressing
Range: 1 (D[0]) to 10 (bit7)

L[0]: Pattern Selection
This control allows you to load a pattern into the currently selected output channel.
Range: Max 256 patterns

L[1]: Shuffle
Shuffle frequency for current selected output channel.
Range: 0-31(of current pattern tempo)

L[2]: Preset
Select preset number for save/load
Range: 1-8

L[3]:Global Tempo
Regardless of which output is selected with A[0] this will set the master tempo of the device.

Tapped out features a knob pickup function that means as you move through the outputs and layers your settings wont change to the current knob positions. To make the knob active simply move it through the set value.
If for some reason you want that behaviour you can disable this feature*1

A[1]: Active Layer
Selects the currently active layer
The active layer changes the function of A[2] and A[3]
There are 4 layers which are referenced as L[0] - L[3]

L[0]: Pattern Speed
This sets the speed of the current pattern. It is possible to edit these values for custom operation*1

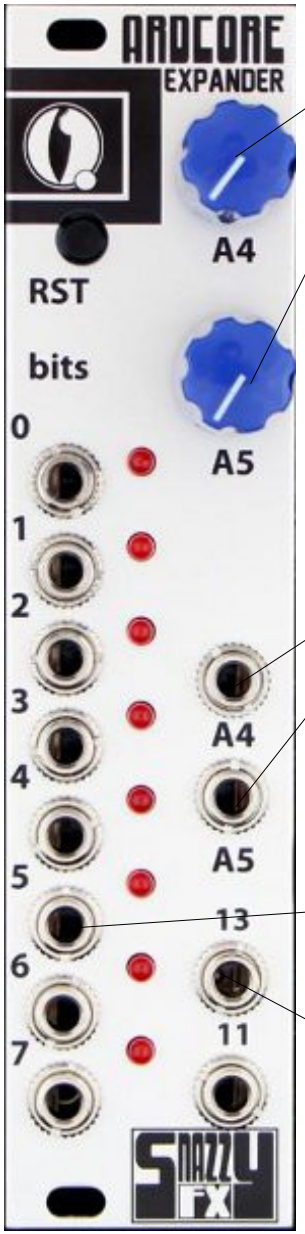
L[1]: Shuffle
Shuffle Time, this can be increased past the next note on time, and thus will suppress all notes until the shuffle conditions are met.

L[2]: Presets
Save or load a preset. To load a preset set the knob to the first 3rd of its range. To save a preset set the knob to its last 3rd. The middle section of the range is left deliberately free of function to make save/loads easier

L[3]:CLK Input Mode
Regardless of which output is selected with A[0] this will set the clock input mode.
If A[3] is < 12 o'clock the CLK input will reset the play position of all patterns
If A[3] is > 12 o'clock the CLK input will sync the tempo to an external clock. There are 2 sync modes*1

*1 see 'user_settings.h' for more information

Expander Layout:



A[4] and A[5]: User Assignable Inputs

These 2 knobs allow you to send modulation to a fixed location.

A[4] sets the output channel to apply the modulation to
A[5] is the modulation input

You can choose the modulation destination in the user_settings.h file.

<u>Destinations Include:</u>
Channel Pattern
Channel Speed
Shuffle Frequency
Shuffle Amount

NOTE! If your A[4] input is the same channel as the A[0] selection the modulation will NOT be applied to the channel, this is to allow you to easily change the initial conditions for the channel, but of course this can also be abused for creative effects

Bits[0-7]: Output

These bit outputs are patterns 3-10 and their settings can be adjusted by selecting them with A[0]

24PPQ out

Output 13 is a constant 24ppq out for DIN sync or syncing another clock

If "EEPROMreset" is NOT define in the user_settings.h file presets are saved in memory when the unit is powered down. Please note that EEPROMreset should be defined the first time you upload the sketch to initialize the required memory area

Patterns:

Patterns consist of a binary pattern of 32bits representing a drum pattern. The maximum number of patterns we can store is 256 (optimum max for knob value) and these are stored in the 'pattern.h' file.

Consider the pattern:
10001000100010001000100010001000

This obviously represents a steady pulse on every 4th tick (though the speed is independent of the pattern in reality). However if you look at 'pattern.h' you will see these patterns are stored as 32bit unsigned integer values, so the above pattern would be stored as '2290649224'

To make it easier to generate patterns there is an additional application that will generate 'pattern.h' files for you). It can generate these from csv (comma separated values) files. The csv of the default pattern along with a spreadsheet template for your own pattern are included with the software. The software is included for OSX, and Windows, though linux users are free to build a working version from the linux makefile in the application source.*[1]

It is very easy to make pattern files yourself though because all you have to do is convert a 32bit binary string into an unsigned integer. If you have python installed (osx users type 'python' in terminal) you can use the command ' int(0bPATTERN) ' where PATTERN is a 32 bit binary string as per our original example above, to output the appropriate value. You can of course use this the opposite way ' bin(INT) ' where INT is an unsigned integer, to check out the patterns that come preloaded with the sketch (if you prefer a more complicated method than opening the included spreadsheet of default patterns).

*[1] App source code is available from:

Software:

The ardcore sketch comes with a cross platform application to help you create pattern files.
This is a simple 2 step process

1) select a pad option, there are two options

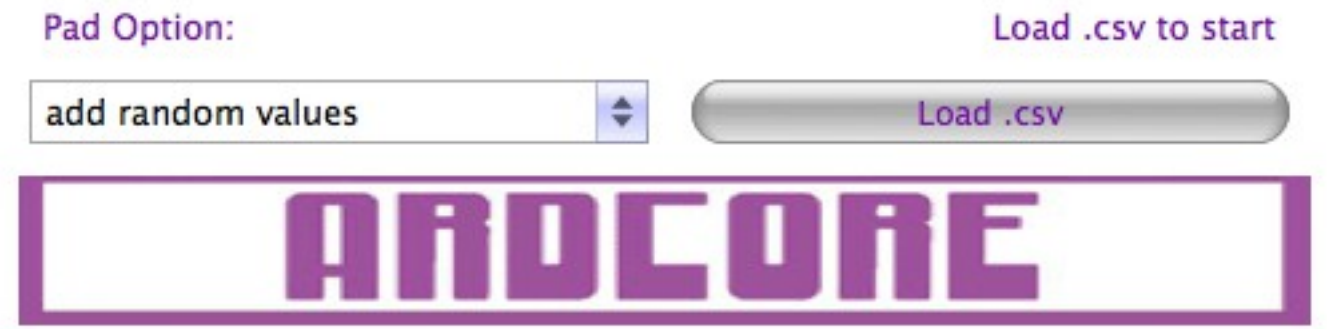
"add random values"
If your pattern file is less than 256 members it will fill the rest of the array with random patterns

"stretch values"
If your pattern file is less that 256 it will attempt to stretch the number of patterns you do have over the full 256 range. This is an easy way to make the pattern control knob less accurate

2) Load a csv and process
Select a csv file from your hard drive and choose the name of the output file (remember to include this file in your sketch when uploading instead of the default patterns.h)

A .csv and .ods (open document spreadsheet) of the default patterns are included with the sketch as well as a template to help you make your own.

Some things to note:
The first line of the csv is always ignored (so we can fill it with 0-31 to make filling out patterns easier)
Entries after line 257 will be ignored (256 patterns + header line = 257)



Tips:

- Play some really fast patterns and get the triggers from outputs[3-7] in a single stream from the DAC out, this creates some weird square wave patterns
- You can run tapped out at faster tempos with the L[3]A[3] CLK control set to sync mode and a very fast clock in, if the external clock is faster than around 30hz it will start to loose sync
- Run it at the fastest speed and send cv to the global tempo control for a very simple square wave OSC
- Try both sync modes. Normal sync is looser but will run if your clock is intermittent. Hard sync requires a clock to always be running but has very tight timing.
- Because the shuffle length can be longer than the interval before the next note you can use this creatively to add extra variation to your pattern playback.
- Sequencers into A[4] and a[5] can yield very interesting results

Troubleshooting:

- If your presets arnt there next time you turn on the ardcore make sure you uploaded the sketch WITHOUT EEPROMreset defined in user_settings.h, or it will reset the eeprom every time you turn on the arduino!
- Dont forget about the knob pickup! If it looks like the knobs are doing nothing, try swinging them through their entire range slowly.
- If your pattern outputs are not what you expect make sure all the shuffle controls are at 0
- If you want to 'reset' the saved presets re-upload the sketch to your ardcore with the EEPROMreset defined
- For any other problems email support [at] asceticunderground [dot] com