Tranzistow v4.8.H [32-bit & 64-bit Windows + Linux Standalone Synthesizer]

https://www.hrastprogrammer.com/tranzistow/

Note: This is not user manual but a brief Tranzistow introduction and description. I simply don't have enough free time to describe all Tranzistow features, modules, parameters and everything.

[*] What is Tranzistow (aka Tranzqomputwerk)?

In global, Tranzistow (this is not a mistake - it is not "Transistow") is a multitimbral multithreaded virtual software synthesizer with 8 independent multimode parts (and much more in private/internal versions). Although it is (to some extent) inspired by Alesis Andromeda, Waldorf Q, Waldorf Microwave II/XT, Yamaha FS1R, DX and SY series, Oberheim Xpander and John Bowen Solaris, it has a unique sound and tons of unique features not found on any other synthesizer. Various forms of synthesis are supported including Additive, Virtual Analog/Modeling (VA), Wavetable (with classic, crossfade and full wave interpolation modes), two completely independent FM engines (oscillator FM + QFM), Formant Synthesis, Ring Modulation, Oscillator Sync, Vectors, Rotors (as in Solaris), various filter models (SVF 12/24dB, 6dB to 24dB Transistor Ladder with fully configurable poles, 24dB Diode LowPass Ladder, ...), Filter Chaining, Filter FM, Comb Filters, extensive modulation capabilities, WaveShaping, freely drawable Contour Generators, exhaustive FX section including quad-channel Chorus/Flanger, Phaser, Delay and Reverb units, etc, etc.

I developed Tranzistow for myself in the first place. My goal was not to (try to) emulate existing (hardware or software, analog or digital) synthesizers (although I was inspired by some of them) but to create something I can actually use in my own music, to master DSP/Assembler/Vector/SSE/AVX/AVX2/GPU/OpenCL/ARM/NEON programming, to refresh my mathematics knowledge and, last but not least - to have fun. As a result, the synthesizer is mostly unconventional, some features (which most users take for granted) could be missing or could look rather strange, user interface may not be everybody's cup of tea because it was designed according to my habits/needs, etc.

Some comments on Tranzistow:

"I will just say that it’s certainly impressive the amount of work you’ve put into your projects, both the synths and the music! Incredible amount of output musically, and so I know you are passionate about it. And I must say, it’s very excellent work! ... I see you did get some inspiration by my Solaris, which is a very nice compliment ... Let me say again how impressive your music production is! It's a lot of work, and beautifully done. I know when I first was making the Scope plugins, each time I would sit and play, sometimes for hours, through certain sounds that just seemed to inspire a musical phrase or a dream idea ... And so much of what is now being produced out there is, to be honest, boring to me. Just running a 16 step sequencer and adjusting filter cutoffs and so on - there's nothing 'skillful' about that, and there's so much of it going on now. But your music is so much more - very deep soundscapes! And the Solaris is very much that type of instrument, with a focus on pad sounds and textures, so I hear what you've done with your Tranzistow synth and I'm quite amazed!" John Bowen (the creator of Solaris synthesizer, https://johnbowen.com/)

"You are totally crazy!" (Holger Drenkelfort of Sonic Core after seeing/hearing Tranzistow presentation in my studio)

"Hrast, i just got to tell you. I have not had such a joy with a synth since impOSCar / VAZ Modular. I don't know what magic you did but the main tone/sound/filters are wonderful. And rumours says you did that in assembler? My hat is off to you dear sir."

"This is my dream synth!"

"I wanted to reaffirm my total appreciation of the DSP programming work you have done with Tranzistow. I have many great synths (hardware, software, analog, digital) but nothing quite compares to the organic qualities that I have been able to coax from your brilliant synthesizer. You have done great work with Tranzistow, make no mistake about it. I can't thank you enough for this brilliant addition to my musical life."

"You can add me to the list of people who consider your Tranzistow one of the best software synth available. It is such a pity it is so obscure, more people should play with it. You deserve much more exposure. I really hope some company will acknowledge your talent and offer you the chance to implement your ideas in hardware."

"Modal should hire HrastProgrammer. I tried wavescaning on his Tranzistow synthesizer, it is by far the best software implementation i've heard so far. You can actually hear that crunchy graining thing where so many others fail."

"Tranzistow is slowly becoming a go-to synth for me. Looks like an 80s NASA control interface, sounds like a dream."

"Tranzistow is a monster. The guy programmed it all in assembly language and some Delphi, even John Bowen praises his work. Check it out! Be warned ... Tranzistow is hard core, haven't even figured it all out. It can do every form of synthesis. The programmer is nothing short of a genius."

"Somebody gave me the URL to these synths because of how bat shit ugly they look. When I took the time to actually dive into them, I found a couple of real gems that turned out to be my favorite synths of everything I've ever owned; bought or otherwise."

"Hrast is, for me, one of the best dev with Gol, Urs and Dmitry from Diversion and Richard from Synapse."

"Diodow is amazing but I have been lost for hours in playing with Tranzistow despite only having the demo version. Yes, the interfaces are, shall we say, idiosyncratic, and their possible complexity is boggling but, even so, HrastProgrammer's synths cannot be praised highly enough IMO."

"Just tried Diodow today, what a great sounding synth, really impressed with the core sound. The UI is a bit retro to say the least but that shouldnt detract from the top notch synth engine, better than many commercial synths hard and soft."

"Hrast is a bad ass DSP programmer!"

"Can we please give HrastProgrammer a statue, his musical output is amazing! A humble genius, for sure."

"I strongly believe that your beautiful synth will have a profound and lasting impact on music in this century and beyond."

"Hrast, you just made me a very happy person! Many thanks again for sharing your talent with us, both as a musician and a programmer. Tranzistow is amazing!"

"For me, I'm happy for anyone who makes something amazing like this (for their own use) to choose to share it on any terms that they wish to offer."

"Programming this thing was a blast."

"And well worth getting. Even if all you do is use the presets, of which there are a ton. This thing sounds amazing and even with all my 'expensive' VSTs, I turn to Tranzistow often."

"If people can use their ears instead of their eyes, they will realize how amazing this synth is."

"I thought I had enough synths, but this one is just incredibly brilliant and fun."

"It would be a real shame if only a very few people knew about this gem."

"It's amazing and sounds beautiful!"

"The phase modulation combined with the amazing sounding filters is worth the asking price alone!"

"Most recently, I found this instrument and was very surprised by its capabilities and sound! It's fine! It was created by a wonderful programmer who knows how to program sound! Wow!"

"I have a suspicion that I will only use this synthesizer, and everyone else will go to the landfill of oblivion!"

"Thank you so much Hrast! I love your synthesizer more and more every day! People must know your instrument! Definitely it is a landmark tool in the history of VST!"

"Tranzistow ... This seems like a well-kept secret."

"Tranzistow is def the most amazing VSTi ever."

"If you want to hear a synth with superlative sound quality go download Tranzistow."

"Tranzistow already was one of the most powerful synths out there, now we can safely say it also crushes any FM synth out there."

"And what I can surely tell you is MAN this instrument is a Pad and Atmosphere MACHINE!! Lots of real simple Monosynth sounds too, but if you're looking to make a film or ambient background track then look no further than Tranzistow!"

"Your generosity with your time and talent is very much appreciated. Many thanks again, I may have a go at building a couple of classic synths with this because the sound is BEYOND competitive compared to the current commercially available emulations. Masterpiece!"

"Tranzistow is The Beast. Once again, Hrast, you're a Master."

"I've just installed this and it sounds fantastic, full of character."

"It's an incredible sounding synth. Just flicking through a few of the thousands of presets and you can tell this one is very special."

"This thing is coded with deep love and affection and it's hard to not get swept along on the journey. His music is also very special."

"How deep is the modulation? Extremely f**king deep."

"The downright masochistic GUI is the main thing that's keeping me from using this synth."

"But yes, Tranzistow is certainly one of the very best software synths out there."

"The UI was designed by Satan."

"Once you get to know him, Satan is really a nice guy, and get things done really quickly."

"I really don't understand what you guys don't understand about Tranzistow, you read it from left to right, enable the modules you want etc. There are indeed some tricky trivial parts, but nothing too complicated. It's all about persistance, you either want or don't want to control the beast."

"It sounds exceptional, you can just play the presets to get an idea of it. Most importantly, it has character and Mojo."

"Gosh I love this synthesizer. One of the best-sounding and most capable I've ever used."

"Tranzistow is a fantastic synthesizer! A pure diamond."

"I agree with all the good things said about Tranzistow, it might be the best sounding VST synth existing."

"I see lots of complaints about the UI, and I quite agree with all of that, but at the same time, I'm thinking, imagine if the UI/workflow was on par with the sound, what would there be left for other synths? I think maybe it's better this way, better for the ecosystem :)"

"Yeah, reverb is super dense and smooth. Definitely at the same level as Valhalla verbs."

"I fell in love with this synth. Most of the time I don't really have a clue what I am doing, but the outcome is incredible."

"And in fifth position, the real cherry on top of the cake! This is a love at first glance! This is a sound design geek's dream! The pinnacle of programming! It is just a monster in terms of capabilities! Synthesizer Tranzistow! What can this synthesizer do? It covers almost all kinds of synthesis and it has excellent analog sound thanks to its wonderful filters. In addition, it comes with just a huge number of presets that are impossible to listen in one day! But there is one drawback. It has a very, very unfriendly interface. The fact is that initially Hrast wrote it exclusively for himself, so it was easy for him to work with it, knowing where everything located. But for me, it was immediately very difficult to navigate."

"Biggest thanks of all to HrastProgrammer. Incredible software. Incredible music. Incredible human being."

"Thank you Hrast. What an amazing synthesizer you have created!"

"And even if it's terse and somewhat obscurantist interface is not to your liking, just flick through the many bread and butter and killer cinematic soundtrack patches (for it has both) and soon something will click with your track. There are thousands of them."

"There isn't a synth or preset collection that can even come close at this price. But it's rude to talk about money. This is art. Even if it is underpinned by the most strict principles of Science."

"It is a thing of beauty and joy that has been created. It inspires devotion."

"This might be the best sounding synth ever made. Knocked up by a real musician and Hacker of code."

"This beautiful monster of a synth."

"Words can not describe the power and quality if this piece of genius software."

"One of the best and deepest softsynth ever created, it can do it all with flying colours. It's probably the only synth you will ever need."

"There's a certain wonder in Tranzistow. You don't need to dig deep to find it. Just download it!"

"Hrast is a class act. His music, if you haven't already checked it out, is also first rate!"

"Is it that Tranzistow actually sounds better than other softsynths or is it just me? Maybe it's your patch programming but it seems to me that there's more depth to the sound and I find it more engaging than most softsynths, maybe it's confirmation bias but is there something special about the way it produces sound? I'm a bit scared you'll burst my bubble!"

"That's brilliant, I think I'm right, it sounds lovely, Serum is horrible in comparison. I have no idea what you did but I'm a fan!"

"You sir are a gemstone!"

[*] Hrastow Tranzistow - Public Versions

Tranzistow can be used with VST 2.x compatible hosts/DAWs. Just download the latest package and copy Tranzistow DLLs into your VST plugins folder so the host can find them. Also copy TranzistowPatches and TranzistowWaves folders. But, and a big BUT - if it doesn't work (or doesn't work well) with a particular host then bad luck, sorry. Life is too short to fight with all those hosts out there and that's one of the reasons why I decided to develop a standalone "Hrastow Tranzistow" 32/64-bit package with my own application environment called Hrastow. This environment works as a standalone synthesizer controlled by MIDI (two main and one control input are provided) and also contains ASIO host and a simple VST 2.x host with 2 FX slots for loading two plugins in serial. The main purpose of those slots is the ability to use some external VST effects to process Tranzistow outputs if needed, but they can also load other VST synthesizers like my own Diodow synthesizer, for example. Furthermore, there is an internal oversampler, so you can achieve 88.2/96kHz quality on systems with 44.1/48kHz sample rates. When working with Hrastow you have to control it over MIDI from the sequencer (either from a separate computer or using LoopBe virtual MIDI driver) and mix it using a real mixer (or some virtual mixer directly on the computer).

Tranzistow has been released as a "mandatory donationware" with donations ranging from 50 EUR up to 200 EUR or more depending on how much Tranzistow will be worth to you, on how many computers do you intend to use it, etc. Only PayPal donations to hrastwerk@hrastprogrammer.com are accepted. PayPal fees have to be paid by yourself. Please, keep in mind this still isn't a commercial project so no support, no assistance in getting it to work, no troubleshooting of your system, etc. Of course, I will make enhancements and fix bugs when I encounter them simply because this is the synthesizer I am using on a daily basis and will be using in my music for years to come. This also means that I can actively test it on Windows XP, 7 and 10 with various ASIO drivers for audio interfaces I have (MOTU 896mk3 Hybrid, E-MU 1820M, 1212M and 0404, Focusrite Scarlett, ESi 1010e, ASIO4All). Donations cannot be returned, so think twice before donating, check if it works for you in the current state, decide how much it is important to you and then donate the appropriate amount. And don't insult yourself, me and Tranzistow with donations below 50 EUR! If you think this synthesizer isn't worth at least 50 EUR then it isn't for you, sorry.

Please note that Tranzistow cannot be used without donation/registration anymore, at least - not used properly because there will be various sound artifacts and occasional missing notes until donated/registered. Users who donated have to send me their full name and PayPal address in order to obtain an ID and only after putting this ID into Tranzistow.ini those artifacts will go away.

The following Windows and Linux versions are available as public Hrastow Tranzistow synthesizer package:

(*) Tranzistow v8*4/32 ... 32-bit, 8 parts with SSE3 engine and 4-way multithreading (1 main + 1 sub part)
(*) Tranzistow v8*4/64 ... 64-bit, 8 parts with SSE3 engine and 4-way multithreading (1 main + 1 sub part)

All above versions have up to 128 voices total polyphony and always operate in multithreading mode (singlethreading has been removed and is not available anymore). Of course, the real polyphony (the maximum polyphony which can be achieved under real conditions) depends greatly on the complexity of the sounds and the speed of the CPU. In general, 64-bit SSE3 versions are 5-8% faster than 32-bit versions, depending on the patch structure.

Note that the above configurations are possible only if Tranzistow runs under Hrastow. If you are running it under any other host or DAW then it will automatically revert to 1 instance = 1 thread = 1 core = 1 part = 1 patch = 16 voices mode, so if you need more channels/parts then load more instances, just like with any other plugin:

(*) Tranzistow t1/32 ... 32-bit, 1 part with SSE3 engine and no multithreading
(*) Tranzistow t1/64 ... 64-bit, 1 part with SSE3 engine and no multithreading

Linux versions always work in multitimbral / multithreading mode.

[*] Conditions of Use

(*) Tranzistow is my private project and I developed it for myself in the first place. So, if you don’t like it then don't use it, simple as that ;-) I really don't have the time and patience to argue about something someone doesn't like, or whether "it is or isn't analog enough" etc.

(*) The synthesizer is "feature finished" - no new features will be added so, please, don't ask for a new feature.

(*) I am not responsible for anything you do with this software - use it at your (and only your) own risk!

(*) You may not alter this software in any way, including changing or removing any messages.

(*) You may not decompile, reverse engineer, disassemble, modify, distribute, rent or resell this software, or create derivative works based upon it.

(*) Donations are for what I've done, not for what I should do. So, please, don't have any expectations about my future work on the base of the donation.

(*) I've always tried to do my best to support as many hosts as possible, but - again - if it doesn't work (or doesn't work well) with a particular host then bad luck, sorry. Life is too short to fight with all those hosts out there. I also don't own any host/DAW and don't have intention to buy any of those.

[*] Technical Details

(*) Requirements: 32/64-bit Windows XP/7/Vista/8/10 and SSE3/AVX2 capable 32/64-bit CPU.

(*) The complete low-level audio/processing/control/MIDI/FX engine is written in 100% Intel x86/x64 assembler utilizing SSE3/AVX2 instructions for vector processing - that's the main reason why there are 4 (and multiples of 4) elements of each module (oscillators, filters, etc.) because I am processing 4 vector elements simultaneously (or 2 consecutive voices with 4 elements in AVX2 version).

(*) 32-bit integers and single-precision floating point numbers are used for all processing as a proof that, if done properly, you don't need 64-bit and double-precision to achieve a high-quality sound. The only place where I had to switch to double-precision is the sample engine because I wanted sample oscillators to be able to handle large files with a high resolution.

(*) User interface and all other high-level tasks are written in Borland Delphi and Free Pascal / Lazarus.

(*) The engine is optimized for 96kHz sample rate. Other sample rates can be used, of course, but 96kHz is the only officially supported sample rate. Most high-end hardware synthesizers (like Solaris, for example) use 96kHz sample rate as well, so I didn't want to make compromises here. On today's computers it doesn't have much sense to intentionally degrade a sound by using lower sample rates just to save some CPU, and 192kHz is really an overkill, especially with 2x and 4x oversampling when Tranzistow engine works at 192kHz and 384kHz internally.

(*) Some internal wavetables are modeled to sound similar to various wavetables from existing hardware synthesizers, but they are generated in a different way and they are not copies of wavetables from those synthesizers. They are also of much higher quality compared to the originals (due to highest-quality wavetable generation algorithms in Tranzistow).

(*) I wanted to achieve the maximum possible sound quality and the maximum possible amount of features, so Tranzistow can use quite a lot of CPU, but there are various tips to lower CPU usage in cases where maximum quality is not really needed, especially if working with 96kHz sample rate (recommended sample rate for Tranzistow).

[*] Installation

There is no installation. Just unpack files from the archive into some folder, run the executable (Hrastow32.exe or Hrastow64.exe) and that's it. In order to avoid "Unknown publisher" crying from Windows you should do the following prior to unpacking: right-click on the archive, select "Properties", check "Unblock", then click on "Apply" and "OK". Again, do this *prior* to unpacking! And don't forget to repeat this process every time you download a new version.

Note: I switched from ZIP to 7Z format and it seems like this problem is not present with 7Z archives.

[*] Private Versions

Beside public ones, there are enhanced private versions which I decided to keep for myself and for those who donated some remarkable amount, as a gift. They are not available in regular donationware packages.

The following Windows versions are private:

(*) Tranzistow v16*4/32 ... 32-bit, 16 parts with SSE3 engine and 4-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v16*4/64 ... 64-bit, 16 parts with SSE3 engine and 4-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v24*4/32 ... 32-bit, 24 parts with SSE3 engine and 4-way multithreading (1 main + 5 sub parts)
(*) Tranzistow v24*4/64 ... 64-bit, 24 parts with SSE3 engine and 4-way multithreading (1 main + 5 sub parts)
(*) Tranzistow v24*6/32 ... 32-bit, 24 parts with SSE3 engine and 6-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v24*6/64 ... 64-bit, 24 parts with SSE3 engine and 6-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v32*8/32 ... 32-bit, 32 parts with SSE3 engine and 8-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v32*8/64 ... 64-bit, 32 parts with SSE3 engine and 8-way multithreading (1 main + 3 sub parts)

(*) Tranzistow x8*4/64 ... 64-bit, 8 parts with AVX2 engine and 4-way multithreading (1 main + 1 sub part)
(*) Tranzistow x16*4/64 ... 64-bit, 16 parts with AVX2 engine and 4-way multithreading (1 main + 3 sub parts)
(*) Tranzistow x24*4/64 ... 64-bit, 24 parts with AVX2 engine and 4-way multithreading (1 main + 5 sub parts)
(*) Tranzistow x24*6/64 ... 64-bit, 24 parts with AVX2 engine and 6-way multithreading (1 main + 3 sub parts)
(*) Tranzistow x32*8/64 ... 64-bit, 32 parts with AVX2 engine and 8-way multithreading (1 main + 3 sub parts)
(*) Tranzistow x48*8/64 ... 64-bit, 48 parts with AVX2 engine and 8-way multithreading (1 main + 5 sub parts)

The following Linux versions are private:

(*) Tranzistow v16*4/32 ... 32-bit, 16 parts with SSE3 engine and 4-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v16*4/64 ... 64-bit, 16 parts with SSE3 engine and 4-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v24*4/32 ... 32-bit, 24 parts with SSE3 engine and 4-way multithreading (1 main + 5 sub parts)
(*) Tranzistow v24*4/64 ... 64-bit, 24 parts with SSE3 engine and 4-way multithreading (1 main + 5 sub parts)
(*) Tranzistow v24*6/32 ... 32-bit, 24 parts with SSE3 engine and 6-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v24*6/64 ... 64-bit, 24 parts with SSE3 engine and 6-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v32*8/32 ... 32-bit, 32 parts with SSE3 engine and 8-way multithreading (1 main + 3 sub parts)
(*) Tranzistow v32*8/64 ... 64-bit, 32 parts with SSE3 engine and 8-way multithreading (1 main + 3 sub parts)

All private versions run in multithreading mode and have up to 128/256/384/512/768 voices (8/16/24/32/48 parts) total polyphony. Of course, the real polyphony (= maximum polyphony which can be achieved under real conditions) depends greatly on the complexity of the sounds and the speed of the CPU. In general, 64-bit SSE3 versions are 5-8% faster than 32-bit versions and AVX2 versions are generally 40-60% faster than 64-bit SSE3 versions, depending on the patch structure. No AVX2 Linux version of Tranzistow yet.

There is no sound differences in synthesis engine between "private" and "public" versions. The main difference is more multitimbral parts in "16/24/32/48" versions and bigger total polyphony in "x" versions together with maby advanced multitimbral features, but they all sound exactly the same as far as audio/synthesis engine is concerned. For the most of my previous legacy music I used 64-bit v8 version. On newer tracks I started to use Tranzistow exclusively, so I decided to develop 24-part version because I needed more parts in those "Tranzistow-only" tracks. It was still 4-core only and each CPU core processed 6 parts (instead of 4 parts in v8/v16). The reason is simple: I only have 4-core CPUs (i7-4790K and i7-2700K, both with E-MU 1820M audio interfaces), so I had to squeeze as much as I can from those 4 cores. This is the culmination of my work because I am now able to create complete tracks on Tranzistow synthesizer alone, from sound synthesis to effects, with everything playing in realtime and all parts controlled over MIDI from my KalquLab sequencer. I cannot describe how happy I am because now I can do everything I ever dreamt of. I don't need no hardware synthesizers anymore.

Recently, I developed Tranzistow x48*8/64 engine which can use 8 cores to the full potential because I bought an Intel i7-11700K CPU and wanted to utilize all that power now available. This new setup has 8 cores up to 5.0GHz and is built around MOTU 896mk3 Hybrid USB2/Firewire audio interface which runs at 48kHz with 256 samples buffer. Hrastow runs with 2x oversampling at 96kHz and the buffer has been divided internally into 64 sample frames, so everything runs super tight under 0.667 ms. Multithreading mode 2 has been employed and Tranzistow is running with additional internal 2x oversampling on all patches, so the engine runs at 192kHz. With all this and many AVX2 optimizations I made during the last few months, I am now able to achieve the maximum polyphony of 254 firm voices in total using my ultra-complex test patch for benchmarking. As a comparison, on my older Intel i7-4790K I was able to achieve 86 voices with the same sample/oversample rates, but under 4ms only. Furthermore, I also integrated my KalquLab sequencer into Hrastow, so I finally can do everything on just one machine without any latency and with per-core sample-accurate MIDI in Tranzistow engine. Simply amazing! This is now my setup for the years to come :-)

[*] Internal Versions

Those versions aren't yet available to the public under any circumstances:

(*) Tranzistow v8*4+/32 ... Tranzistow v8*4/32 with Additive+FM/GPU engine
(*) Tranzistow v8*4+/64 ... Tranzistow v8*4/64 with Additive+FM/GPU engine
(*) Tranzistow v16*4+/32 ... Tranzistow v16*4/32 with Additive+FM/GPU engine
(*) Tranzistow v16*4+/64 ... Tranzistow v16*4/64 with Additive+FM/GPU engine
(*) Tranzistow v24*4+/32 ... Tranzistow v24*4/32 with Additive+FM/GPU engine
(*) Tranzistow v24*4+/64 ... Tranzistow v24*4/64 with Additive+FM/GPU engine
(*) Tranzistow v24*6+/32 ... Tranzistow v24*6/32 with Additive+FM/GPU engine
(*) Tranzistow v24*6+/64 ... Tranzistow v24*6/64 with Additive+FM/GPU engine
(*) Tranzistow v32*8+/32 ... Tranzistow v32*8/32 with Additive+FM/GPU engine
(*) Tranzistow v32*8+/64 ... Tranzistow v32*8/64 with Additive+FM/GPU engine

(*) Tranzistow x8*4+/64 ... Tranzistow x8*4/64 with Additive+FM/GPU engine
(*) Tranzistow x16*4+/64 ... Tranzistow x16*4/64 with Additive+FM/GPU engine
(*) Tranzistow x24*4+/64 ... Tranzistow x24*4/64 with Additive+FM/GPU engine
(*) Tranzistow x24*6+/64 ... Tranzistow x24*6/64 with Additive+FM/GPU engine
(*) Tranzistow x32*8+/64 ... Tranzistow x32*8/64 with Additive+FM/GPU engine
(*) Tranzistow x48*8+/64 ... Tranzistow x48*8/64 with Additive+FM/GPU engine

[*] Special Note

Only 32/64-bit Windows VST 2.x and Linux standalone versions are available. There won't be VST 3.x, Macburger, Applesaft, AU, RTAS, AAX, iCrap, iDiot, dumbphone or any other version, ever!

[*] Final Words

***** Individuality, not conformity! *****

Tranzistow is a synthesizer for pure electronic artists, sound programmers, synth lovers and creators of true electronic music, it doesn't pretend to be anything else. So, if you need beautiful, unique, powerful and most complex synthetic electronic sounds of the highest quality - then you are at the right place. It isn't really aimed at keyboard players and, although it can be used for such duties, there are much easier ways to obtain the usual keyboard sounds. It is like buying a ultra-capable 4x4 offroad vehicle and use it for everyday driving - you can drive it on-road, of course, but you must take it off-road from time to time, so it can do what it is supposed to do. And isn't it simpler and much more convenient to use a small city car to go shopping, if you need a car only for this?

[*] Patches / Presets

The Tranzistow package comes with over 4000 of my own patches which can be easily browsed through by clicking on the "Browse" button located on the "Patch/FX" page. Those patches illustrate the way I work and the way I use synthesizers for, so they probably don't appeal to everyone. They also demonstrate a huge range and variety of sounds Tranzistow can make and cover most of Tranzistow functionality, albeit some features are touched very lightly and just in a few patches. Unfortunately, I ran out of time while preparing Tranzistow for production and couldn't allocate more time for patch creation :-(

Tranzistow is a hugely complex and powerful synthesizer which looks and feels more like a fully equipped Jeep Wrangler Rubicon with 2.8 CRD engine, winch and everything, than a BMW Mini or Mercedes C-Klasse. So, most of those patches are very complex and many of them have extralong envelope tails and/or use unison which can eat quite a lot of CPU, although none of them use more than 80% of one CPU core on my i7-4790K @ 4.4GHz with 16 voices of polyphony, 96kHz sample rate and 2x oversampling (this is 64 extremely complex voices across 8 multimode parts on a quad core system at 96kHz, or 128 voices at 48kHz, all with 2x oversampling). There are some advices on how to optimize CPU usage later in this document. Of course, with less complex voices CPU usage will be much lower.

Regarding patch names: I had to name them somehow and you will find everything here. Some names do have a connection to the sound itself and some don't, so if you find something like "analog", "Moog" or "whatever" it just means that a particular sound reminded me somehow of "analog", "Moog" or "whatever". It doesn't mean it sounds exactly like "analog", "Moog" or "whatever" because I couldn't care less if it sounds like "analog", "Moog" or "whatever", neither I tried Tranzistow to be emulation of "analog", "Moog" or "whatever". Furthermore, due to artistic freedom, many names are combinations of words which are not necessarily syntactically or semantically correct, and which don’t necessarily have any meaningful sense to anyone except me.

In addition to that, there is another bank with ~28000 Yamaha DX7 patches I reworked for Tranzistow QFM engine:

https://www.hrastprogrammer.com/hrastwerk/download/TranzistowQFM.zip

Most of them probably don't sound very close to the actual DX7 but they should serve as an excellent starting point for making new patches as there aren't many QFM patches that come with Tranzistow.

[*] Building Blocks & Modules

Starting on the next page ...

[*] Main Oscillators (Osc/Main Page; Osc #1 ... Osc #4 Sections)

There are 4 main oscillators based on Additive, Wavetable and Virtual Analog/Modeling (VA) engine. Those can go from 0Hz to 24kHz and are fully antialiased with a powerful, fat bass and clean & crispy highs. Every oscillator has its own internal sync oscillator, so all of them can be synced at the same time if needed. All of them can be frequency modulated by any other audio/modulation source inside the synthesizer. Additive part of the engine can produce up to 1024 harmonics and automatically removes harmonics over Nyquist when pitching up, and adds harmonics when pitching down. Wavetable part of the engine can work without any crossfading/interpolation between individual waves (Wave and Wave+Pulse/Ramp modes, this is "classic" wavetable mode with integer wave index as found in Waldorf Microwave, for example), with crossfading between individual waves (XWave and XWave+Pulse/Ramp modes), with interpolation between individual waves ([Wave] and [Wave+Pulse/Ramp] modes) as well as with both crossfading and interpolation between individual waves ([XWave] and [XWave+Pulse/Ramp] modes). Crossfading (XWave) modes are also used when pitching up/down for smooth transitions when adding or removing harmonics. Basic waveform of the oscillator can be shaped into pulse/square and ramp/triangle waveforms, which can then be combined with the basic waveform (in Pulse/Ramp modes). Inside each oscillator there is a thermal drift module for a more analog sound. Main oscillators can use a lot of CPU power in wave/crossfade and wave/interpolation modes. That's why there is also a "thru-zero" mode (denoted by * in front of the name) where transition between waves in the wavetable occurs on start of the cycle only, minimizing clicks without the need to use wave/crossfade or wave/interpolation modes (while "thru-zero" not being as smooth as those modes, of course).

In addition to built-in waves/wavetables, there are 27 user banks with 100 waves each (User00.wave ... User99.wave files in TranzistowWaves/A..Z directories). User00.wave from the main bank (TranzistowWaves) contains all Ensoniq SQ waves while all others are free to use and will be loaded automatically if one exists. The format of user wave files are out of the scope of this document. Furthermore, there are four user-definable wavetables per patch, each containing from 1 to 128 individual waves with size of up to 256 samples, bandlimiting and interpolation of missing waves. All internal and user waves can be used here. Those wavetables are accessible through WT #1 ... WT #4 buttons after activating them by clicking on the "Osc" button/section from "Osc/Main" page.

[*] Auxiliary Oscillators (Osc/Aux Page; AuxOsc #1 ... AuxOsc #4 Sections)

There are 4 auxiliary oscillators based on a different Virtual Analog/Modeling (VA) engine. Those can go from 0Hz to 24kHz and are fully antialiased with a powerful, fat bass and clean & crispy highs, just like main oscillators. Every aux oscillator has its own internal sync oscillator, so all of them can be synced at the same time if needed. All of them can be frequency modulated by any other audio/modulation source inside the synthesizer. They are also partially connected to additive and wavetable engines, so some features of main oscillators are present here as well. Basic saw waveform of the oscillator can be shaped into pulse/square and ramp/triangle waveforms, which can then be combined with the basic waveform (in Pulse/Ramp modes). Inside each oscillator there is a thermal drift module for a more analog sound. Aux oscillators use less CPU power and are thus recommended when full additive/wavetable functionality is not needed.

[*] Sine Oscillators (Osc/Main & Osc/Aux Pages)

Both main and auxiliary oscillator modules can be switched to sinewave module with 4 parallel sine oscillators which can be synced and frequency modulated just as Main/Aux oscillators and have the same set of basic features (but without thermal drift). Sine oscillators are antialiased by definition and use little CPU power. They are very useful with frequency modulation synthesis or just to beef a sound.

[*] SuperSaw Oscillators (Osc/Main & Osc/Aux Pages)

Both main and auxiliary oscillator modules can be switched to SuperSaw module with 4 parallel fully bandlimited SuperSaw oscillators which can be synced and frequency modulated just as Main/Aux and Sine oscillators and have the same set of basic features, including thermal drift module. Those don't have some advanced features like wavetables, morph, etc. but they are much more CPU friendly and can be used when advanced functionality is not really needed. Furthermore, bandlimiting filter can be freely adjusted so you can have variable amount of aliasing, ranging from no aliasing at all down to the zero point and terrible aliasing like in the good old days of lo-fi digital synthesis :-)

Each SuperSaw oscillator consists of up to 16 individual (antialiased) saw sub-oscillators (this is 64 saw sub-oscillators per voice) detuned against each other for extremely thick sound very popular in Trance genre, but perfectly usable in all other EM genres as well. Various detune type/level modes are provided and all sub-oscillators are synced individually for a huge sound. Depending on the number of sub-oscillators, SuperSaws can use quite a lot of CPU power. It is very important to note that Tranzistow will always allocate enough CPU for the maximum number of sub-oscillators among all oscillators, so it is much more effective to have 4 SuperSaw oscillators with 4 sub-oscillators each than one SuperSaw oscillator with 16 sub-oscillators and other three oscillators set to zero.

[*] HyperSaw/HyperSync/HyperWave Oscillators (Osc/Main & Osc/Aux Pages)

This is the result of my experimentations with very high sample rate for oscillators. Those oscillators are not bandlimited but instead work at much higher sample rate internally. Hypersampling rate can be adjusted and, by default, on 44.1kHz with 2x oversampling they work at 1.4MHz while on 96kHz with 2x oversampling they work at 2.3MHz. The main advantage of such configuration is that sweeping the frequency of the master oscillator with oscillator sync is continuous while on the other oscillators it is somewhat quantized which could be noticeable on high notes. Not that I ever sweep master oscillators (and especially not at such high pitches), because it sounds "meh", but anyway - if it is needed then HyperSaw oscillator is the best candidate for this. The disadvantages are high CPU usage as well as some aliasing at lower sample rates and very high oscillator pitch. There is also an alternate HyperSync module which is especially suitable for hard-syncing of the oscillators (but using even more CPU than HyperSaw).

HyperWave oscillators use waves and wavetables instead of a saw wave to generate the sound in the same way as on the legendary PPG Wave synthesizers. All "Hyper" modules are more on the experimental side because I wanted to explore some unusual sound generation methods. Strictly speaking, they are not needed for everyday patch creation but can be useful when more "harshness" or "roughness" is desired for a particular sound because, due to the huge bandwidth, they can produce a lot of additional harmonics, especially in combination with FM.

[*] FormantSaw Oscillators (Osc/Aux Page)

Those are a special sort of oscillators which generate sound through some kind of formant simulation. Basically, formants are resonant spectrum peaks and they sound similar to the combination of frequency modulation and bandpass resonant filters. Number of formants per oscillator can be set from 1 to 8 and they can be morphed from zero to full frequency giving them somewhat "throaty" and "vocal" quality. Formant oscillators have an internal feedback loop and respond to sync and FM as well.

[*] Triangle Oscillators (Osc/Main & Osc/Aux Pages)

Triangle oscillators use very little CPU, even less than Sine oscillators. They are great as sub-oscillators (to beef up the sound), or for those beautiful pure high-pitched electronic sounds which I adore, but are not limited to such sounds, of course. FM always operates in TrueFM mode on those oscillators.

[*] Oscillator Sync (Osc/Main & Osc/Aux Pages)

As said before, each Main/Aux oscillator has its own internal virtual sync oscillator, so all of them can be synchronized at the same time if needed. Oscillator sync is fully antialiased. In addition to internal sync oscillators, each oscillator has a "classic" inter-oscillator sync module where any (slave) oscillator can be synchronized to any other (master) oscillator, similar to classic analog synthesizers but much more powerful because you can have up to 8 sync master/slave oscillators per voice. I still recommend using virtual sync oscillators wherever possible because they are less CPU demanding. SuperSaw/HyperSaw/FormantSaw oscillators can also act as masters and slaves, as well as all QFM operators.

[*] Sample Oscillators (Osc/Main Page; Osc #1 ... Osc #4 Sections)

There are 4 high-quality sample oscillators per voice implemented as sub-modules inside main oscillators - they share common pitch/frequency parameters together with the same outputs, so they can be used as audio/modulation sources in tandem with the regular oscillators + sample output can be further processed through Tranzistow engine as any other audio source inside the synthesizer. Sample oscillators can be upsampled up to 32x, with up to 4 antialiasing filters and two interpolation modes (linear and high quality), so a required balance between a sound quality and CPU usage can be achieved for a particular application. The result is that you can transpose a sample up for several octaves without audible/measureable aliasing artifacts, even at lower quality settings and low CPU usage. The engine can load 8/16/24/32-bit PCM and 32-bit FP mono/stereo WAV files. To load a sample click on the oscillator section name (Osc #1, Osc #2, Osc #3, Osc #4). In case of stereo samples - if you load a sample into oscillators 1 & 3 then a left channel will be loaded, otherwise a right channel will be loaded. To remove the sample press and hold Ctrl key and click on the appropriate oscillator section name. The oscillator section name will have an asterisk on front of it if a sample has been loaded into the connected sample oscillator. Thermal drift is not available with sample oscillators but FM and sync are fully supported. Those are mostly useful with short samples (cycles) but if applied creatively they can be used with longer samples as well.

Furthermore, sample engine contains a GrainStatic module. This is an extension which uses sample oscillators to produce varius granular effects and came as a result of my research in granular synthesis field. Both granular and sample oscillators can be mixed inside a single voice. FM and sync are implemented for granular oscillators, too.

[*] Oscillator Frequency Modulation (Osc/Aux & Mixer/FM Pages; Osc FM & AuxOsc FM Sections)

All Main/Aux oscillators can be frequency modulated by any audio or modulation source, including the oscillator itself (so FM feedback is possible). Each oscillator is actually one FM operator (either modulator or carrier) with two independent FM inputs. By default, Tranzistow frequency modulation works as "Yamaha-style FM", which is actually a phase modulation, but people usually associate FM with "Yamaha FM", so I will stick to this terminology. TrueFM ("Linear FM") mode can be turned on individually for each of the 4 slots in each FM module. All oscillator modules, except Triangle, support both modes. Triangle FM always operates in TrueFM mode. Additional linear or exponential FM is also possible through audio rate modulation matrix. Furthermore, phase locking has been implemented for all oscillators. It is not very useful alone (because phase is not advancing and thus the particular oscillator is not producing any sound) but if used with FM it will, more or less, act as phase distortion (as used by Casio, for example) opening a new array of possible sounds. Additional FM side-inputs are available through (audio) modulation matrix.

[*] PulseWidth Modules (Osc/Main & Osc/Aux Pages)

All Main/Aux oscillators, except Sine and Triangle, have an internal PulseWidth module to generate square/pulse/ramp waveforms or to achieve various other PW-like effects. For each oscillator module (where applicable) there is a complementary "-PW" module without PW and no waveform mixer which uses less CPU (especially noticeable with SuperSaw, HyperSaw and HyperWave oscillators). It is recommended to use those variants when PW is not needed.

[*] QFM - Advanced Frequency Modulation (QFM/1 & QFM/2 Pages; OP #1 ... OP #8 Sections)

In addition to oscillator FM, Tranzistow has an advanced FM engine called QFM which consists of 8 operators grouped into two 4-OP FM modules (OPX and OPY), so only one module can be used if only 4 operators are needed to conserve CPU power. Those modules work in parallel with the existing oscillators - all operators can be processed through the rest of Tranzistow engine and used as audio/modulation sources, just like all other Tranzistow audio/modulation sources. QFM engine is totally independent of Main/Aux oscillator FM modules, so all of them can be used at the same time. Each QFM operator is an advanced version based on Yamaha FM operators and has all the usual FM parameters with lots of additional features. Operator matrix for both OPX (OP 1..4) and OPY (OP 5..8) modules is freely configurable meaning that you can use everything as FM sources - including other operators, oscillators, filters, etc. Furthermore, there is an additional fixed operator routing matrix (in 4x4-OP and 8x8-OP modes) where each 1..4 operator (OPX module) can be routed to each 1..4 operator and each 5..8 operator (OPY module) can be routed to each 5..8 operator, all with configurable amounts. Each operator has its own (configurable) level curve and can modulate itself so you can have feedback on all of them. There is a set of 32 DX7 and 8 DX9 algorithms - when you choose some of them the operator matrix is configured to match the appropriate algorithm, but you are free to reconfigure it afterwards. And each operator has its own 10-stage loopable envelope which can ba used as the regular modulation source as well. Operator waveform is not fixed so you can use not only sine wave but all other waveforms available in Tranzistow, including waves, contour generators, rotors, etc. TrueFM mode is available in QFM engine, as well, and can be turned on individually for each operator.

I converted thousands of Yamaha DX7 patches into Tranzistow format. Due to various differences (operator level scaling, interpolation, much higher waveform quality, etc.) between Yamaha and Tranzistow FM engines those patches won't necessarily sound the same (or even close). And my goal wasn't to try to match the original sound because I wanted to develop my own engine with its own sound, so it has been tweaked according to my needs and according to what I consider a good FM sound. The possibility to use DX7 patches was a real bonus and most of them will sound very similar to the originals but some will need a little tweaking. Of all patches I tried I didn't find a single one which wasn't useful after importing, even without much tweaking. And last but not least - QFM editor must be activated by holding Shift while selecting Osc/Main or Osc/Aux page. Although both set of pages (Osc/Main+Osc/Aux and QFM/1+QFM/2) cannot be active at the same time all modules are still active and you can use Main/Aux oscillators together with QFM operators without problem. QFM Mode and Algorithm selectors are located on alternate hidden Osc/Main page.

QFM engine also features a so-called F-Operator OPX/OPY modules for (but not limited to) formant sound generation. F-Operators can operate in serial (F-Operator is connected in series with FM inputs) or parallel (F-Operator works in parallel with FM inputs). Furthermore, fixed formant modes are implemented for each F-Operator. You can set center frequency (so the resulting formant generated by the operator won't move as you play the notes, except if you modulate the center frequency) as well as bandwidth to adjust the number of harmonics around the formant. Both parameters are active in fixed formant modes only. If you set center frequency to zero then a fixed formant will be turned off for the particular operator despite currently active mode.

All inputs #2 on both QFM engine and oscillator FM have additional 1-sample delay which can be used to smooth the modulation. This is especially useful with FM feedback.

[*] Rotors (Osc/Main & Osc/Aux Pages)

Four parallel units of the same kind can be combined into a single sound source called "rotor" which can then be used as an oscillator. Just like in regular oscillator where phase is rotating through the cycle, in a rotor phase is rotating between 4 sound sources. Depending on the amount of interpolation this rotation can be smooth (crossfading between sources) or rough/sharp/harsh. Because rotor is acting like an oscillator to use it you have to change Main/Aux oscillator waveform to a rotor. Those have [1,2,3,4] after the name - for example, "Osc [1,2,3,4]" is a rotor consisting of four main oscillators while "LFO [1,2,3,4]" is a rotor of four LFOs. In addition to rotors, Main/Aux and Sine oscillators can directly pass another sound sources through themselves, like another oscillator, filter output, ring modulator, etc. This allows building complex configurations of various sound sources, including feedback from the output back to the input.

[*] Mixer & Balancer (Mixer/FM Page; Mixer #1 ... Mixer #4 Sections)

After oscillators and operators there goes a mixer. Audio routing inside Tranzistow consists of 4 mono paths combined into 2 stereo paths (lower and upper) at the end of the voice chain (before FX units). Those 4 paths are routed in parallel from oscillators through filters to the amplifier:

Osc1 => Filter1 => PostFilterMixer1 & Amplifier1
Osc2 => Filter2 => PostFilterMixer2 & Amplifier2
Osc3 => Filter3 => PostFilterMixer3 & Amplifier3
Osc4 => Filter4 => PostFilterMixer4 & Amplifier4

In this configuration it is not possible to "cross the paths" e.g. you cannot, for example, route Osc1 to Filter2 etc. To overcome this you can turn on the balancer which allows routing between paths 1 & 2 as well as between paths 3 & 4 (this is the same configuration as in my Diodow synthesizer and is used when importing Diodow patches):

Osc1+2 => Balancer1+2 => Filter1+2 => PostFilterMixer1+2 & Amplifier1+2
Osc3+4 => Balancer3+4 => Filter3+4 => PostFilterMixer3+4 & Amplifier3+4

But it is still not possible to route audio from paths 1 & 2 to paths 3 & 4 and vice versa. To allow for completely free audio routing there is a dedicated mixer with 4 paths and 4 audio inputs per each path:

Mixer1 => Filter1 => PostFilterMixer1 & Amplifier1
Mixer2 => Filter2 => PostFilterMixer2 & Amplifier2
Mixer3 => Filter3 => PostFilterMixer3 & Amplifier3
Mixer4 => Filter4 => PostFilterMixer4 & Amplifier4

Furthermore, the mixer can be combined with balancer for additional flexibility:

Mixer1+2 => Balancer1+2 => Filter1+2 => PostFilterMixer1+2 & Amplifier1+2
Mixer3+4 => Balancer3+4 => Filter3+4 => PostFilterMixer3+4 & Amplifier3+4

Everything else is variable and (almost) everything (be it audio or control signal, all MIDI controllers included) can modulate (almost) everything, up to 4x oversample rate (depending on how things are configured inside a particular patch).

[*] Vectors (Mixer/FM Page; Mixer #1 ... Mixer #4 Sections)

As part of the mixer vectors allow mixing 4 sound sources in the X-Y plane e.g. crossfading between 4 sound sources in the X-Y plane. There are 4 vectors inside a mixer - one for each mixer path.

[*] Noise (Mixer/FM Page; Noise #1 ... Noise #4 Sections)

After the mixer there are 4 noise generators which can be individually mixed and filtered (so all forms of noise can be generated, from white through pink to brown). Noise generators can be synched to all Main/Aux oscillators for variety of spectral/formant-like sounds (similar to "Synched Noise" oscillators on Clavia Nord Lead series of hardware synthesizers). Noise sync seed is freely programmable.

[*] Ring Modulator (Mixer/FM Page; RingMod #1 ... RingMod #4 Sections)

There are 4 ring modulators on-board, each of them with two inputs (both inputs accept the signal from any audio or modulation source). One input can be switched to unipolar which basically turns a ring modulation into amplitude modulation. Ring modulators are located outside the audio path so, in order to use them, mixer must be activated or they have to be routed directly to filters through side inputs.

[*] Routers (Mixer/FM Hidden Page; Router #1 ... Router #4 Sections)

Those modules are used to route signals between paths inside audio chain at various places: before mixer, before and after filters (pre/post-filter) and before amplifier. Routing uses very little CPU because signals are routed directly, without mixing.

[*] Driver (Filter/Amp Page; FilterB #1 ... FilterB #4 Sections)

The driver is a 4-way parallel waveshaper located before filters. It can be used not only to drive the input signal, but to shape and/or distort it using one of many drive curves per each individual waveshaper. Usually, there is no need to drive filter inputs over 25% (= 0.25) because various filter stages have (by default) nonlinear transistor characteristics ("Transistor" driver/feedback/shaper curves) and with higher input levels the signal starts to saturate. Of course, this is a very nice feature (and I was inspired by my Andromeda analog synthesizer while implementing this) but for cleaner signals don't push levels prior to the filter too much.

[*] Shaper (Filter/Amp Page; FilterB #1 ... FilterB #4 Sections)

The shaper is a 4-way parallel waveshaper located after the filters. It can be used not only to limit the output signal (because waveshapers are basically compressors/limiters in their own way), but to reshape, saturate and/or distort it using one of many saturation curves per each individual waveshaper. Shaper can be located either before post-filter mixer (so only filter output go through the shaper) or after post-filter mixer (so filters + pre-filter input & post-filter side input go through the shaper).

[*] Post-Filter Mixer & Amplifier (Filter/Amp Page; Amplifier #1 ... Amplifier #4 Sections)

Post-filter mixer combines output from all filters together with pre-filter input and an additional post-filter side input. The signal then goes through a simple but effective 4-way parallel compressor (aimed mainly at keeping output level under control) to the amplifier which sets the final level of each audio path, pans both lower and upper stereo paths left/right and balances them between lower and upper stereo effect groups. It is controlled by amplifier modulation source which is usually amplifier envelope (Env #4) and is also responsible for setting the output volume and DC correction of the voice output. Volume modulation source has 5 levels of filtering to configure/avoid possible artifacts when control rate is much lower than audio rate, especially with lower sample rates.

[*] Filter A Module (Filter/Amp Page; FilterA #1 ... FilterA #4 Sections)

Filter A module consists of 4 parallel State Variable Filters (SVF) of the ZDF (Zero-Delay Feedback) kind, all of them can provide a separate lowpass / highpass / bandpass outputs (and all combinations of them at the same time). Those filters are resonant, of course, and can be driven into self-oscillation. Each of them consists of two stages, 12dB and 24dB, with configurable distance/routing, separate feedback units and separate outputs (both outputs can be used simultaneously). They go up to 24kHz and although they can go down to 0Hz there really isn't much sense to go below 20-30Hz, otherwise you can experience excessive output levels and similar artifacts with some filter configurations. Many feedback curves are available to shape the feedback and they can be either symmetrical or asymmetrical in order to generate different harmonics. One of many nice SVF characteristics is that (in case of a lowpass filter) they don't attenuate frequencies below cutoff when resonance is increased - basically, they preserve bass and have a very powerful sound.

There are 5 SVF types in Tranzistow and some of them can use quite a lot of CPU. The basic filter type is similar to Waldorf Q 12dB/24dB and Alesis Andromeda 12dB, so it is my usual choice for most sounds. FilterC / FilterH have much different topology and can have much different (I would dare to say - more analog) sound compared to the basic filter. They are mostly identical with FilterC having an internal clipper while FilterH having additional soft limiter/saturator to keep feedback under control. FilterQ is a totally different topology with different resonance characteristics and it's own unique sound. FilterX is a very aggressive and screamy diode based SVF model with a really unique sound. Both FilterQ and FilterX use quite a lot of CPU to keep things under control, so two modes of operation are provided: 12dB only (FilterQ/12 and FilterX/12) and both 12/24dB (FilterQ/24 and FilterX/24). Separate 12dB/24dB configurations are provided for other filter types as well. With all those filters - the key is to combine various feedback amounts/curves with internal limiters for a broad range of filter characteristics. All filters A have a side input for direct signal routing. Side inputs don't go through mixer/balancer and aren't affected by gain settings.

LowPass/HighPass/BandPass outputs of both stages can be set independently, so various filter configurations can be achieved (for example, Yamaha CS-80 configuration with 12dB resonant highpass filter in front of a 12dB resonant lowpass, and many more). All individual A filters can be chained together - the chain is always as follows: F1=>F2=>F3=>F4=>F1 with freely configurable levels between stages, so various combinations are possible, up to the gargantuan 96dB cascaded filter.

[*] Filter B Module (Filter/Amp Page; FilterB #1 ... FilterB #4 Sections)

Filter B module consists of 4 parallel transistor based ladder filters and 2 parallel diode based ladder filters (located on paths 1 & 2 only) of the ZDF (Zero-Delay Feedback) kind, all of them resonant and driveable into self-oscillation. Transistor ladder is inspired by Oberheim Xpander filters so all poles are fully configurable and dozens of various filters types and characteristics can be easily created: lowpass, highpass, bandpass, bandstop/notch, phaseshift, 1-/2-/3-/4-pole = 6/12/18/24dB, etc, etc. Diode ladder is lowpass only with stronger resonance and a very juicy sound which reminds me on some modular filters. It is located before transistor ladder (so diode output can further be processed through transistor in series) and can be set to 6dB/12dB/18dB/24dB where 18dB/24dB modes are "well behaving" while 6dB/12dB ones are much more agresive and loud. Both transistor and diode ladders can be used at the same time and crossfaded/combined together. They go up to 24kHz and although they can go down to 0Hz there really isn't much sense to go below 20-30Hz, otherwise you can experience excessive output levels and similar artifacts with some filter configurations. Many feedback curves are available to shape the feedback and they can be either symmetrical or asymmetrical in order to generate different harmonics.

Due to the design, ladder filters attenuate frequencies below cutoff - if such behavior is not desirable for some sounds there are compensation and normalization units inside to overcome this, as well as the ability to pass only feedback signal through the feedback unit or both input signal + feedback signal. As with SVF - the key to push ladder filters to the maximum is to combine various feedback amounts/curves together with configurable poles for a broad range of filter characteristics. All filters B have a side input for direct signal routing. Side inputs don't go through mixer/balancer and aren't affected by gain settings. B filters use quite a lot of CPU as well, especially diode ladder one.

All individual B filters can be chained together - the chain is always as follows: F1=>F2=>F3=>F4=>F1 with freely configurable levels between stages, so various combinations are possible, up to the gargantuan 96dB cascaded filter.

A simple overdrive/distortion unit is available in front of all filters, so the Driver module doesn't have to be turned on if all of it's functionality is not really needed.

[*] Morphing Filter Bank (Filter/Morph = Shift + Mixer/FM Page; MorphBank #1 ... MorphBank #4)

This module contains 4 parallel filter banks, each holding 8 separate morphing filter slots. All filters have two sets of parameters (Type, Frequency, Q, Gain) and are able to morph between start/end position through modulation matrix (either the whole bank morphing and/or individual filters morphing). Each filter has 12dB slope (2-poles) and there is quite a lot of different filter types to chose from: BandPass, LowPass, HighPass, Notch, Peak, LowShelf, HighShelf and AllPass plus some simple delay and averaging filters. Although the input is directed to all active filters, they can also be chained so various parallel and serial configurations are possible. Those filters are resonant and have an internal feedback saturator but cannot be driven into self-oscillation and there is no keyboard tracking here, as well as no FM and no advanced features like in FilterA/B modules. The goal here was to use as little CPU power as possible because there are a lot of filters to process if many or all morphing slots are active, so they have to be simpler in structure.

It can be placed either in front of FilterA/B modules or right after them (before amplifier) and has a huge practical value because it can be used for various purposes: from formant/resonant filter banks containing bandpass filters to the emulation of E-MU Z-Plane filters where you can morph between totally different filters in realtime. Or it may be used just as a simple 2/4-pole resonant LowPass/HighPass/BandPass when more advanced filters are not really needed because those filterbanks use less CPU in this case, although if you use all 8 filter slots then it will draw more CPU than other filter modules, of course. In order to conserve CPU, set unused slots to Off and always activate them sequentially from the beginning (don't leave unnecessary gaps between slots).

[*] Comb Filter Module (LFO/Comb Page; Comb #1 .. Comb #4 Sections)

Comb filters are essentially a modulated feedback delay lines. They don't actually damp any part of the signal but add a delayed version of the input signal to the output, creating peaks and holes in the harmonic spectrum, depending on the frequency and feedback. That's why it is called "comb filter" because the spectrum looks like a comb after the audio goes through it. There is also a feed-forward comb filter which creates a different set of peaks and holes. Both are supported in Tranzistow and each of the four comb filters can be either type. The maximum comb delay is 1 second and the delay buffer can be shaped for more strong or a more soft sound, depending on the particular application. All comb filters have a side input for direct signal routing. Side inputs don't go through mixer/balancer and aren't affected by gain settings. Beside side inputs, comb filters have additional input/output settings, so filters A/B can be routed directly to them and/or they can be routed directly to filters A/B. Among the other things, this is great for various per-voice chorus/flanger effects etc. Comb filters also have a lot of use in physical modeling because they can create abstractions of various string and wind instruments.

Beside CPU usage, comb filters are large memory consuments because there are 4 of them per voice and there are 128 voices across all parts = 512 comb filters in total, each needing a 1-second buffer of floating point values up to 4x oversample rate (= 384kHz @ 96kHz sample rate). For this reason, comb buffers are allocated on-demand, after comb filters were activated or after a patch using comb filters has been loaded (but once allocated - they remain allocated for a particular part until you restart Tranzistow). As not all 4 comb filters are needed in most applications, if you set the delay time of comb filter 4, or filters 3 & 4, or filters 2 & 3 & 4 to zero then Tranzistow is able to optimize memory access to some extent thus reducing CPU usage, but only if the top filters are set to zero. It doesn't work if, for example, you set delay time of filters 2 & 3 to zero while leaving filter 4 at non-zero value.

All individual comb filters can be chained together - the chain is always as follows: F1=>F2=>F3=>F4=>F1 with freely configurable levels between stages, so various combinations are possible. There are 3 levels of comb buffer note-on initialization: (1) clear the buffer - all values set to zero, (2) initialize with white noise and (3) not initialize at all (the content from the previous note is preserved). All four comb filters are initialized in the same way and it is not possible, for example, to clear the buffer for one comb and initialize it with white noise for another one.

Warning: Beware of levels - comb filters can produce extremely high output level!

[*] Filter FM (Mixer/FM Page; Filter #1 FM .. Filter #4 FM Sections)

All advanced filters (A, B, Comb) can be frequency modulated by any audio or modulation source, including filters themselves. Contrary to oscillator FM, each filter has only one FM input but additional side-inputs are still available through (audio) modulation matrix in the same way as with oscillator FM. Morphing bank filters cannot be frequency modulated and, due to the simpler design, they don't respond very well to fast modulation.

[*] Voice Engine (Osc/Main & Patch/FX Pages; Osc, AuxOsc, Modules, Voice & Control Sections)

Voice engine activates/executes all modules and routes audio/modulation signals between them based on parameters set inside a patch. It combines outputs from all individual voices into a single 4-mono/2-stereo audio stream and sends it to the effect processor. It is also responsible for audio input and MIDI messages handling, including unison mode where more than one voice is used per note with configurable detuning and modulation drift. Both note low/high and velocity low/high ranges are available for all oscillators and operators.

Voice engine supports 3 levels of audio quality: Single Processing (no oversampling), Double Processing (2x oversampling) and Quad Processing (4x oversampling). Oversampling can be activated for audio/synthesis engine only, of for both audio/synthesis and FX engine. On the other hand, the engine supports audio reduction where audio quality can be degraded either by lowering the resolution of the signal (bit reduction down to just 1 bit) or by dividing the (over)sample rate by a given factor (down to 1/1024 for totally quantized and disharmonic audio - great for industrial sounds, for example). Reduction works on audio generators only (Main/Aux oscillators and QFM engine) meaning that FM, ring modulators, filters, waveshapers, amplifiers and other audio processors are not affected. In many cases (especially when working with 88.2/96kHz sample rates and/or oversampling) audio generators (over)sample rate can be reduced by a factor of 2 without any audible or measurable audio artifacts but with lower CPU consumption.

Voice engine contains the dedicated declicker unit for envelopes and filters. Clicks can occur under various circumstances: short envelope times combined with mono mode and/or free-running oscillators, voice stealing, playing speed and style, etc. Declicker can minimize those clicks in most cases. Use it carefully (otherwise it can create clicks when there aren't any) and don't forget that not all clicks are (always) bad ;-)

I addition to audio signals, voice engine generates all modulation/control signals at the rate set inside a patch (from up to audio/oversample rate down to sample rate / 4096; 4 to 8 is optimum for most applications) including MIDI modulation (approx. 333x per second), note-on modulation and the modulation of FX parameters (from sample rate down to sample rate / 4096). Although sources modulate destinations through various modulation matrices (and almost everything can modulate almost everything) there are 4 fixed parallel control signal lanes which are used for direct modulation and the modulation of rate/time parameters.

Audio and control signals are interconnected inside the engine, meaning that audio signals can act as control signals and vice versa - you can put audio signals as sources in modulation matrix, or control signals as oscillator FM sources, for example. Of course, audio signals will always be sampled at control rate if used as control signals and control signals will remain quantized at control rate if used as audio signals, even if used in audio-rate modulation slots. To overcome this there is a dedicated Slicer module which can smooth LFOs, envelopes and lag processors if they need to be used as audio signals. Slicer is turned off by default to conserve CPU power.

[*] Low Frequency Oscillators (LFO/Comb Page; LFO/AuxLFO #1 ... LFO/AuxLFO #4 Sections)

Low frequency oscillators (LFOs) are similar to audio oscillators but their purpose is to act as modulation sources, so the parameter set is completely different. Although they are designated as "low frequency" they can go well into the audio range, up to 24kHz depending on the control rate. But, keep in mind that they aren't antialiased so they can generate tons of unwanted harmonics if you decide to use them as audio sources. There are two sets with 4 LFOs in each set: main and auxiliary (8 in total). Main LFOs are always active while auxiliary ones can be activated on demand to conserve CPU, otherwise they are exactly the same. Beside using the built-in shapes, LFOs can directly pass another sound sources through themselves, just like Main/Aux oscillators but quantized at the control rate. LFOs can be synchronized to Hrastow tempo (which itself can sync to incoming MIDI clock) and all patch LFOs can be synced together as well (so all LFO modulations per chord are synchronized, great for chord sweeps). Because those "Voice LFOs" are tied together their rates cannot be individually modulated or synced to other modulation sources, although they do react to MIDI realtime messages (Start, Cont and Measure). Each LFO can also act like an envelope in "one-shot" mode meaning that it will go through one conmplete cycle only.

[*] Envelopes (Env/Main & Env/Aux Pages; Env/AuxEnv #1 ... Env/AuxEnv #4 Sections)

There are two sets with 4 envelopes in each set: main and auxiliary (8 in total). Main envelopes are always active while auxiliary ones can be activated on demand to conserve CPU, otherwise they are exactly the same. Each envelope has 10 stages (8 from note-on and 2 after note-off) with time/level parameters, configurable curves and loop points from each stage to any other stage for building very complex modulations including the simulation of complex multistage LFOs. Beside using the built-in curves, each envelope stage can directly pass another sound sources through itself, just like Main/Aux oscillators but quantized at the control rate (see "Env Sequence" patch). Envelopes can be synchronized to Hrastow tempo as well.

Env #4 is the "Amplifier Envelope" and it controls the overall duration of the voice. It can be routed to other modulation destinations just as all other envelopes but it controls the voice activity no matter what destination it is routed to. Amplifier modulation source is usually (and by default) set to Env #4 and, although it can be set to any other source, Env #4 still acts like amplifier envelope and controls the voice duration.

All envelopes have an internal sub-envelope path with a separate set of levels. This parallel second envelope is not smoothed and is not under the sequence control but, otherwise, it is identical to the main one and all other parameters (like times etc.) are shared between them. So, Tranzistow envelopes are actually dual-envelopes.

In addition to the regular editor, each Main/Aux envelope has a built-in graphical editor which can be activated by clicking on the appropriate envelope section. It enables easier editing because you can draw the envelope with the mouse and see how it actually looks in time. To work with a particular stage you can also press and hold keys 0..9 before and while using the mouse. This is especially useful if some or all stages are closed (after patch initialization, for example) because you cannot access them directly with the mouse in this case. This editor also features a configurable per-envelope vertical grid, saved with a patch. The grid is in time divisions meanig that, for example, with duration set to 10s and grid set to "/20" the envelope will be divided into 0.5s segments. Furthermore, envelope curves can be changed directly from editor (right-click with the mouse on a particular envelope segment).

[*] Lag Module (Spec/Mod Page; Lag #1 ... Lag #4 Sections)

A lag unit is like the attack stage of an envelope - it can lag the sound a bit but can also be used to implement glide/portamento. There is no dedicated portamento module in Tranzistow and this is rather uncommon portamento implementation, but it works. And, to be honest, I am not really a big glide/portamento fan and rarely use such sounds.

[*] Sample & Hold Module (Spec/Mod Alternate Hidden Page; S&H #1 ... S&H #4 Sections)

As its name would suggest, sample & hold unit samples an input signal at regular intervals and holds this value at a constant level for a specified period of time. In other words, it quantizes the input. There are four S&H units in this module, each able to sample/quantize any control or audio source. S&H output can then be used as any other Tranzistow modulation source.

[*] Modulation Matrix (Matrix/16 & Matrix/32 Pages; ModMatrix #1 ... ModMatrix #32 Sections)

All modulation routing between various sources and destinations goes through one of 5 modulation matrices. This main/fast one is processed at the control rate (from up to audio/oversample rate down to sample rate / 4096) and has 32 slots grouped into 8 groups of 4 consecutive slots. Each slot has one modulation source, two separate destinations, one modulation curve/shape and various other parameters for precise control of a particular modulation routing. To conserve CPU power only groups with at least one active slot (one or both destinations assigned) are processed. So, don't leave unnecessary gaps between slots and always allocate them sequentially from the beginning. There is also the possibility to run the first 4*N slots at full audio/oversample rate while the others are processed at the regular control rate. This allows for processing various audio rate modulations like real exponential oscillator FM etc. Furthermore, the first 4*N slots can be set to "Fast" so they process only Source, Destination and Amount parameters to save CPU power (especially useful with audio-rate modulation slots). Modulation destinations also provide a choice of side-inputs for oscillator FM, QFM operators, ring modulators, mixer, etc.

[*] MIDI Modulation Matrix (MIDI/Mod Page; MIDIMatrix #1 ... MIDIMatrix #16 Sections)

This slow modulation matrix is processed at the MIDI rate (approx. 333x per second) and has 16 slots grouped into 4 groups of 4 consecutive slots. Each slot has one modulation source, two separate destinations, one modulation curve/shape and various other parameters for precise control of a particular modulation routing. To conserve CPU power only groups with at least one active slot (one or both destinations assigned) are processed. So, don't leave unnecessary gaps between slots and always allocate them sequentially from the beginning.

[*] Note-On Modulation Matrix (Note/Mod Page; NoteMatrix #1 ... NoteMatrix #16 Sections)

This modulation matrix is processed at note-on only and has 16 slots grouped into 4 groups of 4 consecutive slots. It's main purpose is to process modulations which have sense at the beginning of the note only - like velocity, keytracking etc. Each slot has one modulation source, two separate destinations, one modulation curve/shape and various other parameters for precise control of a particular modulation routing. To conserve CPU power only groups with at least one active slot (one or both destinations assigned) are processed. So, don't leave unnecessary gaps between slots and always allocate them sequentially from the beginning.

[*] Note-Off Modulation Matrix (Spec/Mod Page; OffMatrix #1 ... OffMatrix #4 Sections)

This modulation matrix is processed after the note-off MIDI message has been received and has 4 modulation slots. Like the main/fast modulation matrix, it is processed at the control rate (from up to audio/oversample rate down to sample rate / 4096) as well. Each slot has one modulation source, two separate destinations, one modulation curve/shape and various other parameters for precise control of a particular modulation routing. To conserve CPU power this matrix is processed only if at least one slot is active (one or both destinations assigned).

[*] Direct Modulation (Spec/Mod Page; Direct Modulation Section)

Beside modulation matrices, some sources (like envelopes, LFOs, etc.) can modulate six most important parameters (Main/Aux oscillators pitch, the cutoff of filters A & B and comb filter frequency) directly over four fixed control lanes. Sources on one lane can modulate destinations on the same lane only. For example, LFO #1 can directly modulate the pitch of oscillator #1 but cannot directly modulate the pitch of oscillators #2, #3 and #4 (modulation matrix has to be used for this instead).

[*] Edit Mod Matrix (Edit/Mod = Shift + MIDI/Mod Page; EditMatrix #1 ... EditMatrix #16 Sections)

This modulation matrix is used to control thousands of editor parameters directly with MIDI controllers, NRPNs and AutoMod parameters. As the name implies, it is connected to the editor meaning that only currently selected part can be controlled because there is only one editor for all parts, of course. Each slot has one modulation source and two separate destinations with appropriate amounts and offsets. By default, processing/update rate is 10ms (except for AutoMod parameters which are updated at control rate) but this can be changed in Tranzistow.ini ([Editor] section, ModMatrix=<ms> option) if needed. And as always, don't leave unnecessary gaps between slots and always allocate them sequentially from the beginning.

[*] Modifiers (Spec/Mod Page; Modifier #1 ... Modifier #4 Sections)

Each of the 4 modifiers can perform various mathematical/logical operations on two input control signals and two constants. The results of those operations can then be used as modulation matrix sources. Furthermore, an modifier can be used as input to another modifier for building more complex modulations.

[*] Sync Modifiers (Spec/Mod Hidden Page; Modifier #1 ... Modifier #4 Sections)

There are 4 sync-modifiers per voice which run at audio/oversample rate and, beside additional LFO/envelope/etc. syncing (over the existing functionality built into the modulation engine), they can also be used to sync oscillators to various modules and even MIDI controllers. Contrary to the virtual sync oscillators and interoscillator syncing, sync-modifiers are not antialiased and can create all sorts of digital artifacts which can be desirable or undesirable, depending on the particular case. They don't sync at the end of the cycle as regular oscillators (because many sync-mod sources don't even have a cycle, only oscillators, LFOs, envelopes and lag processors do) but at the crossing from negative to positive values (MIDI controllers cross at the middle of their range). For this reason, you can have more than one sync points during the same cycle, when using wavetables, for example, which is not possible with the virtual sync oscillators and interoscillator syncing.

[*] Rate/Time Modulation (Spec/Mod Hidden Page; Modifier #1 ... Modifier #4 Sections)

Various rate/time parameters (like LFO rates, envelope times for all segments, chorus rates, delay times, etc.) can be modulated in realtime over four fixed control lanes, each with a separate ModFactor parameter: #1 for LFO #1 / AuxLFO #1 / Env #1 / AuxEnv #1 / Lag #1, #2 for LFO #2 / AuxLFO #2 / Env #2 / AuxEnv #2 / Lag #2, etc. Note: All envelope / lag / delay times are translated into rates internally (for example, 4s envelope time translates into 0.25Hz envelope rate).

The formula for rate modulation is: ModulatedRate = Rate * (1 + RateModulation * ModFactor)

Example 1: LFO #1 Rate=1Hz, ModFactor1=1000, Amount=1 => The resulting LFO #1 rate will be 1001Hz
Example 2: LFO #4 Rate=1Hz, ModFactor4=1, Amount=-0.9 => The resulting LFO #4 rate will be 0.1Hz

Example 3: Env #1 Time=0.5s (Env #1 Rate=2Hz), ModFactor1=1, Amount=-0.75 => The resulting Env #1 time will be 2s (0.5Hz rate)

[*] Contour Generators (Osc/Main Page; Contours Button/Section)

In addition to predefined waves/wavetables, LFO shapes, envelope/lag curves, filter/drive/shaper and other curves/shapes, there are 100 freely drawable contour generators (with built-in waveform/spectral graphical editor) per patch, each generator with up to 8192 points (although contours with over 1024 points don't have much practical sense). This provides a whole new level of versatility because they can be used as additional Main/Aux oscillator waveforms, LFO shapes, envelope curves, filter curves, modulation curves, etc. If used as main oscillator waveforms they can be bandlimited (only contours with 4 points and up) or non-bandlimited, with freely configurable min/max window parameters (the same as built-in waveforms). All contour generators have various resynthesis options and capabilities. Among the other things, WAV files can be loaded, resynthesized and used as the source material. Furthermore, the editor can be switched to spectral mode where harmonics can be drawn in the same way as the actual waveform. Unfortunately, I ran out of time while preparing Tranzistow for production and, except "Contour 00" example patch, couldn't create more patches exploring this extremely powerful capability. Contours editor also features a per-contour configurable vertical grid divided in samples, saved with a patch.

[*] Linear vs. Exponential Pitch Modulation

While pitch itself is exponential by definition, pitch modulation is linear by default (this goes back to early Tranzistow days), meaning 1 octave up for +1 modulation value and 0Hz for -1 modulation value (with oscillator/operator range set to zero). Exponential pitch modulation can be activated on-demand for individual (or all) oscillators and operators, so you have the best of both worlds. FM is always linear, except if you do FM through audio modulation matrix where you can use different modulation curves.

[*] FX Engine (Patch/FX Page)

Tranzistow contains a full-fledged effects engine with a dedicated chorus/flanger, phaser, delay and reverb FX modules. Except in case of chorus/flanger, all other modules contain two parallel stereo FX units which operate on lower and upper stereo audio paths, and both of them can be arranged in parallel or serial and everything in between. Phaser/Delay/Reverb units have two sets of lowpass/highpass filters (one at the input, one at the output) and can be connected in arbitrary order. FX engine can work at the sample rate or at 2x or 4x oversampling rate. The order of modules inside the FX chain is fully configurable.

[*] Chorus (Chorus/FX = Shift + Spec/Mod Page; Chorus #1 ... Chorus #4 [A/B])

Stereo chorus/flanger FX unit contains 4 delay lines and 8 LFOs per channel (A/B) - especially great for strings and pads but perfectly usable for many other sounds out there. By default, it is located at the end of the voice chain, on lower stereo path only and before Phaser/Delay/Reverb effect units, but can be positioned anywhere inside the FX chain if desired. The times for all delay lines and rates for all LFOs are freely configurable, together with LFO shapes and many other parameters. Delays 1 & 2 on A+B channels as well as left and right delays 3 & 4 on A+B channels can be arranged into a so-called "matrix" configuration for more smooth and spacious sound.

[*] Phaser (Patch/FX Page; Phaser #1 & #2 Sections)

The phaser is a combination of several all-pass filters working in parallel. This generates a strongly colorized signal with a "spacey" and "swooshy" character, depending on center, spacing and feedback parameters. Up to 16 stages can be activated for both lower and upper units. Each phaser unit contains an dedicated LFO with configurable shape, rate and phase (both rate and phase separate for left and right channels). Phaser LFOs are syncable to Hrastow tempo and can also modulate the times of delays #1 and #2 enabling them to act as additional chorus/flanger units if needed. Those LFOs can even modulate reverb times for a very special reverb sound. Setting wet amount for both phasers #1 & #2 to zero will turn the whole phaser module off to conserve CPU.

[*] Delay (Patch/FX Page; Delay #1 & #2 Sections)

The delay is an effect which produces echoes of the input signal and is the most usable effect of all effects (at least to me). Those echoes can be routed back and added to the input through an feedback unit for much denser delays. Both parallel stereo delay units can be individually synced to Hrastow tempo, crossed between and arranged into a so-called "matrix" configuration for more smooth and spacious sound (a "poor man's" reverb without using the dedicated reverb unit). Setting wet amount for both delays #1 & #2 to zero will turn the whole delay module off to conserve CPU.

[*] Reverb (Patch/FX Page; Reverb #1 & #2 Sections; Reverb/FX = Shift + PatchFX Page)

The reverb effect adds ambience to a dry audio signal making it more spacious and 3-dimensional. Both parallel stereo reverb units contain and internal early reflection sub-unit with up to 16 taps, internal feedback-delay sub-unit with up to 16 taps and internal all-pass filter sub-unit with up to 16 taps. There are tons of various parameters to configure all of them for any particular application. Groups of 4 successive feedback-delay sub-units can be arranged into a so-called "matrix" configuration for more smooth and spacious sound.
There is also an additional auxiliary stereo reverb embedded into reverb #1. Setting wet amount for both reverbs #1 & #2 to zero will turn the whole reverb module off to conserve CPU.

Ed Ten Eyck (www.edtaudio.com) on the Tranzistow reverb:

"The Reverb sounds very smooth and spacious to me. At first the Endelverb preset reminded me of the Eventide Blackhole reverb because of it's smoothness. It would be excellent on it's own as an effect plugin!"

[*] FX Modulation Matrix (Spec/Mod Page; FXMatrix #1 ... FXMatrix #4 Sections)

This modulation matrix is processed at FX control rate (from sample rate down to sample rate / 4096) and has 4 modulation slots for modulating various FX parameters. Each slot has one modulation source, two separate destinations, one modulation curve/shape and various other parameters for precise control of a particular modulation routing. Only MIDI sources have sense and are available here. To conserve CPU power this matrix is processed only if at least one slot is active (one or both destinations assigned).

[*] MIDI Implementation

Tranzistow has a full MIDI implementation and all MIDI control messages (including control change messages, channel aftertouch, poly aftertouch and pitch bend messages) as well as note-on and note-off (release) velocity can be used as sources in all modulation matrices. There is no awkward "MIDI learning" and similar idiosyncrasies - MIDI works in the same way as used to work on hardware synthesizers for ages.

[*] Multitimbral Operation & Multithreading (Hrastow only)

Tranzistow is always operating in multitimbral mode with 8 independent multimode parts (4 main+sub pairs) where each part occupies a separate CPU core (utilizing 4 CPU cores in total). Parts 5 to 8 can be accessed by clicking on Part #1 ... Part #4 buttons while holding Ctrl key. There are two multithreading modes (1 and 0) which can be switched in Tranzistow.ini (this file should be located in the Current User directory or, if not found there, in the Hrastow directory:

[Engine]
Multithreading= 1 | 2 | 0 ; Default: 1 for x32/x64 | 2 for X2
...

(*) Multithreading=1 ... Each main+sub part pair runs inside a separate thread and thread for the first part pair is used for mixing (8 parts total = 4 main + 4 sub). This is the default configuration, optimal for 4-core CPUs.

(*) Multithreading=2 | 0 ... Each main+sub part pair runs inside a separate thread + there is an additional thread for mixing all parts (8 parts total = 4 main + 4 sub). This is the preferred configuration if you have an CPU with 6, 8 or more cores.

Each part in a multimode setup has additional parameters (Note Low/High, Velocity Low/High, Transpose, Pan and Level) which are saved with the project and are separated from the patch itself. There is a dedicated editor for those parameters accessible by clicking on "Control" button on "Patch/FX" page. To conserve DSP resources, the sample rate for each part can be divided by 2, 3 or 4 when full processing rate and the best sound quality is not really needed. So, for example, if working at 96kHz some parts can be left at 96kHz, some can be processed at 48kHz and some at 36kHz or even 24kHz. Tranzistow has an built-in timing module to measure CPU usage for each of the 4 threads/cores, very helpful when avoiding CPU overload.

Private versions only: Main parts can borrow voices from sub parts, so it is possible to have up to 32 voices per each main part. For example, if you set the polyphony for part 5 to 0 then you can set the polyphony for part 1 up to 32. Or set it to 8 and have up to 24 voices on part 1 etc. The same for all main+sub part pairs (1+5, 2+6, 3+7, 4+8), as well as for additional parts in v16/v16+/x16/x16+, v24/v24+/x24/x24+ and v32/v32+/x32/x32+ versions.

And last but not least, Tranzistow engine allows individual MIDI paths inside a single patch. The basic concept behind this is: Tranzistow voices are very complex and you don't really need all modules for a single voice in most cases. So, it would be very nice if some modules from the same voice could respond on different MIDI channels. As described before, there are 4 mono paths per voice and each mono path can be assigned to a separate MIDI channel (assignable relative to the part MIDI channel) giving you the possibility to have different sounds from the same patch. This enhancement won't extend the available polyphony because paths assigned to different MIDI channels aren't disabled while playing (internal engine paralelism doesn't allow this), they are still executing but are muted after post-filter mixer. All this opens a whole new world of interactions between various paths/modules, something not possible with any other synthesizer out there. The same as with most other Tranzistow features, for that matter ;-)

[*] Arpeggiator (Alt+Patch/FX Page)

Each multimode part has it's own independent arpeggiator. It is always synced to Hrastow tempo, responds to MIDI Start/Stop/Continue realtime messages and has the usual set of features: Mode, Clock, Gate, Range, Notes, Pattern (16 fixed rhythm patterns + 1 user-definable), Reset and Sort.

[*] What is missing from this manual?

Well, quite a lot ... This is a partial list of Tranzistow features which are implemented but haven't been described or even mentioned yet:

(*) [HyperWave] main oscillator mode
(*) Wave additive generators + harmonic/graphical editor and resynthesis capabilities
(*) F-Operators and appropriate modes for QFM engine
(*) SY77/TG77 operator wavetables/waveforms for QFM operators, oscillators and all sound/modulations sources
(*) Additive generators
(*) Step sequencer and everything about that
(*) Chord memory
(*) Microtuning (plus editor), BaseNote and BasePitch
(*) AuxEnv "Release-Continue" mode
(*) TrueMono declicker/voice mode
(*) Response curves for MW, CAT, PAT and Velocity
(*) Various voice-related modulation sources + StereoSpread modulation source
(*) Phaser and Chorus LFOs as modulation sources
(*) Extended modulation sources (faders etc.)
(*) Vector/Mix and Vector+Mix modes
(*) Averagers, AuxAveragers and FMDelay2 feature
(*) TransMod Matrix
(*) Voice Mixer and Amp Overdrive/Distortion
(*) Patch Overdrive/Distortion and Limiter
(*) Vocoder
(*) Patch Wavetables sample import/synthetize
(*) Various contours features: copy/paste, import/synthetize, normalize, multiple contours import/normalize etc.
(*) Custom names for user waves
(*) Custom editor page where user can arrange his own mini-synth controls
(*) ... and I am pretty much sure I still forgot many of them.

[*] User Interface

Tranzistow user interface is, in general, designed and optimized for touch-screens, and works great with touch-screens, so it has just a few basic elements: sections, buttons, lists and sliders. To access various parameters through lists (LFO shapes, modulation sources/destinations, etc.) just click on the parameter and a list will pop up. All sliders have an edit box above them where you can enter the value directly. Double-click on the edit box will show the calculator (result is returned back to the edit box). All sliders also have a vertical mode where you can move a mouse vertically outside the slider area and use the whole display height for a single slider. Right-click on the parameter will reset it to a default value. Double-click on the parameter name (or right-click on the parameter slider/list while holding Ctrl key) is "undo" - it will reset a particular parameter to the previous value (undo buffers are unlimited). Press and hold Ctrl key while moving a slider to activate the "microscope" mode for precise parameter adjustment. Sliders and lists support mouse wheel, so you can hover over them to increase/decrease the value.

Many parameters have extended ranges e.g. they allow values outside the regular slider range. If a value falls outside the slider range then a slider will be colored red. You can enter such values directly into the edit box or you can press and hold Alt key and move a mouse outside the slider area. In case of "clocked" parameters (LFO rates and phaser/delay times with Clocked parameter activated) numbers represent notes e.g. 0.0625 = 1/16 note, 0.125 = 1/8 note, 0.25 = 1/4 note, 0.5 = 1/2 note, 1 = whole note, etc. To simplify entering of such parameters you can enter a number and press / (divide) key. For example, 4 followed by / will give you 0.25.

Tranzistow has a huge number of parameters (almost 6000 per patch) and there simply isn't enough space on main pages for all of them. That's why there are hidden parameters on hidden pages (accessible by double-clicking on the page button) and alternate hidden pages (accessible by clicking on the page button while holding Alt key). Furthermore, a few shifted pages exists as mentioned in QFM, Chorus and Reverb sections.

In addition to the on-screen GUI elements, a few keyboard shortcuts are available as well: F2 = Save, F3 = Load and F9 = Init. Those shortcuts must be enabled in Tranzistow.ini:

```
[Editor]
Shortcuts=1
...
```

User interface us fully scalable so it can be accommodated to large displays which are more and more common today. Scaling can be enabled in Tranzistow.ini:

```
[Editor]
Scale=150
...
```

Scale parameter is in percents and the default is 100 (meaning the original 100% scale). The user interface is optimized for 1280x720, 1280x768 and 1280x800 resolutions so the alignment of various elements can be slightly different with scales other than 100. The lowest possible scale is 10. If you set Scale=0 then user interface will be scaled automatically according to the current screen resolution.

[*] Tranzistow.ini

As mentioned before, Tranzistow.ini is a configuration file used to store some options which aren't available through the user interface. It's just a text file which should be created in the folder where Tranzistow DLLs are located or in the "Current User" folder (C:\Users\<YourWindowsUserName>\AppData\Local\Tranzistow). If located in the "Current User" folder then it takes precedence and the one in the folder with DLLs is ignored. To create this file go to the folder where you want it, right-click, select New => Text Document, name it Tranzistow.ini and then you can edit it with Notepad or some other text editor.

[*] Patches & Presets

Each Tranzistow patch is a separate file which can be loaded from disk, saved to disk, organized into directories and manipulated like any other file. Patch file locations, MIDI channels and the polyphony for all parts are saved by Hrastow as part of the project file. For easier browsing through patches there is a separate browser page accessible by clicking on the "Browse" button located on the "Patch/FX" page. On the same page there are 3 more patch buttons: "Init" to initialize the patch to default settings, "Load" to load the patch from the file and "Save" to save the patch to the file. Holding Alt key while clicking on the "Save" button will act as "Save As" e.g. the save dialog will pop up, so you can set the file name and directory where you want the patch to be saved.
Patch browser has the ability to show directories for easier navigation. The position of directories in the list is controlled by the following option in Tranzistow.ini:

[Editor]
BrowseDirs=0 | 1 | 2; Default: 1

0 = No directories in browser
1 = Directories at the beginning (before patches)
2 = Directories at the end (after patches)

Furthermore, patches can be categorized and filtered by the category in both patch selector and browser. Just right-click on the patch selector or inside patch browser to set the category and only patches with this category will be shown. Please, note that none of preset patches have a category set because this feature has been introduced quite late during Tranzistow development.

[*] Randomizer

Tranzistow has a full-blown randomizer with a parallel user interface (activated with Alt+Init) where you can set min/max ranges for all parameters you want to randomize. It mainly generates various industrial drones/noises and that's what I like about it and what I am using it for ;-)

The patch can be randomized from randomizer editor or directly from the regular editor using Ctrl+Init combination (or Ctrl+F9 if shortcuts are enabled). There are two types of randomization: absolute and relative (parameters are randomized relative to the current values). The type and intensity is configurable through "Randomizator" parameter: positive = absolute, negative = relative. All randomization parameters are saved together with the patch so you can recall all settings later. If you want to load a patch without loading randomization parameters then press and hold Alt key while loading. Randomization parameters from the current patch will be preserved in this case = easy way to copy randomization parameters from one patch to another.

[*] External Control Interface

External Control Interface (ECI) is the module which enables the direct communication between Tranzistow user interface and external/hardware MIDI controllers. Tranzistow has a built-in native support for Behringer BCR2000 which makes creating/editing patches a breeze. Everything you can do using the standard Tranzistow screen editor can be done through ECI and BCR2000. The only thing you have to do is to transmit the appropriate system exclusive file I created for Behringer BCR2000 (this must be done only once). Various other nice features have been implemented as well, like knob anti-jittering, curves and 4 levels of knob precision for fine granularity of parameters etc. ECI uses Edit/Mod matrix module so the update rate is the same, providing very smooth and fluent experience.

External Control Interface must be enabled in Tranzistow.ini:

```
[Editor]
ExternalControl=1
...
```

ECI is working on MIDI channel 16 meaning that all ECI Control Change messages must be sent on channel 16. It is tied strongly to Behringer BCR2000 controller but any MIDI controller with similar capabilities can be reprogrammed to use it. My BCR2000 ECI configuration can be downloaded here:

https://www.hrastprogrammer.com/hrastwerk/download/TranzistowECI.syx

In the following text ShiftKey means "Shift key on the computer keyboard", CtrlKey means "Ctrl key on the computer keyboard", AltKey means "Alt key on the computer keyboard" and "|" means "or". You don't have to reprogram all those knobs/buttons on your MIDI controller. Basically, in order to achieve a minimum of "controlability" you have to program 8 knobs to send 8 relative values for MIDI CCs 0..7 and you'll be able to control first 8 column parameters on each Tranzistow page.

The following controllers are defined:

(*) Knobs 1..32 (MIDI CC: 0..31, Value: $00..$3F = 0..63, $40..$7F = -64..-1)

Those must be endless pots/encoders which can send *relative* values e.g. a difference between the current value and previous value (MIDI CC values must be in two's complement: 0..63 = 0..63, 64..127 = -64..-1). Knobs are arranged in 4 rows with 8 knobs in each row forming an matrix with 8 columns of 4 successive parameters. Those column are tied directly to parameter columns (https://www.hrastprogrammer.com/hrastwerk/tranzistow/TranzistowExt.png). There is knob anti-jitter built in and ECI is ignoring the first knob movement after a threshold time (around 500ms).

(*) KnobButtons 1..8 (MIDI CC 32..39, Value: $00 = Off, $7F = On)

Those are buttons used for scrolling parameter columns up/down. On BCR2000 the top 8 knobs are push-knobs so you can push a knob and rotate it to scroll between parameters on the particular column. As it is sometimes rather unconvenient to rotate a knob while it is pushed down, you can push a knob and rotate another knob => parameters are always scrolled on the column for which a knob is rotated and it doesn't matter which of 4 knobs in the particular column is used. Furthermore, you can hold ShiftKey and rotate a knob for the same effect.

If you push and release the button without rotating any knob then a coresponding column will be scrolled down by one parameter (or scrolled up if ShiftKnob or ShiftKey was hold down).

(*) Buttons 1..16 (MIDI CC 40..55, Value: $00 = Off, $7F = On)

Buttons 1..13 are used to switch between Tranzistow pages while the remaining 3 buttons have the following functions:

Button 14 ... Patch Load
Button 15 ... Patch Save
Button 16 ... Patch Init (but only if ShiftButton and DefaultButton pressed to protect from inadvertent initialization)

Furthermore, the following functions are also available:

* DefaultButton | AltKey + Button 14 ... Load patch without randomization parameters (if Randomizer has been activated in Tranzistow.ini)

* DefaultButton | AltKey + Button 15 ... Save As

* DefaultButton | AltKey+ Button 16 ... Toggle between Randomizer/Synthesizer editors (if Randomizer has been activated in Tranzistow.ini)

* ShiftButton | CenterButton | CtrlKey | ShiftKey+ Button 16 ... Randomize patch (if Randomizer has been activated in Tranzistow.ini)

* ShiftButton | ShiftKey + Buttons 1..4 ... Load sample into oscillators 1..4

* UndoButton + Buttons 1..4 ... Remove sample from oscillators 1..4

On BCR2000 those are 16 buttons below push-knobs.

(*) PartButtons (MIDI CC 56..59, Value: $00 = Off, $7F = On)

Used to select parts #1..#4.

Furthermore, the following functions are also available:

* ButtonShift | ShiftKey + PartButtons ... 4 levels of knob movement smoothing

On BCR2000 those are 4 "ENCODER GROUPS" buttons.

(*) DefaultButton (MIDI CC 60, Value: $00 = Off, $7F = On)

Used to set default value for a particular parameter (hold down the button and rotate the parameter knob).

On BCR2000 this is "STORE" button.

(*) CenterButton (MIDI CC 61, Value: $00 = Off, $7F = On)

Used to set center value for a particular parameter (hold down the button and rotate the parameter knob).

On BCR2000 this is "LEARN" button.

(*) UndoButton (MIDI CC 62, Value: $00 = Off, $7F = On)

Used to undo the value for a particular parameter (hold down the button and rotate the parameter knob).

On BCR2000 this is "EDIT" button.

(*) ShiftButton (MIDI CC 63, Value: $00 = Off, $7F = On)

Used for various shifted operations (hold down the button and perform the operation).

On BCR2000 this is "EXIT" button.

(*) AuxButtons (MIDI CC 64..67, Value: $00 = Off, $7F = On)

Used for various functions:

* AuxButton 1..4 ... 4 levels of knob movement precision (1, 0.1, 0.01, 0.001)

* ShiftButton | ShiftKey + AuxButton 1..4 ... 4 levels of knob movement multiplier (1x, 2x, 3x, 4x)

* UndoButton | CtrlKey + AuxButton 1..4 ... Knob movement in decimals (1, 0.1, 0.01, 0.001)

On BCR2000 those are 4 additional buttons in the right-bottom corner.

(*) Additional knob/button functions:

* CenterButton | Alt + Knob connected to list parameters ... Show parameter values list

* ShiftButton | KeyShift + CenterButton ... Toggle extended parameter range on/off

* ShiftButton | KeyShift + UndoButton ... Toggle between KnobMode (the regular knob/button usage as described above) and EditMode (knobs/buttons are used to enter values directly into parameter edit boxes)

(*) EditMode functions:

* Knobs 1..32 ... Select a parameter for editing
* Buttons 1..10 ... Digits 1,2,3,4,5,6,7,8,9,0
* Button 11 ... Decimal point
* ShiftButton | ShiftKey + Button 11 ... Hexadecimal input
* Button 12 ... Negative sign
* ShiftButton | ShiftKey + Button 12 ... Divide
* Button 13 ... Back key
* ShiftButton | ShiftKey + Button 13 ... Delete key
* Button 14 ... Left key
* ShiftButton | ShiftKey + Button 14 ... Home key
* Button 15 ... Right key
* ShiftButton | ShiftKey + Button 15 ... End key
* Button 16 ... Enter/Return key and leave EditMode
* ShiftButton | ShiftKey + Button 16 ... Enter/Return key without leaving EditMode
* PartButton 1 ... Up key
* ShiftButton | ShiftKey + PartButton 1 ... PageUp key
* PartButton 2 ... Tab key
* ShiftButton | ShiftKey + PartButton 2 ... Shift+Tab
* PartButton 3 ... Down key
* ShiftButton | ShiftKey + PartButton 3 ... PageDown key
* PartButton 4 ... Enter/Return key and leave EditMode
* ShiftButton | ShiftKey + PartButton 4 ... Enter/Return key without leaving EditMode
* AuxButton 1 ... Up key
* ShiftButton | ShiftKey + AuxButton 1 ... PageUp key
* AuxButton 2 ... Tab key
* ShiftButton | ShiftKey + AuxButton 2 ... Shift+Tab
* AuxButton 3 ... Down key
* ShiftButton | ShiftKey + AuxButton 3 ... PageDown key
* AuxButton 4 ... Enter/Return key and leave EditMode
* ShiftButton | ShiftKey + AuxButton 4 ... Enter/Return key without leaving EditMode
* ShiftButton | KeyShift + UndoButton ... Leave EditMode

(*) Knob/Scrolling jitter sensitivity can be configured in Tranzistow.ini (values are in milliseconds):

```
[Editor]
...
ExternalControl=1
KnobSensitivity=500
ScrollSensitivity=10
```

[*] Hrastow

Hrastow is my own 32/64-bit Windows application environment developed exclusively for Tranzistow. It is an actual interface between Tranzistow and the outer world (visual, file, keyboard, mouse, audio, MIDI). Hrastow uses project files to save the complete environment for each track you are working on (or for other purposes, of course). The last project is automatically loaded on start. You can also drag&drop the project file onto Hrastow executable to start working with the project or create a shortcut and put the project path into the command line.

The following operations are available:

Load ... Load an existing Hrastow project file. Hold Ctrl while clicking on the button to load a new empty project.

Save ... Save a project file. If you hold Alt while clicking on the button then "Save As" dialog will appear, the same with right-click.

Slot ... Activate an FX slot. The GUI for the plugin in slot will be displayed (if loaded but hidden) and will come in focus. Right-click on the button will show a slot mixer for the project.

VST ... Load a plugin in the active slot. Hold Ctrl while clicking on the button to remove a plugin from the slot (if loaded).

ASIO ... Opens a menu where you can choose ASIO driver, configure audio inputs/outputs, open driver control panel, set configuration and activate oversampling.

MIDI ... Opens a menu where you can choose MIDI inputs. Both inputs are mixed so you can, for example, control the synthesizer from the sequencer and attached keyboard at the same time. Right-click on the button will open an arpeggiator window where you can program 8 independent arpeggiators on 8 MIDI channels. This is from the time when Tranzistow didn't have an arpeggiator but can also be used with other synthesizers if loaded into FX slots.

Tempo ... Sets the tempo for the project. Enter zero to sync Hrastow to incoming MIDI clock.

Plugin states (together with positions and visibility status of all GUI windows), tempo, slot mixer and arpeggiator parameters are always saved with the project. Other parameters are by default stored in the registry in one of the 9 configurations (default, 1..8) but can be saved with the project as well, if so desired. There are some other parameters configurable through Hrastow.ini but I don't think they are important for regular users. I will reveal them if needed.

Projects can be interchanged between versions meaning that 64-bit Hrastow will load 64-bit Tranzistow and 32-bit Hrastow will load 32-bit Tranzistow no matter if a project has been created with 32- or 64-bit version. Other plugins cannot be interchanged, of course, so you must have a separate 32/64-bit projects if you want to use other plugins.

Keep in mind that all FX slots are running inside the main thread and are sharing one CPU core which is also shared with the first Tranzistow part by default. If you have a CPU with more than 4 cores then it is recommended to set Multithreading=-1 in the INI file so Tranzistow can use 4 cores leaving a separate core for the FX slots.

Although internal Hrastow host is VST 2.x compatible, I don't guarantee that all of your plugins would load. But in case of a free plugin (so I can download it freely) you can send me a mail and I will check it with Hrastow.

[*] Hardware

A few notes about PC configuration:

(*) Intel SpeedStep (EIST) should be disabled in BIOS.

(*) Windows power plan should be set to "Max. performance" with minimum and maximum processor state set to 100%.

(*) It is recommended to disable hyperthreading in BIOS.

[*] CPU Usage Tips

(*) Turn off modules you don't use.

(*) Use auxiliary oscillators instead of main oscillators if only saw/pulse/triangle waves are needed.

(*) Use sine oscillators if only a sine is needed. Four sine oscillators (or 8 if both Main/Aux oscillator modules are set to Sine) are a rudimentary additive engine which use very little CPU.

(*) Use FM because you can get very complex sounds with moderate CPU usage.

(*) Use main oscillators without crossfading and/or interpolation between individual waves where possible.

(*) Use "-PW" oscillators when PW is not needed.

(*) Once you turn them on - use all oscillators, filters, etc. All 4 units inside the same module are working in parallel, so you won't gain any advantage in using just one, two or three of them.

(*) Use SuperSaw oscillators instead of unison and use all 4 SuperSaw oscillators per module. It is much more effective to have 4 SuperSaw oscillators with 8 sub-oscillators each than 2 SuperSaw oscillators with 16 sub-oscillators and other two oscillators unused.

(*) Remove redundant modulations and modulations with source/destination set to "None" or amount set to zero.

(*) Don't leave unnecessary gaps between modulation slots and always allocate them sequentially from the beginning.

(*) Don't put slow modulation sources in fast modulation matrix. MIDI controllers should be put into a MIDI modulation matrix while velocity, keytrack and similar sources should be put into a note-on modulation matrix.

(*) Turn off oversampling because in most cases (especially when working with 88.2/96kHz sample rates) oversampling is not really needed.

(*) Use reductor because in many cases (especially when working with 88.2/96kHz sample rates and/or oversampling) audio generators (over)sample rate can be reduced by a factor of 2 without any audible or measurable audio artifacts but with lower CPU consumption.

(*) Use delay instead of reverb if full ambience is not really needed.

[*] IniTranzistow.hp

This file is a hidden feature where you can set initialization values for almost all patch parameters. It should be created in the directory next to your Tranzistow DLLs. Those values will then be used as defaults when initializing, loading and saving patches meaning that a particular parameter won't be saved into a patch file if it has the default value. Be very careful with this because if you, for example, set FilterCutoff to 1000 it will be the default cutoff for a particular filter. If you save a patch with this value and somebody else loads it into Tranzistow he will have a value of 0 because this is the default value on his system and the patch will sound completely different.

[*] VST Automation (only when used inside a VST host)

Although I removed VST automation during the development, I returned it back in the final version. This is a byproduct of Edit/Mod matrix because I now developed a completely new module for updating editor parameters. By default, automation is off (meaning only AutoMod parameters are active) and can be turned on in Tranzistow.ini ([Editor] section, Automation=1 option). Automation uses EditMod matrix module so the update rate is the same, except for AutoMod parameters which are updated at control rate. There are 128 of them in total on Tranzistow, grouped into 8 slots (AutoMod1..AutoMod8) with 16 parameters in each slot. On the host side you automate those parameters as usual but they don't change anything on the plugin side directly. Instead, they can be used in any modulation matrix just like all other modulation sources. AutoMod processing is also very fast, with no overhead as it is with direct parameter automation where GUI has to be updated etc.

[*] Memory (64-bit version only)

For various reasons (mainly to improve performance, make code smaller and to lower memory usage because I am using 4-byte offsets instead of 8-byte pointers), 64-bit version has it's own internal 4GB address space (sample memory excluded). It is divided into two 2GB parts ("negative" and "positive"). "Negative" part is used for reverb and comb filters (the largest memory consuments) and "positive" for everything else. When there is no free "negative" memory anymore, Tranzistow starts to use "positive" memory for reverb and comb filters as well. Memory for samples is allocated separately for each sample when you load it.

Tranzistow allocates 2GB of memory when running under Hrastow, but this can be changed in Tranzistow.ini file:

[Engine]
Memory=... ; Memory size in MB (default: 2048)

If you have enough RAM then set Memory option to 4096 and allocate full 4GB. Each Tranzistow instance allocates its own address space, so if you are used to load a separate instance for each track then you can get into trouble. Lower Memory option or (much better) use Tranzistow multitimbral capabilities instead and process one channel per part - up to 8 parts per instance (or more with Hexengine versions), not one channel per instance.

Note #1: As mentioned before, sample memory isn't included in this internal address space, so you can use all remaining RAM for samples.

Note #2: Hexengine versions support up to 16GB internal address space (8GB "positive" memory and 8GB "negative").

Note #3: When running inside DAW or other hosts then 1GB of memory is allocated per instance. This can be changed in Tranzistow.ini file as well:

[Engine]
DAWMemory=... ; Memory size in MB (default: 1024)

[*] Hexengine (private/internal versions only)

Hexengine is the latest Tranzistow engine and the foundation of X2 version. It has the following capabilities:

(*) Completely new AVX2 engine with average 40-60% performance increase over 64-bit SSE3 version.
(*) 8 / 16 / 24 / 32 / 48 parts with 16 / 32 / 48 MIDI channels and up to 128 / 256 / 384 / 512 / 768 voices in total.
(*) Full utilization of 4, 6 and 8 CPU cores (4, 6 and 8 processing threads).
(*) Sidechaining where the amplifier of one part can control the overall volume of some other part(s).
(*) Part sample rate reduction (1/2, 1/3 and 1/4) to decrease CPU usage if/where needed.
(*) Every odd part can borrow voices from the next even part on the same core (for up to 32 voices per odd part).
(*) Up to 16GB internal address space possible in 64-bit versions (8GB "positive" memory and 8GB "negative").
(*) Ability to link/chain main and sub parts together on a single core (parts can be processed through another parts).
(*) 8 external inputs and the ability to mix inputs (like hardware synthesizers) through sidechainer of each part.
(*) Freely routable core/thread outputs - each core/thread can be routed to any stereo output.
(*) Part overflow so voices stealing can be spread accross several parts on the same channel for more polyphony.
(*) 3-band EQ (High, Mid, Low) per-part or per-patch + 3-band master EQ, master volume and limiter.
(*) Per-patch digitizer module for all sorts of robotic and vocoder-like sounds.
(*) Special Performance mode (INI settable) for fast Tranzistow loading and part-mute capability.
(*) My KalquLab sequencer integrated but undocummented because it's so cryptic than no one except me could use it.
(*) System exclusive MIDI messages implementation to control Tranzistow parameters from the sequencer (deliberately set to work with Hrastow only).

This is Core/Part allocation for all Hexengine modes (2, 3, 4 and 5 are available with Hrastow only, not from DAW):

0 = 8 parts / 4 cores (v8 / x8, default):

```
Core #1: Main Part TW-1 + Sub Part TW-5
Core #2: Main Part TW-2 + Sub Part TW-6
Core #3: Main Part TW-3 + Sub Part TW-7
Core #4: Main Part TW-4 + Sub Part TW-8
```

1 = 16 parts / 4 cores (v16 / x16):

```
Core #1: Main Part TW-1 + Sub Parts TW-5, TW-9, TW-D
Core #2: Main Part TW-2 + Sub Parts TW-6, TW-A, TW-E
Core #3: Main Part TW-3 + Sub Parts TW-7, TW-B, TW-F
Core #4: Main Part TW-4 + Sub Parts TW-8, TW-C, TW-G
```

2 = 24 parts / 4 cores (v24 / x24):

```
Core #1: Main Part TW-1 + Sub Parts TW-5, TW-9, TW-D, XW-1, XW-5
Core #2: Main Part TW-2 + Sub Parts TW-6, TW-A, TW-E, XW-2, XW-6
Core #3: Main Part TW-3 + Sub Parts TW-7, TW-B, TW-F, XW-3, XW-7
Core #4: Main Part TW-4 + Sub Parts TW-8, TW-C, TW-G, XW-4, XW-8
```

3 = 24 parts / 6 cores (v24 / x24):

```
Core #1: Main Part TW-1 + Sub Parts TW-5, TW-9, TW-D
Core #2: Main Part TW-2 + Sub Parts TW-6, TW-A, TW-E
Core #3: Main Part TW-3 + Sub Parts TW-7, TW-B, TW-F
Core #4: Main Part TW-4 + Sub Parts TW-8, TW-C, TW-G
Core #5: Main Part XW-1 + Sub Parts XW-5, XW-2, XW-6
Core #6: Main Part XW-3 + Sub Parts XW-7, XW-4, XW-8
```

4 = 32 parts / 8 cores (v32 / x32):

```
Core #1: Main Part TW-1 + Sub Parts TW-5, TW-9, TW-D
Core #2: Main Part TW-2 + Sub Parts TW-6, TW-A, TW-E
Core #3: Main Part TW-3 + Sub Parts TW-7, TW-B, TW-F
Core #4: Main Part TW-4 + Sub Parts TW-8, TW-C, TW-G
Core #5: Main Part XW-1 + Sub Parts XW-5, XW-9, XW-D
Core #6: Main Part XW-2 + Sub Parts XW-6, XW-A, XW-E
Core #7: Main Part XW-3 + Sub Parts XW-7, XW-B, XW-F
Core #8: Main Part XW-4 + Sub Parts XW-8, XW-C, XW-G
```

5 = 48 parts / 8 cores (x48, AVX2 only):

This is a special mode available in AVX2 version only which I developed for 8-core machines. My main Tranzistow machine for many years was Intel i7-4790K with 8GB RAM and all 4 cores fixed to 4.4GHz, running 64-bit Windows 7 and AVX2 version of Tranzistow in Hexengine=2 mode (24 parts). An 32-bit Windows XP machine with Intel i7-2700K, 4GB RAM and all 4 cores fixed to 4.0GHz has been used to run my KalquLab sequencer. Many of my tracks need more than 24 parts, so I also employ this 32-bit machine to run 32-bit Tranzistow in Hexengine=1 mode (16 parts). As already mentioned, I recently assembled a new 8-core Intel i7-11700K machine with 32GB RAM and the clock going up to 5.0GHz, making it possible to double the available Tranzistow power, and quite a bit more, so a new Hexengine mode and x48 version has been developed to utilize it. AVX2 version also has sample-accurate step-sequencer and arpeggiator on all cores (compared to main thread/core only with Multithreading=1 mode in regular 32/64-bit versions).

Basically, it's two Hexengine=2 (x24) synthesizers assembled together in the following configuration:

```
Core #1: Main Part TW-1 + Sub Parts TW-5, TW-9, TW-D, XW-1, XW-5
Core #2: Main Part TW-2 + Sub Parts TW-6, TW-A, TW-E, XW-2, XW-6
Core #3: Main Part TW-3 + Sub Parts TW-7, TW-B, TW-F, XW-3, XW-7
Core #4: Main Part TW-4 + Sub Parts TW-8, TW-C, TW-G, XW-4, XW-8
Core #5: Main Part HW-1 + Sub Parts HW-5, HW-9, HW-D, XW-9, XW-D
Core #6: Main Part HW-2 + Sub Parts HW-6, HW-A, HW-E, XW-A, XW-E
Core #7: Main Part HW-3 + Sub Parts HW-7, HW-B, HW-F, XW-B, XW-F
Core #8: Main Part HW-4 + Sub Parts HW-8, HW-C, HW-G, XW-C, XW-G
```

This mode also supports 48 MIDI channels so Hrastow has been enhanced as well to support 3 MIDI input ports. Also Hrastow now makes it easier to concat two projects into one by employing an Extender slot in the project file.

A real hyper-monster with 48 parts and 768 ultra-complex Tranzistow voices at your disposal :-)

Note that all Hexengine features, except AVX2 (and 16GB for 32-bit version), have been downported to the regular 32/64-bit versions too. Hexengine can be activated by using Hexengine option in [Engine] section of Tranzistow.ini configuration file.

Also, X2 version has the ability to load different engines per-core, in realtime. Currently, only Diodow AVX2 engine (from DiodowX2 which has never been released) is implemented meaning that you can use one or more cores for Diodow-only parts which use less CPU than corresponding Tranzistow parts. Of course, Tranzistow can import Diodow patches for ages but they are still played by Tranzistow engine which uses more CPU due to enormous complexity. Now such patches can be played by less CPU demanding Diodow engine directly, so there should be no suprise to hear up to 96 Diodow voices from a single core (1 main part + up to 5 sub parts). And even if this is a pure Diodow engine, it still benefits from many Tranzistow features like full 32-slot modulation matrix, more LFO shapes and envelope curves, arpeggiator, step sequencer, chord memory, unisono voices, up to 32 voices of polyphony per main part when borrowing voices from a sub part, note-on modulation matrix, etc. Almost all multimode/multitimbral features are supported, including EQ and sidechaining from Tranzistow parts, but Diodow parts cannot be used as sidechain sources. Furthermore, there is a special Tranzistow+Diodow combined engine with Tranzistow main part and up to 5 Diodow sub parts which can also be processed through that Tranzistow part, for example - if you want to process a Diodow patch through the beautiful Tranzistow reverb etc.

This ability will also be used if I decide to develop some new X2 engines in the future, but without the danger of compromising Tranzistow functionality because I don't have to squeeze everything into Tranzistow engine anymore, which can possibly lead to bugs and instabilities.

Note: Although you can switch the engines in realtime, it is not recommended to switch them while holding the note as different engines can have different note-on initialization. Also, with Multithreading=1 mode the engine switching is not supported for core/thread #1, meaning that part #1 and all related sub parts are always tied to Tranzistow engine in this mode.

[*] Additive+FM/GPU Engine (internal versions only, not available outside my studio yet)

Additive+FM engine is a separate engine running on a graphics card (GPU) in parallel with the main Tranzistow engine. It contains 4 multi parts with 16 additive voices (for 64 voices in total), each voice with 4 additive oscillators, each oscillator with 240 dual-harmonics and internal virtual sync oscillator. Dual-harmonic is a combination of two harmonics where the second one is PW phase-shifted in order to have pulse-width at additive level (so, basically, there are 480 harmonics per oscillator * 256 oscillators = 122880 harmonics processed per sample). Each dual-harmonic has two 8-stage loopable envelope, one for amplitude and one for frequency (this is 2 * 61440 = 122880 envelopes processed per sample). All oscillators are fully bandlimited which means that you can, for example, sweep a saw wave and the engine will automatically remove harmonics over Nyquist as you pitch up and add harmonics as you pitch down. To avoid sudden jumps when removing and adding harmonics (which can be quite hearable in some cases) there is a built-in XFade functionality which crossfades between old and new harmonic window.

More or less, this is additive Tranzistow engine ported to GPU with a difference that Tranzistow works with up to 1024 dual-harmonics and it cannot calculate each harmonic in realtime because there isn't enough CPU power to do this together with everything else Tranzistow does at the same time. So, I am precalculating a lot of things in advance on Tranzistow and store calculated data in memory - this process is invisible to the user but needs a lot of RAM. Today's machines have enough RAM, anyway, so in many cases it is better to use more RAM when possible in order to save some CPU for other things. The other limitation is that Tranzistow harmonics don't have envelopes - you cannot control how individual harmonics change over time, which is quite a big difference between those two engines. Additive+FM/GPU can do all this and much more.

Each additive oscillator has it's own 136-band spectral filter (with 8Hz-20kHz range) meaning that 256 filters are running at the same time processing data for a total of 256 * 136 = 34816 bands per sample. This filter can be used for everything, from 136-band EQ via various resonant/non-resonant lowpasss, bandpass, highpass, formant (multiple bandpass) to some completely crazy and yet unheard filters. Furthermore, additive engine has many advanced FM capabilities. Four successive harmonics are grouped into one FM operator, so each additive oscillator (4 per voice) supports up to 60 operators - this is 240 operators per voice. Each operator supports two separate FM inputs with feedback capability, and with 4 harmonics (each with separate Index, Multiplier and Level parameters) per operator much more complex waveforms are possible. Operators can be connected in unlimited number of ways and can be freely combined with individual harmonics (for example, you can use harmonics 1..224 for additive synthesis and harmonics 225..240 as 4-operator FM) and harmonics can even be used for additive and FM duties at the same time.

Additive oscillators are sub-modules inside auxiliary oscillators - they share common pitch/frequency parameters together with the same output and they can be used as audio/modulation sources in tandem with the regular oscillators + additive output can be further processed through Tranzistow engine as any other audio source inside the synthesizer.

The complete 64-voice Additive+FM engine (4 parts, 16 voices per part, 4 oscillators per voice = 64 voices / 256 oscillators in total across 4 parts, each oscillator with 240 dual-harmonics, 60 FM operators and 136-band spectral filter, each dual-harmonic with 8-stage loopable amplitude and frequency envelopes, etc.) is optimized to the maximum and, running at 96kHz sample rate with 120-samples buffer, consumes around 60% of the available power on my Radeon RX580 graphics card (non-overclocked to keep the whole system almost completely "noise-free").

Additive oscillators have various resynthesis options and capabilities. Among the other things, WAV files can be loaded, resynthesized and used as the source material. All those features greatly expand the available sound palette and simplify the initial sound generation. Such complex and powerful Additive+FM engine has never been invented/implemented before, as far as I know :-)

Note: Although it is finished and thoroughly tested, Additive+FM/GPU engine is not available to the public. As I wrote before, I decided to keep the most powerful versions for myself only.

[*] 32/64-bit Linux Standalone Versions

Tranzistow is available on Linux in the form of 32-bit and 64-bit standalone audio applications (no separate Hrastow needed on Linux, it is embedded into standalone). In order to use it you need 32-bit or 64-bit Linux with at least GTK2 version 2.8 installed (GTK3+ is not supported yet). I recommend 64-bit Ubuntu Studio distribution as the best one currently available, IMHO. There is no Tranzistow installation per se - just unpack the ZIP archive into some folder and that's it. Tranzistow use ALSA, Jack or PortAudio/PortMIDI to communicate with audio and MIDI hardware, so you must have correct driver and firmware installed for the sound card you intend to use. You need libasound2 as well, although I suspect it is already installed together with ALSA. By default, Tranzistow will connect to "hw:0,0" audio device and to no MIDI device, but this can be changed through configuration/INI files. If you intend to use Jack then you must have Jack installed and configured properly, of course. The same with PortAudio/PortMIDI.

Tranzistow will search for Tranzistow.ini configuration file in /home/<UserName>/.config/Tranzistow folder (replace <UserName> with your Linux user name). If it doesn't find one there, it will search in a folder where Tranzistow application has been copied and run from. You can open Tranzistow.ini using a regular text editor and configure the following sections/options:

[Audio]
Driver=ALSA | Jack | PortAudio ; Audio driver to use (default: ALSA)
Device=... ; ALSA audio device to use with Tranzistow (ALSA only, default: hw:0,0)
SampleRate=... ; Sample rate to use (ALSA only, default: 44100)
BufferTime=... ; Size of cyclic audio buffer in milliseconds (ALSA only, default: 20)
FrameTime=... ; Size of one audio frame in milliseconds (ALSA only, default: 10)
Resample=0 | 1 ; Turn automatic ALSA resampling on/off (ALSA only, default: 0)
Format=16 | 24 | 32 ; Set the format / number of bits for audio data (ALSA only, default: 16)
Oversample=0 | 1 ; Turn internal Tranzistow 2x oversampling on/off (default: 0)

[MIDI]
Device=... ; ALSA MIDI device to use with Tranzistow (ALSA only, default: none)
Tempo=... ; Fixed tempo in BPM or 0 for automatic synchronization to incoming MIDI clock (default: 125)
TempoSmooth= ... ; Smooths out tempo changes when synchronizing to incoming MIDI clock (default: 0)

[Jack]
ClientName=... ; The name of Jack client (default: Tranzistow)
ServerName=... ; The name of Jack server (default: None)

[PortAudio]
AudioLibrary=... ; The name of PortAudio library (default: libportaudio.so)
AudioDevice=... ; The name of audio output device (default: None = default device)
InputDevice=... ; The name of audio input device (default: None = output device)
AudioBufferSize=... ; Size of audio buffer (default: 512)
MidiLibrary=... ; The name of PortMIDI library (default: libportmidi.so)
MidiDevice=... ; The name of MIDI input device (default: None = default device)
MidiBufferSize=... ; Size of MIDI buffer (default: 256)

Note: PortAudio/PortMIDI device names are not the same as ALSA device names! If you use Jack then you can have a separate MIDI device by specifying ALSA MIDI device.

Other configuration options are the same as described in previous sections.

To avoid unnecessary ALSA resampling, it is much better to match the requested sample rate to be the same as driver-supported one. Additional internal 2x oversampling can be turned on to achieve 88.2/96kHz quality on systems where higher sample rates are not available. Of course, CPU usage will be doubled in this case.

Configuration example:

```
[Audio]
Device=hw:1,0
SampleRate=48000
BufferTime=10
FrameTime=5
Oversample=1

[MIDI]
Device=hw:2,0,0
Tempo=0
TempoSmooth=0.5

[Editor]
Scale=125
```

Note #1: Upper/lower cases are important in all file names and paths because I am not really a fan of lowercase-only names. If an operating system has been designed with case sensitivity in mind then both upper and lower case should be used in my opinion.

Note #2: Configuration file should be writeable because Tranzistow will use it to store various data like window position, patch names/paths and MIDI channels for all parts, etc.

Linux version will sound, look and work more-or-less exactly like the Windows one, with the following exceptions:

(*) ALSA only: Only one (main) stereo audio output is currently available.
(*) ALSA only: No audio inputs are available yet.

Everything else is the same and almost everything written in this document applies to Linux version as well. All patches are interchangeable between versions too. BTW, despite the equal look and functionality on Linux and Windows, Tranzistow is a native Linux GTK2 application which doesn't use Wine or WineLib at all.

[*] 32/64-bit Linux VST Versions

A few words about the background: Full GUI Linux VST versions of Tranzistow and Diodow aren't possible due to some technical problems caused by the development environment I am using (Free Pascal and Lazarus). Something about the GTK2 interface inside those libaries is causing VST builds to crash when loaded into a 3rd-party hosts, and I couldn't found the cause. They are loading just fine into my Hrastow simple VST host, though.

So, a few years back, I was trying to find alternate solution to overcome this issue (because changing my development environment and rewritting everything for something else isn't an option). As a result, I developed an "RunVST" engine which was stripped of any widget toolkit (like GTK2). Then I used RunVST to build GUI-less Tranzistow/Diodow VST plugin which could be loaded into a host without crashing. After loading, the plugin executes the standalone Tranzistow/Diodow and uses interprocess communication to send the data between them. And it worked. I tested it with Hrastow and Ardour 4. The project could be saved, reopened and played without problems. The only hassle is that a standalone synthesizer opens in a separate window because it is a separate process running in it's own address space, which I think is a small price to pay and have the ability to load plugins into a VST host directly. But I never received any feedback on this from anyone and I decided to remove those plugins from the package, although the actual functionality wasn't never removed from the standalone engine. I don't use VST hosts anyway, so this wasn't of any importance to me.

But recently I rebuilt those RunVST-based plugins again and tested them with the latest Tranzistow/Diodow versions. And they still worked as expected. I also had some Linux users tested them and reported positive results, so I decided to return those VST plugins back into the package.

[*] Tranzistow Embedded Synthesizer Interface (Developers Area)

This section is intended for developers who want to develop alternate user interface (GUI) for Tranzistow or use Tranzistow as embedded sound engine for his own applications (synthesizers, games and similar). From build 16.08.2021/1 on, the download package contains two additional DLLs with embedded GUI-less Tranzistow version which can be loaded by a 3rd-party plugin and contains all functionality to control Tranzistow from other programs.

Some history first ... A few years ago, when I started working as a programmer for Gamepires (https://gamepires.com/) on their SCUM game (https://gamepires.com/games/scum/), I developed a small 64-bit game to test Unreal Engine 4. It was a simple 3D Breakout game where you could hit bricks by a ball on a horizontal surface with a paddle on the side towards the player and walls on other 3 sides. Each hit produced a different sound depending on the type of object and I immediately came to the idea to try to use Tranzistow for those sounds in realtime. So I developed an interface to control Tranzistow from outside. It was a regular 64-bit versions with added procedures/functions which remained functional to this day but I never documented them and no one (except me) ever used this functionality. Now I decided to push this to the next level and developed a totally GUI-less Tranzistow, stripped of all user interface functionality and dependencies on any visual control library. Beside existing functions, I also added the new ones for setting/getting values of all Tranzistow parameters etc. All this has now been included in Tranzistow32E.dll and Tranzistow64E.dll which cannot be loaded into the host directly and host won't recognize them when scanning VST plugins. But Hrastow can load them and if you load an existing Hrastow/Tranzistow project it will play just fine, only without any user interface. The engine remains closed and the programmer doesn't have access to any internal structures and objects. The embedded synthesizer contains an internal virtual non-visual editor which has all functionality of the regular editor and takes care of all parameter changes, patch loading/saving, etc. The programmer only needs to send parameter changes and the embedded engine will take care of the rest. All parameters can be accessed by names or indexes but only names are exposed because indexes can change between versions and builds (they won't change during the session, of course). If you want to use indexes to address the parameters you first to need the acquire the index for a particular parameter by using a dedicated function. Both patch and multi params are accessible in the same way.

The list with all Tranzistow parameters (including their types and minimum/maximum values) is here:

https://www.hrastprogrammer.com/hrastwerk/download/TWParams.txt

Extended ranges are in parenthesis. Multi params for all parts are at the end of the list. Ignore params starting with * because those are just section placeholders for easier navigation. The test unit with two usage examples (one as embedded DLL and the other which I used for testing during the development) is here:

https://www.hrastprogrammer.com/hrastwerk/download/TestowUnit.txt

It's Pascal because I use Delphi and FreePascal/Lazarus for all high-level software development (not limited to Tranzistow only). Any programmer capable of developing the GUI should have no trouble translating them to C/C++ or whatever. Basically, the whole process is very simple. First you have to load the DLL and create an instance of the embedded synthesizer:

```
Tranzistow: Pointer;
...
TWInitialize(PluginDir+'Tranzistow64E.dll');
Tranzistow:=TWCreateEmbedded('Tranzistow');
```

Loading looks like this (just get the addresses of all embedded procedures/functions and that's it):

```
var TWHandle: THandle;
...
  TWHandle:=LoadLibraryA(PAnsiChar(DLL));
  if TWHandle=0 then Exit;
  TWCreateEmbedded:=GetProcAddress(TWHandle,'CreateEmbedded');
  TWFreeEmbedded:=GetProcAddress(TWHandle,'FreeEmbedded');
  // ... other procedures/functions
```

On every block size and sample rate change you have to inform the embedded synthesizer about the change by using the appropriate procedures (this is mandatory, otherwise Tranzistow processing would not start):

```
TWSetBlockSize(Tranzistow,BlockSize);
TWSetSampleRate(Tranzistow,SampleRate);
```

In the test plugin I then call the test procedure to do some work:

```
// Open embedded editor interface (mandatory):
TWOpenEmbedded(Tranzistow,nil);

// Load a patch into Part #1:
TWPartLoad(Tranzistow,1,'C:\HrastWerk\Tranzistow\Patches\SampleTalk.hp');

// Change sample for Osc #1:
TWPartSample(Tranzistow,1,1,'C:\HrastWerk\Tranzistow\Patches\TranzistoWave.wav');

// Change Amplifier #2 Level by name:
TWSetFloatParamByName(Tranzistow,'Amplifier2.Level',1);

// Change FilterA #1 Cutoff by name:
TWSetFloatParamByName(Tranzistow,'FilterA1.Cutoff',1200);

// Get index for LFO #1 Rate parameter (LFO1_Rate: Int32):
LFO1_Rate:=TWGetParamIndex(Tranzistow,'LFO1.Rate');

// Change LFO #1 Rate by index:
TWSetFloatParamByIndex(Tranzistow,LFO1_Rate,2);

// Change LFO #1 Shape by name:
TWSetIntegerParamByName(Tranzistow,'LFO1.Shape',29);

// Save changed patch to another file:
TWPartSave(Tranzistow,1,'C:\HrastWerk\Tranzistow\Patches\Testow.hp');

// Change MIDI channel to 2 for Part #1 (TW-1):
TWSetIntegerParamByName(Tranzistow,'TW1.Channel',2);

// Change output level for Part #1:
TWSetFloatParamByName(Tranzistow,'TW1.Level',0.8);
```

In the audio processing routine you should first acquire the host tempo, send it to the synthesizer and then call audio processing procedure using the argument list compatible with VST2 Process/ProcessReplacing function:

```
var TimeInfo: PVstTimeInfo;
begin
   TimeInfo:=GetTimeInfo(kVstTempoValid);
   TWSetTempo(Tranzistow,TimeInfo.Tempo);
   TWProcessAudio(Tranzistow,Input,Output,Samples)
end;
```

... etc, etc. At the end you have to close the embedded synthesizer and unload the library:

```
TWFreeEmbedded(Tranzistow);
TWFinalize;
```

That's it. Everything written above applies to Diodow, too, because it contains embedded DLLs as well.

Very important: Anyone is free to get this interface and develop a GUI plugin for Tranzistow/Diodow or to use it as embedded sound engine, be it open source or closed source, providing that you don't sell your plugin or application commercially. In case of open source you have to choose the license which allows loading of closed source software components/libaries. I am still asking for the donation if you use Tranzistow, doesn't matter if you use GUI or GUI-less versions. GUI plugin developer can ask for a donation as well. **In case of commercial projects the usage of this interface is not free and I am asking for my share. Contact me to negotiate the deal before starting anything! Again: No commercial use in any way without compensation and my written permission!**

Now goes the reference with all embedded procedures/functions. In 32-bit version they are all declared as "cdecl". In 64-bit version it doesn't matter because the calling convention is the same for all compilers.

(*) function CreateEmbedded(PluginName: PAnsiChar): Pointer;

Creates and instance of the embedded synthesizer. Returns pointer to synthesizer object or zero if the synthesizer hasn't been created for some reason.

(*) procedure FreeEmbedded(Embedded: Pointer);

Close the embedded synthesizer and free the instance.

(*) procedure OpenEmbedded(Embedded,Parent: Pointer);

Open embedded editor interface. Parent is only used with regular GUI versions to embed the editor into a window of the calling application. If set to zero then editor will be detached into a standalone window.

(*) procedure NoteOn(Embedded: Pointer; Channel,Data1,Data2: Byte);

Simplified note-on processing.

(*) procedure NoteOff(Embedded: Pointer; Channel,Data1,Data2: Byte);

Simplified note-off processing.

(*) procedure PartEdit(Embedded: Pointer; Part: Int32);

Activate editing for a particular part and load the patch into the internal editor. Only one part can be active (edited and loaded) at the same time.

(*) procedure PartFile(Embedded: Pointer; Part: Int32; FileName: PAnsiChar);

Change file name for patch in a particular part.

(*) procedure PartInit(Embedded: Pointer; Part: Int32);

Intialize patch in a particular part.

(*) procedure PartLoad(Embedded: Pointer; Part: Int32; FileName: PAnsiChar);

Load patch into a particular part. Loading doesn't change the active part/patch.

(*) procedure PartSave(Embedded: Pointer; Part: Int32; FileName: PAnsiChar);

Save patch into a particular part. Saving doesn't change the active part/patch.

(*) procedure PartSample(Embedded: Pointer; Part,Osc: Int32; FileName: PAnsiChar);

Load sample into the oscillator of a particular part. Send empty string to clear the sample.

(*) procedure ProcessEvent(Embedded: Pointer; Status,Data1,Data2: Byte);

Simplified MIDI event processing.

(*) procedure ProcessEvents(Embedded: Pointer; Events: PVstEvents);

Full event processing. The Events argument should be a pointer to the array of records/structures compatible with VstEvent structure from VST2:

```
VstEvent=record
  EventType: Int32; // VstEventTypes
  ByteSize: Int32; // Size of this event, exclufing EventType and ByteSize
  DeltaFrames: Int32; // Sample frame related to the current block start sample position
  Flags: Int32; // Generic flags
  Data: array [0..15] of Byte; // Event data, size may vary depending on event type
end;
```

For MIDI events the structure should look like this:

```
VstMidiEvent=record
  EventType: Int32; // kVstMidiType
  ByteSize: Int32; // SizeOf(VstMidiEvent)
  DeltaFrames: Int32; // Sample frame related to the current block start sample position
  Flags: Int32; // VstMidiEventFlags
  NoteLength: Int32; // Not used
  NoteOffset: Int32; // Not used
  MidiData: array[0..3] of byte; // 1 to 3 MIDI bytes; midiData[3] is reserved (zero)
  Detune: ShortInt; // Not used
  NoteOffVelocity: Byte; // Not used
  Reserved1: Byte; // Reserved for future use
  Reserved2: Byte; // Reserved for future use
  SysExSize: Int32;
  SysExDump: PByte;
end;
```

SysExSize and SysExDump are specific for Tranzistow but should be ignored (set to zero) because Tranzistow system exclusive implementation, although functional, isn't documented and can be used under Hrastow only.

(*) procedure ProcessAudio(Embedded: Pointer; Input,Output: Pointer; Samples: Int32);

Call embedded audio processing. Argument list should be compatible with VST2 Process/ProcessReplacing function.

(*) procedure SetBlockSize(Embedded: Pointer; BlockSize: Int32);

Set embedded synthesizer block size.

(*) procedure SetSampleRate(Embedded: Pointer; SampleRate: Int32);

Set embedded synthesizer sample rate.

(*) procedure SetTempo(Embedded: Pointer; Tempo: Single);

Set embedded synthesizer tempo.

(*) function GetChunkData(Embedded: Pointer; Data: PPointer): Int32;

Get chunk data from the embedded synthesizer. It contains multi data only because patches are saved to files.

(*) procedure SetChunkData(Embedded: Pointer; Data: Pointer; ByteSize: Int32);

Send chunk data to the embedded synthesizer which will then adjust the appropriate multi data and load the patches for all parts contained in the chunk. Parts which are not in the chunk will remain untouched. The GUI plugin doesn't have to use GetChunkData/SetChunkData because it can implement it's own format for multitimbral data.

(*) function GetParamIndex(Embedded: Pointer; Param: PAnsiChar): Int32;

Get index of a particular parameter or -1 if the parameter doesn't exist (cannot be found by name).

(*) function GetIntegerParamByIndex(Embedded: Pointer; Index: Int32): Int32;

Get integer value of a particular parameter by index or 0 if the parameter doesn't exist (index out of range).

(*) function GetIntegerParamByName(Embedded: Pointer; Param: PAnsiChar): Int32;

Get integer value of a particular parameter by name or 0 if the parameter doesn't exist (invalid name).

(*) function GetFloatParamByIndex(Embedded: Pointer; Index: Int32): Single;

Get float value of a particular parameter by index or 0 if the parameter doesn't exist (index out of range).

(*) function GetFloatParamByName(Embedded: Pointer; Param: PAnsiChar): Single;

Get float value of a particular parameter by name or 0 if the parameter doesn't exist (invalid name).

(*) procedure SetIntegerParamByIndex(Embedded: Pointer; Index: Int32; Value: Int32);

Set integer value of a particular parameter by index or do nothing if the parameter doesn't exist (index out of range).

(*) procedure SetIntegerParamByName(Embedded: Pointer; Param: PAnsiChar; Value: Int32);

Set integer value of a particular parameter by name or do nothing if the parameter doesn't exist (invalid name).

(*) procedure SetFloatParamByIndex(Embedded: Pointer; Index: Int32; Value: Single);

Set float value of a particular parameter by index or do nothing if the parameter doesn't exist (index out of range).

(*) procedure SetFloatParamByName(Embedded: Pointer; Param: PAnsiChar; Value: Single);

Set float value of a particular parameter by name or do nothing if the parameter doesn't exist (invalid name).

(*) function GetParamList(Embedded: Pointer): PAnsiChar;

Get a list of all parameters with names separated by CR (ASCII 13) character. Multi params for all parts are at the end of the list. Ignore params starting with * because those are just section placeholders for easier navigation. This function can be used to automate GUI control creation: get the list of all parameter names, iterate through it, get properties for each parameter and create the appropriate control.

(*) function GetParamPropsByIndex(Embedded: Pointer; Index: Int32): PParamProperties;
(*) function GetParamPropsByName(Embedded: Pointer; Param: PAnsiChar): PParamProperties;

Get parameter properties by index or name. The result is a pointer to parameter properties structure:

```
TParamProperties=record
  Kind: Int32; // 0=Unknown, 1=Button, 2=Combo, 3=Slider
  FloatMin,FloatMax,FloatExtMin,FloatExtMax: Single; // Min/Max Float Values (normal / extended range)
  IntMin,IntMax,IntExtMin,IntExtMax: Int32; // Min/Max Integer Values (normal / extended range)
  Entries: PAnsiChar; // List with all entries separated by CR (Button and Combo only, zero if no entries)
end;
```

Zero will be returned for out-of-range indexes and invalid names. Entries are as they should be shown on the GUI. It is not needed to acquire properties for the same parameters which belong to different objects like oscillators, filters, LFOs, etc. For example, "Osc1.Waveform", "Osc2.Waveform", "Osc3.Waveform" and "Osc4.Waveform" all have the same properties, so it is enough to get properties for "Osc1.Waveform" only and use them for the others.

Important: Patch loading and parameter changing should be executed from the GUI thread. There is a possibility to enqueue parameter change from other threads by using the following procedures, so the actual changes will be triggered by EditMod timer (can be used for automation if parameter changes are triggered from non-GUI threads):

(*) procedure QueIntegerParamByIndex(Embedded: Pointer; Index: Int32; Value: Int32);

Enqueue integer value of a particular parameter by index or do nothing if the parameter doesn't exist (out of range).

(*) procedure QueIntegerParamByName(Embedded: Pointer; Param: PAnsiChar; Value: Int32);

Enqueue integer value of a particular parameter by name or do nothing if the parameter doesn't exist (invalid name).

(*) procedure QueFloatParamByIndex(Embedded: Pointer; Index: Int32; Value: Single);

Enqueue float value of a particular parameter by index or do nothing if the parameter doesn't exist (index out of range).

(*) procedure QueFloatParamByName(Embedded: Pointer; Param: PAnsiChar; Value: Single);

Enqueue float value of a particular parameter by name or do nothing if the parameter doesn't exist (invalid name).

In all procedures/functions the embedded synthesizer will just ignore calls with out-of-range parts, oscillators and indexes as well as invalid parameter names. No error will be returned and no exception will be raised. It's the responsibility of a GUI plugin to send correct data to the synthesizer and to handle all user errors.

Regular GUI Tranzistow versions also contain the above interface with the addition of following procedures:

(*) procedure CreateEditor(Embedded: Pointer);

Create editor form with all parameters and controls. If not called then OpenEmbedded will create the editor implicitly.

(*) procedure Demodify(Embedded: Pointer; FromPart,ToPart: Int32);

Turn off modified status for requested parts. If FromPart=0 then start with the part 1 and if ToPart=0 then end with the last part.

(*) procedure DestroyEmbedded(Embedded: Pointer);

The same as FreeEmbedded but turns off modified status for all parts first.

(*) procedure EditorGetRect(Embedded: Pointer; Rect: PPERect);

Get the editor rectangle. Rect is a pointer-to-pointer to the resulting record.

PPERect=^PERect;
PERect=^ERect;
ERect=record Top,Left,Height,Width: Int16 end;

(*) procedure EditorSetRect(Embedded: Pointer; Rect: PERect);

Set the editor rectangle. Rect argument is a pointer to the record containing coordinates and size. Should be called between CreateEditor and OpenEmbedded .

(*) procedure EditorOpen(Embedded: Pointer; Parent: Pointer);

Open the editor and show editor form. Parent is the same as in OpenEmbedded.

(*) procedure EditorClose(Embedded: Pointer);

Close the editor and hide editor form.

(*) procedure EditorIdle(Embedded: Pointer);

Execute editor idle processing.

(*) procedure EditorShow(Embedded: Pointer; Left,Top: Int32);

Show editor form at the requested coordinates. Used to show and editor which has been already opened but hidden.

(*) procedure EditorHide(Embedded: Pointer);

Hide editor form without closing.

(*) procedure SetModifyProc(Embedded,Caller: Pointer; Proc: TModifyProc);

Set the procedure used to signal to the calling application that some synthesizer parameter has been modified. Caller is a user-definable pointer and can be zero. Proc is a pointer to the signaling procedure:

TModifyProc=procedure (Modifier: Pointer);

(*) procedure PartUpdate(Embedded: Pointer; Part,Update,Data: Int32); cdecl; export;

Do specific part updates. Only UpdateWavetables=1 has been implemented for now.