Segment Synthesizer

Andre SKLENAR
Kakapo Electronics s.r.o.
Na Slupi 5
Prague, Czech Republic, 128 00
andre.sklenar@gmail.com

Michal SYKORA
Kakapo Electronics s.r.o.
Na Slupi 5
Prague, Czech Republic, 128 00
michal.sykora.sk@gmail.com

Martin PATERA
Kakapo Electronics s.r.o.
Donatellova 5
Prague, Czech Republic, 100 00
mzstic@gmail.com

Amir HAMMAD
Kakapo Electronics s.r.o.
Janka Krala 18
Nitra, Slovakia, 94901
amir.ozk@gmail.com

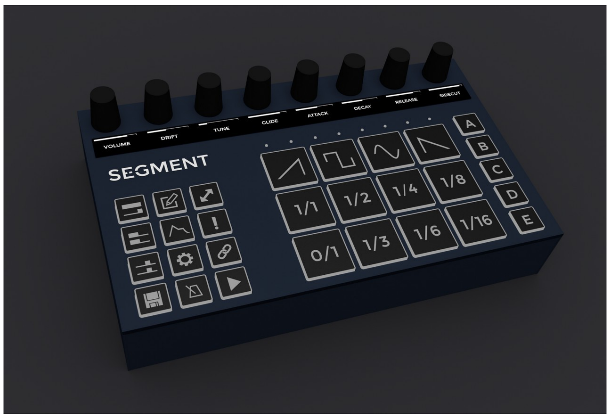
Abstract

This work aims to describe the new hardware *Segment Synthesizer*, its development process, cover a description of the original *Segment Synthesis* and its properties, advantages and limitations. It will also clarify the way the hardware controls and the user interface were designed, Segment's connectivity and its audio features.

Segment's accompanying features, such as the *Segment Cloud* and the *Segment Manager* will also be covered.

Keywords

Segment, Synthesis, Open Source, Open Hardware, Synthesizer



1 Introduction

Segment Synthesizer is an open source and open hardware monophonic digital hardware synthesizer. It features an original synthesis method invented and designed by Andre Sklenar called *Segment Synthesis*. It has no keyboard, so the user has to connect their own or drive *Segment* with a different input. A beats per minute - synced, low frequency oscillator modulated low pass filter is present on *Segment*, directly controllable by primary silicone pads on the front panel. Segment contains several other features, such as *drift* for intentional parameter instability or a flexible parameter routing.

From the hardware perspective, *Segment* runs bare-metal on an ARM Cortex-M4 chip, features a wide-ranging connectivity and supports multiple data protocols.

The first accompanying software is the *Segment Cloud*, where *Segment* users will be able to share their presets along with a short sound context example.

The second accompanying software application is the software that allows for *Segment* - computer communication for up/downloading presets and firmware and creating backups. It will also allow the users to browse the cloud and up/download content directly from/to *Segment*.

2 Segment Synthesis

Segment Synthesis is an original synthesis method aiming to create very thick and harmonically rich timbres in a most simple and elegant way.

2.1 The basic principle

The main idea behind *Segment Synthesis* is that only half of a period of a primitive waveform[1] is needed to get enough timbral information about the waveform. When applying this approach to the waveform construction, after the first half period of a waveform is rendered, another first half of a different primitive waveform is rendered, but one octave higher. When that is finished rendering, we continue with another half period of any primitive waveform, one octave higher again and so forth.

We call each of these waveform bits 'segments' – hence the name of the synthesis.

This basic idea is the main drive behind segment synthesis. Obviously, the approach described above is not practical, so fixed limitations and a specific implementation had to be adopted.

The closest relative of Segment Synthesis is Xenakis' Gendy Synthesis[2].

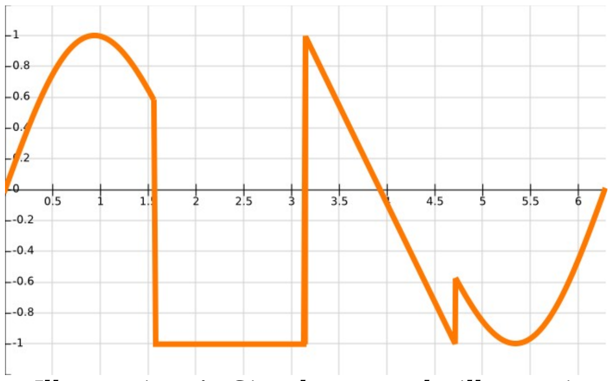


Illustration 1: Simple example illustration of Segment Synthesis (no frequency modulation).

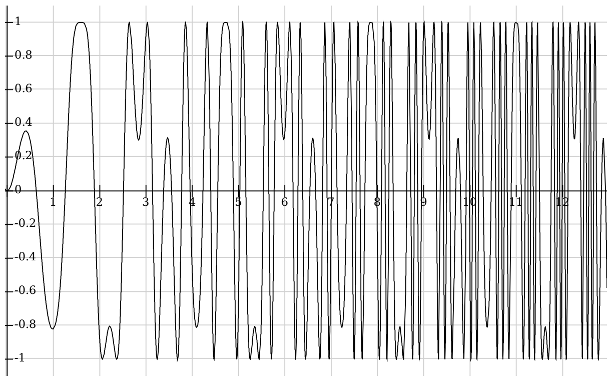
2.2 Frequency modulation

To further increase the harmonic spectrum employed in today's music production, a harsh approach was undertaken. Each segment is frequency modulated by itself using this equation (in this case the primitive waveform is a sine wave):

$$y = \sin(x * \sin(nx))$$

Where x is the phase accumulator running from 0 to 2π and n is the frequency modulation coefficient.

Should the phase accumulator be allowed to run indefinitely, this approach would produce a vastly non-periodic waveform:



Because of this and other practical reasons, the phase accumulator is reset to 0 when $x>2\pi$. This way the output is always periodic.

2.3 Segment implementation

Segment implements this synthesis method by providing the musician with 6 segments, where the waveform of each segment can be set by blending between one of the four primitive waveforms (as described earlier), the amount of frequency modulation, panning and volume. The first segment is a whole period. Second and third segment occur in the same time frame, but can be set independently and cover half a period and tune on double the fundamental frequency.

The fourth and fifth segments cover the time frame from ½ the period to ¾ the period and play 3 octaves higher. Again, the user can set their waveform, amount of frequency modulation, stereo panning and volume independently.

The sixth and final segment renders from ¾ of the period until the end and plays four octaves up.

This particular *Segment Synthesis* implementation is neither the best or the only one working. It is simply the implementation we have tuned to our liking and is one of many possible combinations.

2.4 Practical results

Musicians always seek new ways to craft their sounds. As a direct result, various post-processing is involved in our search for harmonically rich synthesizer patches. It is not uncommon to distort and saturate almost every channel, add choruses, double the tracks and detune their synthesizer voices to reach that goal.

Segment Synthesis tries to tackle this issue right at its root – the actual synthesis.

Direct comparisons with usual additive/subtractive synthesizers show that even when not employing frequency modulation, using the same amount of oscillators, *Segment* produces a much richer harmonic spectrum[3].

Due to a very high number of harmonics, the fundamental frequency can be tuned down way below 20Hz, while still retaining a significant amount of sonic information, which were previously multiple-aliases. This further widens the range of possible timbres.

2.5 Pitfalls

Due to *Segment Synthesis*' sharp edges (similar to hard-sync), *Segment* gets into aliasing very easily. The number of harmonics is so high, 2x oversampling did not produce satisfactory results and this was already over the edge of the computational power available. As this is clearly a problem, more research is needed on this topic.

Another, this time aesthetic, problem is that sometimes it is difficult to fit Segment into existing music because of the high amount of harmonics. Again, more research is needed to gain knowledge about how to control the amount and content of higher harmonics.

3 Auxiliary audio features

Because no synthesizer would suffice with just raw synthesis, Segment has several functions to make it a full-featured synthesizer device.

3.1 Beats per minute synced low frequency oscillator modulated low pass filter

Segment builds on a simple notion that most of today's popular music is grid-based. Based on that, we have implemented a low pass filter controlled by a low frequency oscillator which is permanently synchronized with the main tempo. This tempo can be tapped, set manually or received via any of the communication protocols we support (more on that later).

3.1.1 User interface

To control the LFO, which is one of Segment's main features, Segment has 12 primary silicone pads. These are in a 4x3 arrangement, where the top 4 change the LFO waveform from one of 4 primitive waveforms (as described earlier).

Two bottom rows allow the user to trigger different BPM-synced LFO speeds based on main tempo divisions. These are none, a whole note, a quarter note, an eight note, a triplet, a sextolet and so on.

Because the main tempo is running constantly, the musician never gets off beat with his LFOs. One can visualize this as having 8 running LFOs in parallel and the user is only choosing through which the signal is routed. The user still may manually stop the playhead if there is a need – such as before a tempo change during live performance, the musician can pre-tap the tempo and start on the first beat.

3.2 Sidecut filter

The sidecut filter is a single parameter with no filtering at the top position (0.5 on scale from 0 to 1). The filter progressively acts as a low pass filter for n<0.5 and as a high pass filter for n>0.5. The filter is 4 pole with a resonance parameter. This filter also modifies the main low frequency oscillator-controlled low pass filter by modifying its upper (or lower) limits.

3.3 Sounds and presets

You can store 12 presets in total. This limitation is not due to memory limits, but because of playability and easy preset recalling during a live performance. When pressing the 'preset' button, the 12 primary pads' function changes to preset selectors. Because there is an RGB LED under each pad, the presets can be identified by user-chosen color coding (set by a single encoder as HSV). Each of these presets include 5 *sounds*. The sound buttons are to the left of 12 primary pads and these select one of 5 available *synthesis setups*. All parameters are stored preset-wise, except for the sounds. To sum up – there are 12 presets, each having 5 sounds.

3.4 Playability

Segment's layout is designed in a way so it can be played with a single hand, leaving the other hand to other controllers or keyboards. With this approach, the musician can easily change low frequency oscillator speed, low frequency oscillator waveform and sound simultaneously for each note played.

3.5 Drift

To make the sounds more organic and alive, we implemented a feature we call *drift*. Modifying one parameter (which internally modifies several of them), the user varies the amount of 'instability' of the synthesis parameters. This is done using a recursive filter:

$$x = x + (T_x - x)/n$$

Where x is *current parameter value*, T_x is *target parameter value* and n is the *drift speed coefficient*. The *target parameter value* is randomly rolled at a given interval. All these and the maximum distance of the *target parameter value* from the user-set parameter value are modified by the *drift* parameter. This way the *current parameter value* randomly drifts around its user-set pivot point, creating slight or drastic timbre changes in time.

4 Hardware

The Segment core runs bare-metal on an ARM Cortex-M4 clocked at 168MHz. There are two more ARM Cortex-M0 chips, one for controlling the displays and encoders and one controlling the buttons and pads. The chips communicate via an SPI bus. The code utilises the libopencm3 peripheral library. The audio engine runs on 48kHz at 32-bit and the DAC operates on 48kHz at 24-bit.

4.1 No keyboard

Segment has no included keyboard to play it. There are several reasons for this decision – first, most electronic musicians already own (sometimes a lot of) keyboards and controllers. It seemed redundant to provide them with more as there is only a limited amount of devices you can put on your table. If the user doesn't have a keyboard, he or she can either get one or drive *Segment* from a computer, a sequencer etc. Second, we all like different keys – touch response, size, number of octaves and so forth and we don't believe in a 'one size fits all' philosophy regarding keyboards. Lastly, the absence of the keyboards allowed us to cut down the costs greatly.

4.2 Auxiliary board

The top auxiliary board consists of 8 rotary encoders and 8 monochromatic OLED displays below each of them. The displays show the current function of each encoder depending on the control layer the user is in and the current value of that parameter on a bar graph.

4.3 Control buttons

To the left of the 12 primary pads,

there are 3x4 buttons that either select the layer the user is in or have a single use (such as tap tempo). The three main layers are the segment synthesis layers, which select which segment's parameters are being modified by the encoders.

4.3.1 Linker

The linker button (a small chain symbol) triggers the linker layer, which allows the user to link parameters together or route them from and to any port

4.3.2 Scale

The scale button (with a resize symbol) modifies the amount of modulation applied to parameters via external sources.

4.4 The user button

The user button's function (marked with an exclamation mark) can be defined by the user. This button can have any function from any layer or one that is not currently assigned anywhere, such as the killswitch.

4.5 User layer

When no layer is selected on the control buttons, the *user layer* is selected. The user can pull any 8 parameters from any layers to the *user layer* so they are immediately accessible. The user layer is stored preset-wise, because with some presets one wants to have hands-on control over different parameters than with others. This comes in handy and tries to get closer to the 'one knob per function' approach of the analog synthesizers.

5 Connectivity

Regarding audio, *Segment* provides a stereo balanced XLR output and a stereo 6.3mm headphone jack.

Furthermore, *Segment* has 2 USB MIDI inputs, so the user can connect their USB MIDI controllers and keyboards as *Segment* acts as a USB host.

Segment also acts as a class-compliant USB MIDI device (in and out) and support OSC.

Due to some degree of backward-compatibility, the device provides the user with a MIDI input in the form of a 3.5mm jack.

It is important to note that every parameter on the device is exposed via a MIDI CC.

Because we like analog and some of us have some modular gear we like to work with, *Segment* has 4 control voltage inputs and 2 control voltage outputs. These can be mapped to any parameter and also linked-through (CV in – USB MIDI out) so *Segment* can act as a CV-USB-MIDI converter.

Segment also contains UART input and output exposed on a 3.5mm stereo jack. The user can connect their DIY gadgets and control *Segment* with anything they can come up with.

6 The computer software

A specialised software (written in Java for cross-platformity) can be downloaded. The users can manage their presets, name them, tag them, backup and create them. The software also provides means for flashing updated or different versions of the firmware and uses a custom protocol due to the computational limits of the onboard MCU. From within the application, the user is also able to reach the Segment Cloud.

6.1 The Segment Cloud

The *Segment Cloud* is a cloud-based service providing for a platform where the users can share, comment, tag and download the presets/sounds either to their local database or to *Segment* directly.

Since *Segment* has no USB Audio interface, short sound snippets of around 8 seconds will be rendered to audio files on the server based on user's MIDI input. The synthesis code is mostly portable, so the server will emulate exactly the same audio output *Segment* would produce.

7 Tools used during development

The programming was mostly done with Geany and Eclipse, compiling with GCC.

Java programming was done with NetBeans, PCB design was made possible by gEDA and we used FreeCAD for CAD parts designs.

8 Licencing

Since the team believes in open source and open hardware, the whole code, PCB designs and CAD drawings will be available under GPLv3 licence. We have chosen the GPLv3 licence because of its 'virality' which allows us to protect ourselves from big corporations, but not restricting the users to study, modify, use and contribute to the code.

None of *Segment*'s parts are patented and we do not consider doing that anytime in the future.

Acknowledgements

We would like to thank all who provided us with valuable advice, especially the #lad and #libopencm3 channel at FreeNode on IRC.

1 Saw, sine, square and reverse saw.
2 Stochastic Synthesis: Origins and Extensions, Sergio Luque, 2006
3 http://www.segmentsynth.com/synthesis/