Synstack
Audio DSL for embedded devices

Project aims

Primarily audio toolkit for embedded devices (STM32F4/F7)
Desktop / mobile use is bonus
Use for audio installations using lowcost devices
Arbitrary sample-precise modulations
Easily extendable
Easily scriptable
Livecoding (via BT/UART/USB)
Lightweight (< 32KB, baremetal or linkable as static library)
High performance (ASM hotspots)
Low latency

Status

In active development since May 2015
Currently in 4th iteration / complete rewrite
Project used as teaching context for workshops

Latest version 100% based on a Forth stack VM (24KB, still written in C)
General purpose language
Forth dialect based on JonesForth
Threaded bytecode with customizable inlining
Native, buffer oriented audio vocab
Floating point support (incl. various cmath functions)
Basic string support
Basic memory management vocab
Disassembler (“see” word)

Currently 25+ operators

Oscillators (unlimited & bandwidth limited, PolyBLEP)
    _ sin, saw, square, WT, morphing WT, bezier, Karplus-Strong
    _ also used for LFOs
Single tap delay line (n-channel w/ feedback)
Filters (IIR x4, Biquad x7, Moog ladder LP, 1pole, 4pole LP, Allpass)
Panning
Sample & Hold
Saturation
Foldback distortion
Buffer algebra (+,-,*,/,madd,addm,mix etc.)
Audio input
MIDI support (currently STM32 only)
FFT (ARM only, using CMSIS DSP lib)

Why Forth?

“Lisp is the ultimate high-level language,
Forth is the ultimate low-level language.”

Richard Jones

Why Forth?

Super lightweight
Heavy focus on simplicity
Easy bootstrap, most of the language defines itself
No real syntax, completely shapeable
Easy to create highlevel, domain-specific abstractions
Unique combination of compiled & interpreted execution
Code-generating words (think Lisp macros)
Livecoding
Used successfully in low-powered envs since 1960s

Above all: Stack based & fast(!)

Also: ...because it's been an enlightening learning experience

Why Forth?

```
\ Forth is stack based & uses postfix notation
\ All items go on stack in order given
\ Known words are executed and manipulate the stack
\ usually placing result back on stack
1 2 + .
3
\ here we define a new word “double” which computes x + x
: double dup + ;
21 double .
42
\ define another word, using double
: quadruple double double ;
21 quadruple .
84
```

Why Forth?

```
\ stack based ADSR with LFO modulation and single OSC
\ init data structures
0.0f >osc val> lfo1
0.0f >osc val> osc1
0.005f 0.1f 0.5f 1.0f 0.25f >adsr val> env
\ compute LFO (sine oscillating @ 0.125Hz between 0.1 .. 1.9)
lfo1 0.125f hz osc-sin-c 0.9f b* 1.0f b+
\ now use LFO as time factor to compute modulated envelope
env swap adsr
\ compute main osc
osc1 440.0f hz osc-sin-c
\ multiply with envelope
b*
\ final output left on stack
```

soundcloud.com/forthcharlie

Project audio documentation

hackaday.io

Semi-regular project log

thi.ng/ws-ldn-4

Previous workshop repository

workshop.thi.ng

Upcoming workshops

resonate.io/2016/

Next 3-day workshop in Belgrade
(11 - 13 April)

asm.thi.ng

Thanks :)

Karsten Schmidt
k@thi.ng
@toxi
@forthcharlie