The Book of CLAMS

M. Edward (Ed) Borasky

2023-12-15

Table of contents

Command Line Algorithmic Music System (CLAMS)

> "I've never seen a happy clam. In fact, most of them were really steamed." ~ M. Edward (Ed) Borasky

Overview

`CLAMS` is a text-based interactive environment for composing and performing music and visuals on a Pimoroni PicoVision. It can be made to work on other boards using the RP2040 microcontroller, but you will need to buy additional hardware and port some code.

How does it work?

`CLAMS` is a domain-specific language built on a Forth compiler / interpreter. The user connects to the board via a USB or UART serial connection and enters `CLAMS` / Forth code interactively.

Why Forth?

1. Forth (Brodie 2022) is an extensible interactive operating system. It supports editing, assembling, compiling, debugging and running real-time tasks from a terminal.
2. Forth is efficient. A well-designed Forth will usually run a task at no worse than half the speed of a hand-optimized assembly version. `CLAMS` will have several optimizations built in for the ultimate speed.
3. Forth is lean. There are very few concepts to learn, there is minimal run-time overhead in RAM, and the whole package takes up much less flash space than MicroPython or CircuitPython.

What about Forth standard (Forth 200x Committee 2012) compatibility?

`CLAMS` is an extended subset of the standard. It won't contain all of the standard's core word set, and it will contain some extensions to support the Raspberry Pi Pico C/C++ SDK, RP2040 assembly language programming, the PicoVision hardware, cooperative multitasking, and high-speed digital signal processing.

What about portability?

Within the RP2040 ecosystem, as long as the PicoVision and C/C++ SDK work, porting should be straightforward, though tedious. And you will undoubtedly need to buy more hardware.

Outside of the RP2040 ecosystem, there are a number of other micro-controller music boards, most notably the Electro-Smith Daisy and the Rebel Technology OWL platforms. But they have their own SDKs, so there's not much need to port `CLAMS` to them.

There are also a number of audio projects that use the Teensy® USB Development Board, which has a comprehensive audio library. Like the first two, it has its own SDK. And the Daisy, OWL and Teensy processors are all more powerful than the RP2040.

By contrast, there's not much music-specific development software for the Raspberry Pi Pico / RP2040. There are some simple demos, a few do-it-yourself hardware offerings, and there's the Allen Synthesis EuroPi, a Eurorack module with an open source MicroPython software platform. `CLAMS` will be a different approach.

The overall concept is an interactive language for making music on Raspberry Pi Pico / RP2040. I'm aiming for ChucK (Salazar et al. 2014) semantics with Forth syntax - a single text-based language to implement both the definitions of synthesized instruments and the sequences of sounds they make, intended for live coding / algorave performances.

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

---

1 About My Music

> "Markovs of the world unite! You have nothing to lose but your chains!" ~ M. Edward (Ed) Borasky

1.1 AlgoCompSynth

"AlgoCompSynth" is a word I made up to describe what it is that I want to do. It's a compression of "algorithmic composition and digital sound synthesis." That's a pretty broad class of music; to narrow it down, the following are my main inspirations.

- Iannis Xenakis' *Formalized Music* (Xenakis 1992). Xenakis took the applied mathematics of his day, for example, operations research and game theory, and used these algorithms to create scores for conventional and electronic performers. He also invented a technique called "Dynamic Stochastic Synthesis", which uses Markov processes to specify not just the score of a piece but the parameters of the sound waveforms (Hoffmann 1996; Brown 2005; Xenakis 1992).
- Alternate tunings. Primary among these is William Sethares' *Tuning, Timbre Spectrum, Scale* (Sethares 1998, 2013). Also influential: Wendy Carlos (Carlos 1987), Harry Partch (Partch 1979), Erv Wilson (Narushima 2019) and Nick Collins (Collins 2008, 2012).
- Physical modeling synthesis. A comprehensive reference can be found at (J. O. Smith accessed 2023-10-21).
- Spectral music. This is another advanced synthesis methodology; a recent reference is (Lazzarini 2021)

My current home for published music is on Bandcamp at https://algocompsynth.bandcamp.com/. I may move to Gumroad, which allows publishing a broader range of digital media artefacts than music and music videos.

1.2 Other AlgoCompSynth projects

1. AlgoCompSynth-One: This is a platform for doing high-performance digital signal processing and musical AI on NVIDIA GPUs. I currently support Windows 11 WSL Ubuntu 22.04 LTS and NVIDIA Jetson JetPack 5.

   If I can make everything work I will be supporting native Windows 11. `AlgoCompSynth-One` is based on Mamba, and Mamba runs on Windows, so unless other dependencies can't be found, it should be possible.

2. eikosany: This is an R package for algorithmic composition with musical scales derived by Erv Wilson and students of his theories.

3. consonaR: This is an R package to perform the computations described in *Tuning, Timbre, Spectrum, Scale* (Sethares 2013). This is a superset of any such algorithms that will be deployed in CLAMS; only calculations that need to be performed during a performance need to be deployed in CLAMS.

There is a fairly large overlap in functionality between `eikosany` and `consonaR`. There's a plan for refactoring the two, but that work is on hold until I get more of `CLAMS` done.

2 CLAMS-Forth Overview

"If you've seen one Forth, well, you've seen one Forth." ~ Unknown

2.1 Requirements

1. Compatibility with the Raspberry Pi Pico C/C++ SDK, drivers, and libraries: The ability to use the massive toolset the SDK and community open source projects provide is absolutely crucial to minimize developer time for complex projects. The USB and WiFi / Bluetooth stacks alone would take many months to duplicate in Forth. The primary references for this are Stephen Smith's *RP2040 Assembly Language Programming* (S. Smith 2021), and, of course, *Raspberry Pi Pico C/C++ SDK* (Raspberry Pi Ltd Accessed 2023-10-22).
2. Optimized for speed: CLAMS-Forth will be written in RP2040 assembly and will provide an RP2040 assembler.
3. Digital signal processing: A high-speed block floating point digital signal processing library will be provided.

2.2 Desiderata

1. Forth 2012 standard compatibility is a strong desire but not an absolute requirement. Much of the CORE word set and some of the CORE EXT word set will be implemented, but a number of tricky, risky or little-used CORE words will be omitted. Many of the two-cell operations, for example, aren't really needed in a Forth running on a 32-bit architecture like the RP2040.

   Parts of the Search-Order and Programming-Tools word sets will be implemented. I want to have the Block word set using blocks in flash memory, but that's not going to be in the first release.

   Custom word sets will be provided for high-speed digital signal processing and SDK / hardware access. ***All access to the hardware / system level operations will be performed via C language calls to the SDK.***

2. Cooperative multitasking: The RP2040 has two cores, and each core has a divide coprocessor and two interpolators. In addition, the RP2040 has two programmable input / output (PIO) blocks. Cooperative multitasking is the Forth way to exploit this available concurrency and is well-supported by the SDK.
3. Portability to other boards with the RP2040 microcontroller is possible, but is not on the roadmap yet. As with CLAMS itself, the initial target is the Pimoroni PicoVision.
4. Floating point support is desirable, but is a fairly low priority. Floating point arithmetic is convenient, and the RP2040 SDK provides optimized floating point libraries, but there's no hardware floating point arithmetic on the RP2040. It's too slow for real-time digital signal processing, so it's not obvious how useful this capability would be.

2.3 Design / architecture

The top-level design and architecture are based on Dr. Chen-Hanson Ting's *eForth Overview* (Pintaske and Ting 2018). In eForth, a small number of primitive words are implemented with a macro assembler, then the rest of the system is built in Forth on top of those words. Brad Rodriguez' CamelForth (Pintaske and Rodriguez 2018) also provides some implementation guidance.

The original plan for CLAMS-Forth was to use subroutine threading and inline code. There is a partial implementation, but I decided to switch to a more "traditional" but less efficient direct threaded implementation. There are two main reasons:

1. The branch instructions in the RP2040 are program counter relative, and the available target distances are limited by the instruction formats. In particular, the function call (`BL`) instruction has an available range of plus or minus 16 megabytes. While this is enough within the RP2040's flash or SRAM, it is *not* enough to cross the address space distance between SRAM and flash.

   CLAMS-Forth will have a dictionary split between flash and SRAM. The entire system dictionary - Forth virtual machine, text interpreter and compiler - will be in flash. Only words created by the user at run time will be in SRAM.

   Dr. Ting's *eForth for Discovery* (Pintaske and Ting 2019) gets around this limitation by copying the system dictionary from flash to RAM at boot time. But the hardware he used has more RAM and a more complete instruction set than the RP2040. The RP2040 only has 264 kilobytes of SRAM and much of that will be used for audio buffers. The more code and tables safely stashed away in the 2 megabytes of flash, the better.
2. Even with a dictionary entirely in SRAM at run time, constructing a `BL` function call instruction while compiling a user "colon" definition is a tricky process, difficult to document, understand and maintain. If you're interested in the details, see (Pintaske and

Ting 2019, sec. 3.5.5). I decided to switch to direct threading, rather than spend an extra week coding both a flash-to-SRAM relocation and an algorithm to compute a relative branch address that is then split into four separate bit fields of two 16-bit `thumb` instructions.

2.4 Status / roadmap

1. There is a partial implementation of a subroutine-threaded version with inlining. I am leaving that in the repository in a development branch but won't be working on it unless I find that the direct-threaded version cannot meet speed requirements. The subroutine-threaded version is in branch `forth-stc`.
2. The direct-threaded version is nearing a release. I expect to have the system dictionary, text interpreter and compiler done by January 1, 2024. It is in the branch `forth-dtc`.
3. The next step is to implement synthesis algorithms. The general process will be to prototype them in Forth, converting to assembler when needed. That will create the requirements for the digital signal processing library.

References

Brodie, Leo. 2022. "Starting Forth." FORTH, Inc. https://www.forth.com/starting-forth/0-starting-forth/.

Brown, Andrew. 2005. "Extending Dynamic Stochastic Synthesis." In *Conference Proceedings: International Computer Music Conference: ICMC 2005 Free Sound*, 111–14. Escola Superior de Musica de Catalunya.

Carlos, Wendy. 1987. "Tuning: At the Crossroads." *Computer Music Journal* 11 (1): 29–43.

Collins, Nick. 2008. "Errant Sound Synthesis." In *ICMC*.

———. 2012. "Even More Errant Sound Synthesis." In *Proceedings of the Sound and Music Computing Conference (SMC2012)*. Vol. 6.

Forth 200x Committee. 2012. "Forth 2012 Standard." Forth 200x Committee. http://www.forth200x.org/documents/forth-2012.pdf.

Hoffmann, Peter. 1996. "Implementing the Dynamic Stochastic Synthesis." In *Journées d'informatique Musicale*.

Lazzarini, V. 2021. *Spectral Music Design: A Computational Approach*. Oxford University Press. https://books.google.com/books?id=sns_EAAAQBAJ.

Narushima, T. 2019. *Microtonality and the Tuning Systems of Erv Wilson*. Routledge Studies in Music Theory. Taylor & Francis Limited.

Partch, H. 1979. *Genesis of a Music: An Account of a Creative Work, Its Roots, and Its Fulfillments, Second Edition*. Hachette Books.

Pintaske, J., and B. Rodriguez. 2018. *Moving Forth - Internals and TTL Processor: Forth Internals*. Amazon Digital Services LLC - Kdp Print Us.

Pintaske, J., and C. H. Ting. 2018. *EForth Overview: C. H. Ting*. Amazon Digital Services LLC - Kdp. https://books.google.com/books?id=OpEDygEACAAJ.

———. 2019. *Irreducible Complexity: EForth for Discovery*. Amazon Digital Services LLC - Kdp Print Us.

Raspberry Pi Ltd. Accessed 2023-10-22. "Raspberry Pi Pico C/C++ SDK." https://www.raspberrypi.com/documentation/microcontrollers/c_sdk.html; Raspberry Pi Ltd.

Salazar, S., A. Kapur, G. Wang, and P. Cook. 2014. *Programming for Musicians and Digital Artists: Creating Music with ChucK*. Manning.

Sethares, W. A. 1998. *Tuning, Timbre, Spectrum, Scale*. Springer London.

———. 2013. *Tuning, Timbre, Spectrum, Scale, Second Edition*. Springer London.

Smith, Julius O. accessed 2023-10-21. *Physical Audio Signal Processing*. http://ccrma.stanford.edu/~jos/pasp/.

Smith, S. 2021. *RP2040 Assembly Language Programming: ARM Cortex-M0+ on the Raspberry Pi Pico*. Apress.

Xenakis, I. 1992. *Formalized Music: Thought and Mathematics in Composition.* Harmonologia Series. Pendragon Press.