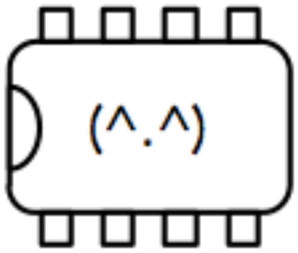
4th Saga ch3
4th Pedigree



www.ZENMCU.com

Draft 1 2023-03-22

Copyright © 2023 by Trenton Henry, All rights reserved

4th pedigree

It seems that I have ended up trying a lot of different implementation variants for my 4ths, each of which I use for a little while and then move on to the next improvement. So I decided it might be a good idea to write a list of them, just for my own reference.

mcu4th - This was after I finally understood CH-8 of Let Over Lambda and decided I needed to act on it. I had little idea what I was doing, but I had to start somewhere. I was using computed gotos for dispatch. I tried generating the dictionary and code, but I was fairly clueless about Forth overall.

ibtil + ibtui - The 'itty bitty threaded interpreted language' along with a crude tui to display trace data etc. This was when I was reading "Threaded Interpretive Languages". I was still exploring. This was still computed goto, but I tried macrology to define the dictionary.

z4th - This was a 64bit Forth with indirect threading. It primitives were actual functions. It was intended to be scaleable at build time so it could run on an MCU with 32bit pointers. That is when I realized that I'd be wasting over half of my RAM storing useless high bits in pointers. The idea of ZENMCU was gestating so I called it z4th.

u4th - This was a 16bit 4th running on a 64 and 32bit machines. It was token threaded used computed gotos for dispatch. It had a split dictionary, NVM and RAM. This is where I think I finally got the hang of <BUILDS DOES> etc. It's called "micro4th" but I used a 'u' because I don't have a mu handy.

x4th - I had been on Discord talking to Fortherers like Charley Shattuck for a while, and had been trying to osmose cross compilation. So I tried to write a cross compiling forth. It is token threaded and the same code runs on the host or the MCU and it can download compiled code over USB into the MCU's flash. I even added a GUI sort of thing experimentally, as I had been playing with a GUI project at work (Sysview). I decided to put it to sleep after I realized a couple of things. a) I don't actually need all that GUI-fu, I only actually need a few GUI thinks like graphs of variables. b) Things were getting more complicated than I wanted them to be. Saving image files, tracking which MCU they belonged to, etc, was all doable, but I was still building infrastructure and not actually using it for much. This was all experimental, so I called it x4th.

s4th - So I decided to write a 4th variant with a split dictionary, NVM for builtins, and RAM for colon definitions again, but designed only to run on an MCU. The idea was to force myself to use it on MCU instead of on the host. It seemed to work ok, but I found myself running out of code space for the primitives. This was intended to run on my Spanda cards, thus it was called s4th. But I kept running out of code space cramming primitives into it.

h4th - This is a hybrid of x4th and s4th concepts, thus h4th. This has a single dictionary on the host, and only compiled code in the MCU RAM. It limits what the MCU can actually do at runtime, like x4th, as it cannot do anything needing the dictionary. I went back to running on host and MCU so I could debug it at night while working on the cabin with no MCU on hand. I was frustrated with the tedium of splitting things into host vs MCU c files with headers and other nonsense and wanted a single file implementation like s4th. So I used some large ifdef sections, which I may come to regret. I did, like s4th and others before it, use a generator to emit the dictionary, enums, prototypes and other detritus.

a4th - Towards the end of h4th I took a week off to work on the cabin. My subconscious worked on the 4th stuff for me. I had just designed the first Rykor, and was planning the Kaldane. And I was thinking that the Kaldane should be a SAMD21 with 256KB of flash and

32KB of RAM. And I thought, screw it, I am going back to the s4th split dictionary, NVM for builtins, and RAM for colon definitions. And it can run on a SAMD21, dammit. Sure the Rykors can be SAND10's and they can run a slimmed down version of a4th, but the Kaldanes can be SAND21's and run the full thing. I mean, why not? I've done a lot with the SAMD10/11 parts and used their limitations to hone the ZDK. The Rykors are equivalent to the Spanda remotes, and the Kaldanes are equivalent to the Spanda bases. And anyway no one but me will every see or use any of this so I can do whatever I want. I couldn't come up with a good name for this one so I decided to start naming them from a .. z from now on, skipping over any names I've already used.

b4th - Not a thing. Yet.

End