Levels

of code in Forth programming

I had been programming in Fortran and assembler for a number of years when I got my first personal computer. It was exciting. On the first generation PC one had to toggle in a paper tape loader from front panel switches, byte by byte, then start the paper tape reader and then one could output to the led on the front panel. I had waited for the first model that booted from ROM, had a parallel port for keyboard input and had video output from he factory, it was 1974. The PC I got had a wonderful monitor program that replaced a front panel with a nice video display of the contents of all the CPU registers where one could change anything and single-step through programs seeing everything visualized before you. One could also the video for text and graphics display by your programs.

I found it very educational. About a year later I upgraded from 1KB of RAM to 9KB and could now run an assembler or BASIC interpreter/editor on the 8080. There was DOS but I could not afford a disk and used cassette storage. I also found FIG-FORTH about that time. **Since I had assembler training I found that Forth let me do everything I could in assembler and much more and do it much faster and re-use my old code that I had compiled and added to the dictionary in my-language.**

I started with voice and speaker recognition, voice output, and then I added graphics and sound and 3D visualization with interactive simulations of objects and access to the sub-logic/MS 3D databases extensions to FIG-FORTH and sold my first system to NASA. I found porting the whole environment fairly trivial from a cassette based environment on 8080 to a better graphic and sound environment on 6502 with block based disk I/O access. I had been using Forth quite a bit now and starting to learn more about it.

My Forth code was carefully layered. I had actually modified the ROMs on my first machine to default into loading Forth. I had access from the ROMs on up. I could use ROM code from RAM or write my own CODE (assembler) words in Forth to form the lowest level of software above the hardware. I had a layer of code of this type for the registers, memory, and I/O ports. In some cases there was a layer of code to define a serial port or other abstracted device on some parallel port pins using a timer or something. Then there was a layer of code to provide abstracted control over all the abstracted hardware and software devices. A layer for the Forth kernel, a layer for the high level Forth extensions, a layer for my 3D and sound and voice and object, and finally of course any application layer on top of this that I could save as a stand-alone system. This seemed to fit with the evolving ideas in telecommunications standards providing portable levels or layers.

I was deeply influenced by my first exposure to Smalltalk. At that time my idea of an ideal environment was a Forth with Smalltalk-like object, graphic, and GUI extensions. It ultimately wanted to have a source browser assisted the user in tasks like metacompilation making them almost transparent. I did add a layer for a Smalltalk-like GUI toolkit to my Forth dictionary. It only adds a few K at most.

In the eighties I worked in telecommunications and I added appropriate telecom protocol layers into

the system and support for new modems, and a scripting language layer, and of course scripts. I was also researching AI in Forth, implementing ideas from LISP examples and doing expert systems and neural nets and mixing them and building robots. In the software I added a layer for an inference engine for English language descriptions of rule sets and a layer for the rules. I wrote a learning email report and conversation engine AI program and had it running for a few months. **My boss could not distinguish it from me. That was my idea of AI, smart enough to do my job for me and get paid at my salary while I took a vacation.** A couple of years later I purchased a Forth kit with the first Forth chip and got personal friendly support from Charles Moore. My ideal language now needed to have neural and rule based background tasks intelligently cooking knowledge bases. I also studied parallel programming paradigms and worked on parallel programming extensions to Forth. [Fox, 1991] [Fox, 1995] My idealized environment became a simple and fast parallel Forth chip optimized for AI code and multimedia and an integrated version of all my Forth code. [Fox, 1993]

Now to jump ahead ten years, I found myself working with Charles Moore, the inventor of Forth, on our latest custom VLSI. It was be his fifth generation Forth chip, my first. I had worked with several of the earlier generations of chips but not contributed to or owned a license on the design as with the F21.

Working closely with him for a number of years changed most of my ideas about Forth. I thought I knew Forth by this time and was somewhat perplexed by what I saw Chuck do. I still could not get close to the results that he got, either in productivity or quality. I tried very hard to study and understand what he was doing different than the Forth that I had learned. **I recall that the first thing that jumped out at me was that Chuck's code didn't have all those layers that my code had. There was the abstracted code on top and the hardware drivers on the bottom and very little in between.** There was not a clear distinction in style at various levels as in my code where there were always someone else's code layered under mine that I could not or dared not change. Chuck's code looked abstracted and high level right from the bottom, the top used almost as many primitives as the bottom code.

Of course we had optimized the hardware to get an instruction set to minimize the software required to build an optimizing native code compiler for Forth. This new improved Forth virtual machine was now no longer a virtual layer in software, it was a layer of hardware. Chuck put most of his efforts into optimizing the hardware for speed and simplicity after the specification to support the improved virtual machine and simplified optimizing compiler that was about five years old now. This also eliminated the need for an assembler layer, our native code was Forth, Forth was the assembler. We couldn't have it much nicer than this.

I was working mostly with software for the actual chips while Chuck focused on the hardware, and using and refining his CAD software to aid him in this process. I observed that he applied the same style to his work on his PC based CAD software to the software intended for this own chips. I asked him about it and he said that he just ported to Pentium by mapping the ideas without adding any extra layers of abstraction. He said that his chips sort of enforced good Forth style but that on Pentium it takes a little more effort.

To jump forward another seven years, earlier this month Chuck made a comment in an interview about Forth in an Internet chat room. [Moore, 2002] He was talking about the notions of levels in the

ANS Forth CATCH/THROW error handling method when he stated, "**What I do is to mix the low and high-level code into an integrated whole; modify the low-level code as necessary for the application. The notion of levels of code, as in communication protocols, is wrong. There needn't be so much code to make it necessary!**"

It jumped out at me again. His approach is that there is a problem, constrained by hardware, and an ideal solution expressed by brevity, accuracy, reliability. In short quality. It's the ideal solution and nothing but the ideal solution. Chuck's code has no layers that are fixed. He wrote all of it and it is as easy for him to change any part of it as any other part of it. I knew that. No wonder he doesn't need all those levels. No wonder he is so much more productive and writes so much better code. I had written essays about this before, he doesn't use all those layers of abstractions. [Fox, 1998]

Later on in the chat session interview Chuck went on to say, "**Portability is not important. Portability is not possible. Real applications are closely coupled to hardware. Change the platform and all the code changes. If it didn't, you wouldn't have changed the platform. To abstract the problem from the hardware requires massive software like Windows, that's a permanent tax on all applications to save some one-time programming. Programmers should object to job-elimination concepts. Of course, jobs are actually multiplied to deal with the hyper-complex abstraction. And modern hardware has computers in the displays and disks. They've already made many interfaces portable. How many layers of portability are needed?**" [Moore, 2002]

Which brings me to the heart of the essay. Jack Woehr quotes Chuck as saying, "**Forth is more an approach than a specification for a programming language.**" [Woehr, 1992] Chuck said that the reason for Forth was to liberate the programmer, to get them out from behind things that they couldn't change, to let them realize that idealized solution at the cutting edge of the capabilities of any given the hardware, and to maximize their control over everything. [Moore, 1970]

There is something that is a near mirror image of this and completely reversed. The ANS Forth Programming Language specification IS a formal specification for a programming language. Instead of existing to avoid the inclusion of unwanted programming constraints on a solution it formalizes further layers of mechanisms to embrace and mask over these constraints, it adds to them. It mandates adding a few further layers of abstraction in the name 'portability.' It loses any notion that the return stack is actually a return stack. It looses any actual access to registers or even real memory. It looses any notion of how anything works below more layers of abstraction. It loses the real and useful behaviors of I/O devices behind layers of abstraction.

Some authors of large ANS Forth systems for popular environments have said that 99% of their work and code was to deal with problems of the constraints of the other abstracted layers in the systems into which they integrate these ANS Forths. They tell us that these constraints dwarf the standard Forth compilers themselves. This fact alone should account for Chuck's claim that they will write at least ten times as much code as he will write to reach a solution. [Moore, 1999] I think Chuck is just being modest or hasn't paid close attention to just how much code these people have to write. I think the evidence supports my claim that they are likely to write a hundred times as much code as Chuck will write for a given problem specification. Chuck has only claimed that 'C' programmers write a hundred times more code than he will write. [Moore, 1999]

The ANS Forth standard is at least one, maybe two orders of magnitude more complex than Mr. Moore's approach to Forth. He says that code should be so simple that most type of errors simply can't happen. In the late eighties and early nineties Chuck quit writing code in Forth and experimented with sourceless programming. His first versions of his VLSI CAD software, OKAD, were constructed without source using his tools in his OK operating system. Later he return to Forth programming and rewrote OKAD II under his new colorForth. [Moore, 2000] In the chat session Chuck was asked, **"How did you come to the conclusion that Forth was too complex, and that sourceless programming was your next move?"**

His reply was, **"Maybe by reading the Forth Standard."** [Moore, 2002]

References:

[Fox, 1991] Fox, Jeffrey A., Forth-Linda Parallelizing Forth, FORML Conference Proceedings, 1991

[Fox, 1995] Fox, Jeffrey A., Distributed Shared Memory in Forth, Parallel Forth, Internet, 1995

[Fox, 1993] Fox, Jeffrey A. and Montvelishsky, Michael, F21 and F*F, Parallelizing Forth, FORML Conference Proceedings, 1993

[Moore, 2002] Moore, Charles H., Internet Chat, May 5, 2002

[Fox, 1998] Fox, Jeffrey A., Low Fat Computing, Internet, 1998

[Woehr, 1992] Woehr, Jack J., Seeing Forth, page 83, available from Offete Enterprises, San Mateo, CA, 1992

[Moore, 1970] Moore, Charles H. and Leach, Geoffrey C., FORTH - A Language for Interactive Computing (pdf) (html), Amsterdam NY: Mohasco Industries, Inc. (internal pub.) 1970.

[Moore, 1999] Moore, Charles H., 1x Forth, Charles Moore, Interview 4/13/99

[Moore, 1999] Moore, Charles H., Dispelling the User Illusion, SVFIG, 5/22/99

[Moore, 2000] Moore, Charles H., colorforth web site, Internet, 2000

Jeff Fox, 5/29/02