Low Fat Computing

A politically incorrect essay by Jeff Fox 12/6/98.

I have had the pleasure of working directly with Chuck Moore the inventor of the Forth language for the last eight years. Last month in email I described what we are doing as **low fat computing**. I like that expression. It conveys a lot more meaning than calling our computers Minimal Instruction Set Computers.

As Chuck said in the interview that he did for me in June of 93 what he was doing was not just hardware or software but an attempt to create the simplest combination of the two. On the chip that he has been doing for my company, Ultra Technology, almost everything is experimental. To understand F21 one must understand that we spent years refining certain ideas and simplying both the hardware and the software to remove more and more of the fat. For many years I had been practicing another art that requires refining and economizing and using mind and spirit in place of brute force. Chuck's art involves a very similar art of continuously refining his hardware and software by continuously redefining rewriting and refining his own computers and programming language. He tries to solve problems with cleverness rather than brute force.

Forth is an odd programming language. Simply put it is the smallest and simplest programming language I have ever seen. It is simple because its essence is two stacks, one for data and one for return addresses, and a dictionary of WORDS that get defined. A programmer turns the language into an application by extending it to solve a problem. People say that Forth forces people to reinvent the wheel but for many jobs you want to invent the new wheels that you need and Forth is good for that.

We have a tiny computer with Forth in hardware in the architecture of our CPU and instruction set. It is a Forth machine optimized to run Forth. As Phil Koopman says in his excellent online book on the design of stack computers, *STACK COMPUTERS the new wave* while there are problems porting C to stack machines programs written in Prolog, LISP, and OPS-5 are very well suited to stack machines. They are very efficient for neural simulations and real-time expert systems as he explains in section 7.2.3. It has to do with the efficient direct mapping of data structures to simple code on stack machines. Often even Forth programmers miss the whole idea of the hardware and software in our design and ask why we don't have FP hardware. This is because Forth means such different things to different people.

I think of Forth as a manual of how to explore new ideas. It is an approach and a process and an art. To some people it is just a programming language or a standard and to others it is just a tool. Forth is not generally a language that appeals to people who want to march in step with their friends or a big team. Forth is generally speaking a language that appeals to people who want total control over their experiment to explore some new ideas as far as they can be taken.

As a result if you have fifty Forth programmers in a room they will give vastly different reports about their work. I used it to explore space in radio astronomy. I used it to explore the ocean with robotic floaters. I used it to explore new theorums in math. I used it to explore AI. I used it to explore the idea removing the fat from the hardware and the software in a computer. They all say that they use Forth

but they use very different defintions and versions and styles of exploring. What is the true Forth? Is it a particular implementatin of ANS Forth on a particular machine? Is it Chuck's latest Color Forth? Forth is different things to different people.

Chuck has said that Forth was designed to avoid the unsolvable problems in computer science but that an industry will always exist marketing new solutions that won't really work. You just can't solve unsolvable problems. Many people use Forth today to try to find new solutions to those problems rather than using Forth to avoid those problems to solve some real problem. I have done that too and decided that Chuck was right about that.

A Forth community adopted some of his ideas twenty years ago and went off in many different directions than Chuck. They keep debating the same unsolvable problems like the rest of computer scientists. Chuck has been refining his art and simplifying his language, his computers, and his methods. He describes his Color Forth as "**Brutally simple**."

His programming style is thoughtful. He thinks about the problem a lot and writes very little code. He thinks it through again and rewrites the code. He objects to letting too much code accumulate from historic reasons and likes to rewrite. He is the most productive programmer I have ever seen yet he has only written about 15K of code in the last fifteen years. He has rewritten it a number of times. It is quite amazing and has almost no fat. The people who buy or sell six figure VLSI CAD software sometimes get very upset by his ideas of giving away his ideas and tools so that children could play a VLSI CAD game under OK on their workstation in a mouse and simulate and print state of the art chip designs.

Chuck thinks a problem down from the top to the bottom from the word on top that names the application down to the little words at the bottom close to the metal. He very carefully chooses his algorithms for these low level routines. He very carefully scales the quantities in the problems to get the most efficient methods to manipulate the low level data. He very carefully chooses the data structures to optimize the efficiency of the access by the most important routines. Then he implements those data structures and low level routines and tests them and moves up to the top level code. Factoring, scaling, and data structure design all take place using problem specific environmental conditions. He does things at design time and compile time that other people do in their code at runtime. He ends up with very little but very well thought out code that runs very fast.

Other people think of Forth differently and use it to do very different things. I understand that a Forth written in 'C' or C++ can be very portable and is readable to 'C' programmers. Such a Forth can provide a powerful interactive development tool that can easily interface to other programs or OS features in systems dominated by 'C'. Where Chuck would write his own code to do something these people prefer to use the standard interfaces that everyone else uses in those environments.

Forth and 'C' use very different styles. Forth uses stacks for data and lots of factoring. 'C' uses less factoring and data is in variables and structures and stack frames. The 'C' compilers are usually smart enough to juggle the system resources and keep important references in registers rather than in much slower memory. The compiler does a lot at compile time to produce optimized runtime code. Forth prefers to use the stack rather than variables or locals for data. On a stack machine like F21 stack operations take 2ns while variables and locals can take hundreds of nanoseconds depending on the

memory timing.

One of the worst forms of Forth abuse in my opinion is to translate 'C' directly into Forth. The code needs to be designed to use the stacks or it is not really Forth. I can think of an example where an assignment to write a jpeg decode and display routine was given to two programmers. One simply translated the code from a 'C' program into Forth. The second wrote Forth code. The former program had bugs and was 100 times larger and 1000 times slower than the latter. I plan to compare these two programs to the same thing done in other languages and other operating systems on other hardware.

I like to tell people that a Forth compiler can be 25000 times smaller than a 'C' compiler. The essence of Forth is that it is small and simple. Many people in the Forth community want to make Forth more mainstream and want to promote Forth as what I see as just another scripting language for 'C' enviroments.

The comp.lang.forth FAQ: General Information (1 of 7) from 12/2/1998 states that:

3.5. Is Forth faster or smaller than C?

> Not in itself. I.e., if you translate a C program literally into Forth, you will see a slow-down (e.g., a factor 4-8 with Gforth, a threaded-code system; for typical native-code systems you will see a factor of 1-3). Similarly, there is no inherent size advantage in Forth. For details see http://www.complang.tuwien.ac.at/forth/performance.html.

I have always said that I will believe that 'C' is as small as Forth when I see a 1K source and object 'C' compiler. Both Chuck and I have done Forths with 1K of object code and Chuck has done Color Forth with with less than 1K of source. The source I was using was about 25K and only about 1000 times smaller than the source to a 'C' like GCC.

When I say that Forth is 1000 times smaller than 'C' I am talking about a Forth compiler or application with compiler. But even object code produced should be signifigantly smaller. Chuck was talking about application code when he said that Forth code is 100 times smaller than 'C' code and that if it is not it isn't Forth. Forth means different things to different people.

People want to know what would happen if we put the fat back in. The predictable things happen. It slows down, it needs more memory, it costs more, and it gets buggier. We want to take make hardware and sofware 1000 times smaller by taking out the fat. Our idea is not to put Forth on top of the fat hardware and fat software to manage the spiraling bloat.

People ask what would happen if we put the fat back in and ran 1000 times as much code by using Unix and 'C'. I say the whole point was to design hardware and software where we won't have to deal with that. Taking out the fat is how we made it 1000 times smaller and cheaper and simpler and lower power. If we put the fat back in we lose that advantage. They often just don't understand at all and say that they cannot evaluate the design unless we give them Unix specmarks. I tell them that Unix specmarks are them most inappropriate benchmark possible. Those are the standard Unix and 'C' fat overhead benchmarks, they take a simple problem and add Unix fat overhead to simulate typical 'C' and Unix programs. The only thing more inappropriate for us than SpecInt is SpecFP. For many years MISC chips have been left out of various internet chip lists because Unix Specmarks were not

provided.

The way I see it Forth includes Chuck's chips and his methods for designing them. He starts at the top and thinks things through down to below transistor and interconnect level. He focuses on the problem that he is facing including what the code is going to look like and he doesn't spend time solving a problem that he can avoid. He plans out the layout and interconnet and starts building and designing from the bottom up. Of course he has mostly been simulating and debugging for some time. He developed his Color Forth to write simulation scripts. He plugs in some simulated jumpers and probes, loads in some code, simulate various inputs then let it run and print the results. Neither Spice nor Chuck's OKAD equations have modeled the actual behavior of his chips exactly. OKAD has always done a better job and is now very close as he said in his recent Fireside Chat to the FIG

Chuck's custom VLSI CAD software OKAD has only about 15K of code. It includes his OS and GUI, OK, a graphic screen editor, a text editor, a font editor, and OKAD with the ability to view and edit different chip layers, to cut and paste sections, run design rule checks, view actual geometry, and simulate analog and digital and thermal behavior, output cif files, run Color Forth scripts, and other things. The people who do these things with six figure software either get offended that they use 1000 times more code than Chuck and that they didn't write it themselves or that their software can't model Chuck's chips or that I didn't mention that they can write LISP scripts in their editor or whatever.

Chuck didn't just make his code 1000 times smaller than the code that was marketed with lots of fat, he also did it to make his code many times faster. Chuck has setup the problem so that he does the important calculations with very little code. He has setup OKAD to model some transistor behavior in ways where his code can do a couple of shifts and adds that happen very quicly where the standard SPICE approach requires solving several differential equations. Chuck also claims that his code is not just thousands of times faster but also more accurate than SPICE. It is hard to argue with him about it because SPICE gets his circuits wrong and rarely predicts that they will work.

The idea of his chips is to get the fat out of the hardware and the software. Modern PC and workstation CPU chips are very complex. Most of that complexity is there to get high performance but the complexity and costs tend to increase geometrically while performance tends to increase linearly. Much of the complexity is there to deal with the complexity.

At the CPU hardware level PCs are very complicated. They have millions of transistors and burn watts of power. Chip manufacturing costs and power consumption are proportional to the number of transistors. The reason for those millions of transistors is to get performance. Chuck has been trying to find a way to reduce the number of transistors. His first design the Novix was faster than the fastest Intel chip at the time (386) on many real-time applications but only used 4000 gates. He wanted to move from gate array technology to custom VLSI so that he could make the chips fifty times faster and let them be manufactured for a hundred times less.

But our chips have more than a CPU. F21 has a memory controller, a video I/O coprocessor (like a video card), an analog I/O coprocessor (that can be used as a soundcard, modem, cable modem), a serial/network coprocessor, a parallel port and a real time clock on chip in addition to the Forth engine CPU. It is more like a complete PC or workstation sans memory than just a CPU. It is orders of magnitude smaller and cheaper and simpler than a PC.

The first time Chuck talked about his new architecture in public in 1990 people didn't understand. Someone asked if it was a toy. Later he said, "They say it like it is a bad thing." I like to tell people that F21 is a toy workstation on a chip. I like toys. They can be fun and educational. I would like to give toy workstation farms to children. I want some computer chips that make nodes that plug together like Lego blocks.

In low fat computing we don't have to deal with the problem of supporting a dozen different CPU chips, hundreds of BIOSes, thousands of different I/O cards from different manufactures. Much of the complexity has just been removed. We can do a lot with a little and do it very cleverly. When programs get so bloated that they will not fit within megabytes of memory they thrash on a hard disk and slow things down by orders of magnitude.

I have been telling people for years that a GUI can be done in a few K and that it doesn't need to eat many megs or most of the computing power of a computer. I like to point to the 1K GUI in iTV's 4OS. I like to compare it to Microsoft Windows. I have Forths that interface to Microsoft Windows. They are very different than the Forths that we have on our low fat computers. There are ten thousand words in those Forths just to interface to Windows. We replace it with a couple of K of code. We don't have to support dozens of different CPU and hundreds of video cards and thousands of terrible standards inside of the OS or GUI. If we want to present TCP/IP or some other standard to the outside world then that is fine.

People want internet access these days. The best access is presented on expensive computers and interconnects to the internet. It is a commodity experience that will expand to include low cost internet appliances. If you get all the fat out an internet smart appliance is about the same cost as the dumb appliance with the more limited embedded computer in it. The first stage will be TVs and microwaves that are on the internet. Perhaps we will have light bulbs and light switches there eventually.

The analog coprocessor on F21 is another example of a low fat device. To understand the device one really needs a background in A/D and D/A and DSP and other background information. People usually assume that the A/D D/A pair on F21 operate like other chips. They assume that for each sample a count down timer will fire an interrupt and the CPU will run a program to save context, service the device, and restore context. They are used to fat designs. They are used to this approach being the limiting factor for performance. If the CPU runs at 1mip and needs only twenty five CPU instructions to service an analog sample it cannot perform more then 40K samples per second. On F21 a count down timer fires and clocks an analog coprocessor read, write, or write- after-read memory cycle for each sample. At the end of a buffer it can interrupt the CPU to process more data. The overhead has been removed and as a result we can run at 40 megahertz speed on the analog pins like much more expensive analog devices.

It is really fun simulating, designing, building, debugging, and programming low fat computers. We think it is really fun porting real world programs to these low fat computers. It was really fun a couple of years ago helping to port a company's 1.2 megabyte PC application to 16K on a low fat computer. It is really fun working with software that is 1000 times smaller than what other people use. It is really fun working with computer hardware that is 1000 times smaller and lower power and cheaper than what other people have. We really like being thin. When we contrast what we are doing to fat

computers many people get very offended.

Chuck Moore is a perfectionist and will try to take his ideas as far as he can to test them out. He has created such a lean and mean computer architecture that most people don't understand it and may feel insulted by his or my comments about bloated computer hardware and software. Neither of us are very politically correct in our presentations and tend to just say what we think. The first time I heard Chuck give a presentation on our project I listened very carefully from the back to see if he would say anything too insulting to anyone. I felt he had been really good that day. He hadn't mentioned Intel or Microsoft or 'C' once so I felt a sense of relief. I asked this guy in the back what he had heard in Chuck's presentation. He said that Chuck had said that he was an idiot for using 'C'. Chuck had never even mentioned 'C' that day but this guy felt that if what Chuck had said was true then he was an idiot for using 'C' because he was doing only a tiny fraction of what Chuck was doing.

Likewise when I say that "the Emporer is not wearing any clothes" I realize that it does upset many people. The whole idea of our project and low cost low fat computers is very threatening to many people for many reasons. The computer industry is basically a shell game of marketing fat. If you have a job in that industry or if you are a user supporting the industry then low fat computing could seem threatening. I have said many times that the marketing of computers is mostly an issue of style and status. Personal computers are the new barrier between the haves and have nots. For many years while owning a high tech startup company I was always amazed that most of the children that I knew had more expensive computers than I. I am not talking about our low fat computers, I mean that we developed our low fat computers on smaller and older PCs than the ones people buy for their kids as toys.

Don't get me wrong. I think Personal Computers are great! They can do real work and they are great toys. For many years I was a consultant to Bank of America and Pacific Bell and published articles in PC World and ComputerWorld. I saw the way mainframes were marketed to big companies by IBM and later how PCs and PC software was marketed. I just object to the ethics of an entire industry that adds fat to products and markets fat to increase profit margins.

I have done a number of public presentations about our project over the years to the Forth Interest Group, the Parallel Processing Connection, and various robotics groups. I was always amazed by the questions that I was asked. It has taken me a long time to understand why people see us the way they do.

If I am in a room full of computer professionals I think I am in a room full of people who mostly make their living dealing with computer fat. If you ask that room full of people what their companies do what will they say: We sell software to get the accumulated fat out of your system. We sell software to clean up the garbage left behind by your programs. We sell software to deal with the growing complexity of your hardware. We sell software to deal with the growing complexity of your software. We just keep selling bigger upgrades to our product. We sell a bigger CPU. We sell bigger memories. We sell a service solving people's upgrade problems. We sell PCs to people who don't need them using clever marketing. Often for these people quality refers to the profit margin not to the user's computing experience.

I always told people that the thing computers are best at is adding unwanted complexity. If you are not

clever using a computer may make a job more difficult. If you are clever a computer can do a lot of work for you.

Whether you talk about the way IBM got major corporations hooked or how Microsoft got the public hooked on the upgrade path for their computer hardware and software you must admit that it worked. But many people have seen that the conspicuous consumption PC market is the barbed wire of our times setup to keep the undesirables from cyber space. The internet is still a private gated community with only a few public access points. The computer in the home and all the information and access that it provides is one of the things that distingues 20% Americans from the other 80% and America from most of the world. The idea of low cost low fat computers threatens many things here.