Dispelling the User Illusion

Charles Moore
May 22, 1999

to the Silicon Valley Chapter of the Forth Interest Group.

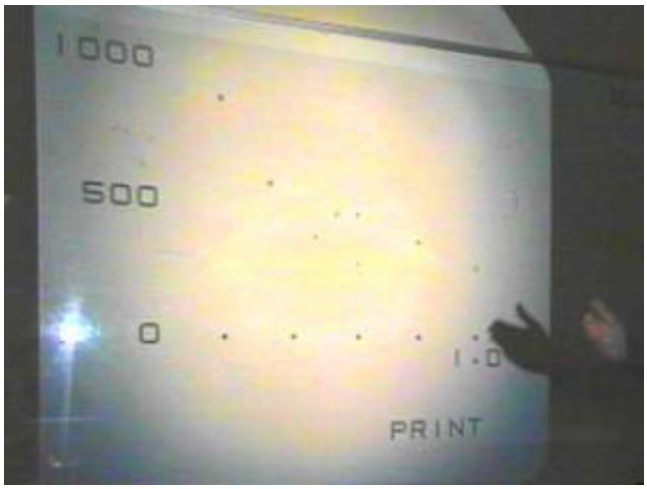
video in store

Charles Moore
I asked a question on comp.lang.forth to see what I should talk about today and I got a, (pause) I got a reaction. (laughter) I will talk about maybe 10% of it. It had to agree with what I was going to already say you see. But I invite you to do the same. I will stop early enough that less than trivial questions can be asked.

To answer one question: iTV is alive and well. Things look very promising who will say what will happen next week. But we are working at it. I am not permitted to say more than that except that, an amuse aside, we had a due diligence review from a silicon valley technical expert. I was looking forward to this because I wanted to see what a real authority would say about what we were doing. It was disastrous, absolutely disastrous. I think the kindest word he said was esthetically elegant and the harshest word was anachronistic. And in a sense he is accurate. Silicon valley is not looking forward to the products that we are making.

Q: Who is this person?
Chuck: I shouldn't say. He is a respected authority. I would welcome a chance to debate him some time. But that will never happen.

```
1000  .
500   . . .
    0 . . . . 1.0
```



Here is an interesting picture. We are talking to a new fab. We burned through two fabs. I don't know if you know this but. But the HP .8 micron is gone and that is what I was using for the last four years. We ported to a .6 micron process at LG last year but that is now gone.

This fab had a lot of different processes so I had a chance to make this picture on the right you see the feature size from .1 to 1 micron. and vertically is the number of micro amps of current per micro meter of transistor width. So this is the current through a transistor. Roughly speaking the speed of a process is going to be proportional to the current through the transistor. That is about the only variable left. Here we are at .8 micron at about 500 and that just happens to be mips too, which is convenient. If we were at one we would be slower if we were at .5 or .6 we would be faster but not by much. This is five volts (in black), this is three volts in red. And when you go to three volts you slow way down because you aren't going to get as much current. This is .35 and this is .25.

I've plotted it offscale too to make it look linear. It is interesting that is mostly linear on each line (5V and 3V). This is an IBM process up here at .2 (u) and 1.5V and that to me is enormously impressive. Given that these are the marginal gains that you get at 3V to get way up there at 1.5. I would like to work with that process. But that is only a factor of 2 in speed from where we are now. Which is a little bit disappointing. I think it may be more than that, maybe 2.5 or 3. You can't know until you try. But these are going to be 50% faster than we are now but not a factor of two. (pointing to .25 and .35) And at a fairly considerable cost in going to smaller geometries in cost, dollars. The smaller geometries cost you more than larger ones. That is why we were sitting at .8 for so long because it was a nice mature comfortable process low cost fabrication. we can't sit there, this is obsolete there is no particular reason to move to there unless we run out of speed.

One of the criticisms of our process is that we were claiming these ridiculous speeds. But everybody in the world has agreed that Drystone mips as the proper measure of processor speed... Now am I wrong is saying that Drystone mips measures 'C' compiler performance? That has nothing to do with processors. That is a weird comment to make.

These speeds are, I think, honest. I have been claiming, I have been telling you for years, that this is burst rate (pointing to 500 mips) for a series of four instructions and then we have to wait for memory access.

Well I spent some time building and designing an SDRAM interface. SDRAM will give me 10ns access to a significant amount of memory. Something that was long overdue. I've got to redesign the memory interface and the interface to the coprocessors to take advantage of that. And when I do, and that is another months work, if it is funded, I'll have a sustained 500 mips.

If we were to go to .35 I would have a burst of 600 mips but I can't sustain it. I certainly can't sustain 1000 mips because memory isn't fast enough any more. If I get 4 instructions per word, you take whatever your memory access rate is and you divide that by four and that is the maximum speed that can be sustained. I have always felt comfortable having a clock speed two or three times as fast as I needed so I would have something to do with it. SDRAM may do that.

All right let's go on to what people really want to hear about. This was one of the most popular questions.

```
ANS FORTH
 COMPLEXIFY
 INTERNALS
   WORD STATE HOLD
 DOUBLES
   2@ 2!
 CREDIBLE
 FOCUS
```

ANS Forth is an increasingly significant fact of life in the world as we know it. It may not to be to my taste but I think it is to lots of people's and I have no quarrel with it. What is it that I don't like about ANS Forth?

Well Skip (Carter) showed you a whole list of people working on ANS Forth and I figure I don't need to be one more of them. So I can do something different. That's really all I want to do, to try something no one else has explored. But if I were to use ANS Forth, or try to become compatible with it, I've got some objections. One is the whole concept of a "standards" document. **If you read the document it gives you the impression that Forth is more complicated than it is.** Or else it corrects my misinterpretation of how complicated Forth is.

I like to quote Goedel on this, it goes back to 1930. he proved that a self contained formal system cannot be contained. horribly non- sequator but, all right within a system you can't describe itself. Within forth you can't describe Forth. Well, our systems are not so esoteric as Goedel's but I think you can say that within a system you cannot describe it simply. Now the standard is not inside of Forth, it is outside of Forth. It is looking at it and trying to describe it from a meta standpoint. But even so when you have to explain what each term means accurately like; What is a cell? What is a character? And how does it fit in with all these other things? It gets ominously complex. Particularly so since we all know what it means anyway. It is an exercise in concise writing rather than an effort to convey meaning and I object to that aspect of the standard and there is nothing that you can do about it except that **I'm rewriting the standard to suite my taste and I hope it will be much simpler and it will convey much of the same information and some that was not previously available.**

Another thing I don't like is that the internals are exposed. There is far too much discussion of words that an application programmer will never use. Particularly if that application programmer doesn't decide to write their own interpreter. I don't like people writing their own inpreters because that means that as a user I have to learn a different interpreter. Forth has a perfectly good interpreter if you will just accept the word order. So words like **WORD STATE** and **HOLD** and another dozen or two, not that they shouldn't be in the standard, **they shouldn't be in the CORE wordset. They should be de-emphasized by the standard or there should be another standard appropriate for application programmers and this one is for systems programmers.**

And then the double words. **Get rid of all the double words.** We don't need to do 64 bit arithmetic. Machines now are 32 bits or at least 20 bits and in no case are double precision arithmetic relevant. And it it complexifies things horribly. If you look at these arithmetic operators: **UM*/MOD** or something, I mean what in God's name does that do and why should I care? And you shouldn't. It can be gone. **U/MOD** is too complex. **/** certainly. ***/** yes. **MOD** maybe, but that is about the end of it.

That includes words like **2OVER 2SWAP 2@** and **2! You oughtn't use such words.** They make your stack more complicated. They are making presumptions about the layout of data in memory which you probably shouldn't be doing. It is just too complicated.

If you want an extension wordset with all these double arithmetic operators fine. If you want an extension wordset with all the internal necessary to write your own interpreter fine, but not the CORE wordset.

The good things about the standard are: yes, it has creditability. A lot of people will consider forth who wouldn't have considered it without the ans standard. So a good thing. Another thing, it focuses our attention as the Forth community on issues. Instead of arguing about what Forth is you can argue about what **POSTPONE** means. That at least is talking about something that has some hope of resolution or coveys some meaningful debate among the community. so I do like those aspects of the standard. If the standard didn't exist I couldn't be here bitching about it. (laughter)

(question from John Carpenter about **2@** and **2!**)

Then you should call them **D@** and **D!**, and that they should be and extension. Not that these words are not useful or not nice, but Standard? **The standard is too much common practice and I think it needs a little more pruning.**

All right here are two other Forths. I like to argue that they are not the same. I don't intend Color Forth to be Machine Forth although in a sense it kind of is.

```
MACHINE FORTH
  ADDRESS REG
  @+  !+
  SDRAM

COLOR FORTH
  F1-F5
```

```
EDITOR!
ICE
4X3 STACK
JUMP TABLE

.BMP
```

One of the questions about Machine Forth had to do with the address register. So I though I would explain why there is an address register. And yes it is not very Forth-like.

It goes back to Novix days. I don't know if you remember the Novix. I got an email question about it recently and I didn't remember the Novix either. But when you did a fetch the problem was that the address was on the top of the stack where the data had to go. so the address was driving the address lines and data comes in on the data lines and where do you put it?

Well what the Novix did was it put it under the top of the stack and then it threw away the address on top of the stack. Optionally if you were doing and incremented fetch it put in on top of the stack and incremented the address that was on the top of the stack and everybody was happy. But implementing that in silicon was a mess You have address lines going to the wrong place, the second element on the stack. You have data lines going the wrong way. You have address lines coming to the top of the stack which is equally inconvenient. You need a second cycle to fix up the stack after the operation is completed. To eliminate that it makes sense to move the address out of the way to a special place out of the way and then you clear up the stack to receive the data. Now this really wasn't my idea. It was Russell Fish's idea in ShBoom. I had lots of quarrels with Russell but he did have some good ideas and I appreciate them.

So given that the address register was kind of the least inconveniently way of handling addresses I kept it for P21. It does let you do fetch plus (@+) and store plus (!+). It makes it easy to do, almost free.

I would like to recommend that you use those words in your Forth. It is easy to define a fetch plus. If you do you will find it has a significant effect on your programming style. Your keeping and address off stack. It offers the most efficient addressing mechanism that I know of for sequential addresses.

The alternative is something like **DO LOOP** with **I @ (fetch)** inside which is a very expensive sequence of words. That is a hard way to something equivalent to incrementing an address. This is much more efficient.

I would like to see it moved into the standard. To superseded some of these old fashion slow ways of doing things. The standard is documenting Forth as it was 20 years ago. Let's move a little faster than that.

In SDRAM this kind of fetch plus becomes even more important as an efficient way of getting at memory. Everyone knows SDRAM I imagine. You can get up to 256 sequential accesses at 10ns intervals. If you break that sequence of fetches it will cost 20-30ns hit, so sequential accesses are

cheap.

In the case of F21 you could do **@+ @+ @+ @+** in one word streaming your fetches onto the stack. No interference from instruction fetches elsewhere. SDRAM also has two blocks of DRAM so you could Ping-Pong back and forth without having to read a page each time. It is still a hit, but it isn't as big of a hit. So you can have a data page and an instruction page. My new interface will optimize that but if you don't know it is happening you can't take advantage of it.

Now Color Forth. Most of the work I have done on Color Forth is on the PC because I have never gotten any nice software on the i21. I am too busy getting the hardware to work to worry about software. It is as if I have a threshold, a container of interest. And when I fill up my container for i21 with hardware and there is no where for the software to go except to the PC.

Nothing that I am doing will have any difficulty in porting to the i21. The fact that i21 is 20 bits wide and modern computers are 32 bits wide is totally irrelevant, is it a non-problem. In the case of the software that iTV has developed I think only one or two places in TCP/IP where inherent 32bitness of TCP/IP data was a factor. a 20 bit chip is capable of doing almost anything that a 32 bit chip can do. You can even pack two ten bit characters into a twenty bit word. Three six bit bytes.

How do I Edit in Color Forth? Yes I do have a custom editor for doing this. I think everyone should have a custom editor for editing their program files. I don't believe in universal editors. An editor is so easy too write that why not?

A browser is very easy to write too, and I know people don't like to believe me but I will give you some examples in the course of the next year.

How do I edit color? I assign function keys. I've got six space bars. I hit the right space bar and I get the right color. I will label the function keys will color dots as soon I find my color dots. Meanwhile it's green, red, and blue is one, two, and four. The keyboard entry, the command line the same thing. you hit the color space that you want to.

Why is the space before the word? I would make sense that you accept the word and then terminating character is a postfix command telling you want to do with that word. I thought long and hard about making it work that way. The alternative is to have the space in front of the word which makes it much easier for the editor or to format the word correctly. And it turns out that I was right. Because not only is the space but the space after the word also. SO this way I have two spaces, in fact there are three spaces. There is the current color, the color for the current word, the next work, and the previous word. I require all three of those in order to do the right thing in the interpreter.

So it finally came to me that what I've got is a 4x3 stack, I've got a 3 entry stack of 4 bits to describe each of these states. So I have a little push down loop in which I multiply the previous value by 16 and add in the next value. And I've got the three in one variable in the order that I need them and I can peel of the one I want by shifting. This is much better than having named variables for each one or a little array in memory of individual words storing 4 bit values. So I am pleased about the invention of the four bit stack.

Given what I have determined what I am going to do with a word I go through a jump table to actually

do it. I know the preceding space, I know the word. I go through a jump table to determine whether I am going to convert that word to a number or whether I am going to look it up in a dictionary or whether I am going to throw it away or whatever. And one of these destinations in the jump table takes me to another jump table where I take the next word, the next color, and go through another jump table to decide what to do with this first word that I've got. I'll show you some examples. It's a very efficient process, much more complicated than the old fashion state variable compile or execute.

I would like to propose not a new concept but certainly a new emphasis to an old concept. I like to call it ICE. It has a nice cold connotation. It stands for, Interpret, can you guess? Compile, and Execute.

ICE

INTERPRET
COMPILE
EXECUTE

And this is what makes Forth different than 'C'. I think this one of the unique capabilites of Forth. In addition to the two stacks, in addition to high factoring, the fact that you can move back and forth in the three behaviors of words is very important and can lead to very compact code on the order of 1% of the kind of code you get if you don't have these three behaviors.

So I want to give you an example. I've always wanted to document the behavior or OKAD and I've had a lot of difficulty doing that because no matter how many words I write down it helps if you can look at a picture. So I wanted to be able to transfer portions of the OKAD displays into HTML files. And I was going to construct GIF files and Mark Smeder pointed out that I had already constructed BMP files and that I could use those and avoid the royaltee. They are not as compact but compact doesn't really matter. So I wrote an application for createing .BMP files and I will show that to you now. I hope you won't be disappointed.

It is 4 of my 256 byte screens. Here is a bit of Color Forth.

```
 FRAME EMPTY VARIABL
E BUF W H : ROW A! 1
69 BEGIN @+ IF + ; T
HEN DROP NEXT 0 ; RO
WS DUP BUF ! ROW IF
DROP DROP ; THEN DRO
P -1 H  +! DUP A + R
OWS ; 448 H ! VGA 0
OVER ROWS -1280 SWAP
 640 447 * + ROWS
```

First I am starting with a 640x480 screen full of stuff. I have a little inverter in the middle of that

screen and I want to convert that inverter to a BMP file. But first I want to throw away all the margins. So that what this word **FRAME** implies. I am throwing away the frame on the image. I appologize because this code does not work with an emtpy screen, it will crash. But someday I will fix it and meanwhile I promise not to construct and empty BMP file.

EMPTY throwing away everything else. So what I have got is a minimalist Forth system underneith here. Essentially nothing more than Machine Forth is available to me at this point and time. I'm defining some variables I ended up needing three: **BUF W** and **H**. Width, Height, and an address to the buffer where I am constructing an image.

The first thing I do is define the word **ROW** which is going to scan accress a row of the image checking to see if it is empty. And if it is it returns a zero apparently if it falls out of this **BEGIN NEXT** loop. It uses fetch plus. It isn't very long, it is rather fast. It goes through 159, which is 160 4 byte fetches looking for a non-zero in the course of doing this. So I am taking advantage of the fact that the image is um; oh I forgot one tiny thing: this is a vga image packed in a horrible way, first I have a little machine language word that goes out and takes this and stores this in bytes. So I really have a byte image here. That cheat will go away as soon as I get to 256 colors instead of 16. This is the way it will be.

Given the word **ROW** I define the word **ROWS** to scan rows until I come to something non-zero and that extends to here. **ROWS** repeats, it's just another loop. These counted loops are a real pain. I don't like counted loops but in a case like this this is the way you have to do it. So I have compiled this stuff then I switch to interpreter and reference it. I set **H** to 448. I call **VGA** to construct my image. **VGA** tells me where to put it. I call **ROWS** to come down from the top until I find something non-zero. I call **ROWS** to come up from the bottom until I find something non-zero and then I go on. Guess what happens next?

```
COL A! H @ -1 + BEG
IN @+ FF AND IF + ;
THEN DROP A -544 + A
! NEXT 0 ; COLS DUP
 BUF ! COL IF DROP ;
 THEN DROP -1 W +! D
UP A + COLS ; 640 W
! BUF @ H @ 640 * -1
 + OVER 639 + COLS 2
 + SWAP COLS DROP W
@ 1 + -2 AND W !
```

The final word **COL** and **COLS** to do the same thing. Then I switch to interpret and I use **COL** and **COLS** to bracket the image other way. And I end up with **BUF** marking the beginning of the image, **H** containing the height, and **W** the width.

But there are some interesting things happening. First up here in **COL** I want to know the height of

the column that I have got to examine. That is stored in **H**. So I fetch the value of **H** and I subtract 1 from it and I pass that as the controling argument for my **BEGIN NEXT** loop. And this is executed at compile time becauses it's black. And when it switches from black to green it is compiled as a literal. So this is an expression that is evaluated. Here is an example of **BUF**. Now **BUF** is a variable but a black **BUF** is executed at compile time so here I am getting an inline literal representing the address of the buffer instead of a subroutine call to the code for variables. This is faster, a little bit bigger.

Here again in **COLS** I want to subtract 1 from the width. -1 **W** plus-store. -1 is a literal this reference to **W** is black so it gets executed at compile time. As an inlined literal again just like **BUF**. Unfortunately there is a little problem here. If this -1 had a white (black) space following it then it would be left on the stack. I don't want it to be left on the stack. I want it compiled as a literal. So this is a green space here and then there is a white (black) space to be the prefix for **W**. Now it's a little bit obscure on the listing but if you move a cursor through here you will see the cursor change colors so it will look alright.

Q: about Chuck's cursor being the ring character. Chuck: I was thinking about that. We need a secret decoder ring. (laughter)

OK, so there is the code for framing the images that I am after. Now here is where we get into the BMP file. Here is the nice blue **BMP** and all the nice hex digits. This is the boiler plate necessary to build up the header for the BMP file.

```
 BMP BUF @ B71000 BU
F ! , 4 N, SWAP BUF
@ ! BUF +1 ; 2, 2 N,
 ; BM 4D42 2, W @ H
@ 2 */ 16 4 * + 54 +
 , 0 , 118 , 40 , W
 @ , H @ , 1 2, 4 2,
0 , W @ H @ 2 */ , 0
 , 0 , 0 , 0 , ORGB
FFFFFF , FF00 , FF ,
FFFFF , E00000 , E0C
000 , FFFF00 ,

 808000 , 408080 , 4
0F040 , 40FC , E000C
0 , E00040 , C0FFFF
, 404040 , FCFCFC ,
PACK DUP 1 256 */ SW
AP 16 * OR ; ROW W @
 2/ -1 + BEGIN @+ PA
CK 1 N, A -2 + A! NE
```

```
XT ; ROWS A! H @ -1
+ BEGIN ROW A W @ -
 639 + + A! NEXT ; R
OWS END
```

I have redefined the word comma to put down 32 bits constants as they want. Then I use the comma also to put down a 32 bit color lookup table appropriate for this image. And it slopped over into the next screen. Then low and behold I define the words **ROW** and **ROWS** again to go scan the image and put out the pixels. And I have defined the word **PACK** to pack the data 2 pixels per byte as in inverted order if they want it. And then finally I reference the word **ROWS** and I'm done. I've got the file in a buffer. I know where the buffer is. I can go back DOS and run a little program that writes it out to disk.

I would argue that these two screens which is all that is all that is involved in BMP is a pretty small amount of code to write a BMP file. The equivalent 'C' in the Supercharged Bit Graphics that I looked at that told me how to do this was two pages of 'C' I guess. And of course that had lots of comments and I don't have very many comments. I do have **BMP** at the top though and that counts.

I also make the claim that given any application that you want I can do it in about this much code that will do it. If I want an editor I will have an editor in a handful of these screens. If you want a protocol stack OK I will do some more. There are some considerations. I don't have all this stuff compiled in memory at the same time. I don't have a system which does everything. All I have is enough code to generate the code that I need.

I would do this with any application. I used to this at Forth Inc. If you wanted to do a report from your database you compiled the code to do the report at the time that you requested the report. It didn't take any time. It didn't require any name conflict resolution. It gave you a small self-contained description of the report. It was a very efficient way of writing software.

And I think that this has gotten lost in these mega-Forth systems. They have lost the ability to compile at runtime. All you can do is execute. They have lost the ability to do some things in interpret mode in the course of compiling.

Interpretting is more efficient than executing. In order to compile anything you have to interpret it first. If you are going to interpret it you may as well execute it and be done with it instead of putting it down and coming back and looking at it again at execute time.

The side effect of doing this is that you end up with a system that consists of a large number of small applications instead of a large system will all the applications somehow embedded in it. I think that is the cause of a lot of trouble. It goes back to the comments I made about the standard. The standard has everything built into the **CORE** wordset if it were factored a little bit better it would be easier to understand and provide better versitility.

I have a list of more questions people have asked. But let me ask you for some questions first before I go into those. (no questions) You are not prepared?

Question about iTV's status and funding.

(something about deals are in progress.) Negotiations are delicately proceeding. One consequence is that we have not had the funding to make any chips for the last four of five months so there have not been any chips made.

I have done some work. I have added features. I came up with a whole new chip that is the best of all chips. But at the moment I don't know if that chip is relevant. I don't know what other chips might be relevant. And I am sure that someone is going to come to me someday and say "This is what we want." And my reaction will be, "But this is different than anything you've ever said."

I am kind of waiting. I am in the document mode. I have got some very pretty HTML screens gates in it. Showing off the various kinds of gates the system uses. There is a bug in it. I can display this fine with I.E. but my color inkjet printer won't print it with the right colors. I showed you the color table there. Those are the colors and I know they are right because I displayed them on the screen. But the printer won't print them that way. I don't know who is responsible for putting out the colors. Is it the printer driver or Windows? But someone is dropping the ball.

I will show you that sometime. It will be on a website sometime. It is very pretty. And it is much easier to understand silicon design when you can see it rather than when you have to read the netlist.

We are still at our old site, the second site, in Redwood City.

Dr. Ting ask questions about a USMC foundry and the shuttle program and how they are looking for customers.

That sounds to me very attractive. I mentioned it to Gary Langford and he says that USMC has a very active legal department defending themselves against intellectual property claims. He doesn't want to go there. He is afraid of losing whatever proprietaryness we have in our design.

I am more sanguine about this is an invulnerable design. You can't replicate it without the design tools. But he doesn't want to bet the company on that. If that turns out to be the only alternative it is the way we will go.

We are talking to some Israelis. The Israelis are not known for honoring intellectual property either so I don't understand the distinction that he is making. These kind of choices tend not to be economic but political. If we are dealing with a company that is dealing with a company that knows somebody in Israel then that is where we will go. We have few choices. I've never had a good choice. First there was ORBIT. They were the only one I knew about and they were cheap. Then there was Mosis with HP. And now Mosis has three or four others that we could pick. But it is a 5 to 10 thousand dollar shot and with that kind of money you can pay somebody for a month for what it costs to do a fab. That is what we are doing.

John Rible: Have you had any time to make OKAD work in a hierarchical fashion?

Chuck: No.

Q: What do you mean by hierarchical?

Chuck: The chip layout is flat. All the latches are laid out in the registers and t here is no way I change the layout of a latch and have it propagate everywhere there are latches. But I am not strongly motivated to do that. It might have been a better way to start. But it is very important that I have the flat layout in order for the simulator to simulate. So I have to flatten everything anyway.

Given that to make it hierarchical is more work not less. There are a number of changes I want to make to OKAD but it is not clear that there will ever be a convenient time to do that. You start down a path past the branchpoints and where do you end? More importantly, I would like to get the a .. John has suggested changing the tiles from the 16 different patterns of corners and crosses and things to about 4 different patterns of lines and corners and that is higher on my list than a hierarchical layout.

Dr. Ting asks a question about million gate FPGA availability and doing chips in FPGA. "At least you have something that runs at 1/10 the design speed."

John Rible answers, "You may as well use the software simulator because the only thing that that will give you in this environment is the logical correctness. It won't give you any other information."

Chuck says The timing would be completely different and some of the tradeoffs would be different. If I were just starting out I think that would be a very interesting path to follow, things get complicated. But given when I am now I don't need any practice. I've got a simulator that I'm convinced now tells me true. And if nobody wants to make production quantities of these chips I've got other things to do. I'm not going to beat a dead horse.

My goal at the moment is not to design another chip, it is to sell some chips. Selling some chips means your doing a million, if you are not doing a million it is not worth doing. The economics of the industry have changed in the last ten years. More and more reliance on large production runs. And you get great concessions from the manufactures if they anticipate a large production run. And no one is interested in 10,000 parts, it is not worth anyone's trouble; designing or manufacturing. If you want 100 parts then yes, FPGA, but there is still a big gap between 100 parts and 10,000 parts that is completely empty.

Q: What are you going to do with Color Forth? Chuck: I suspect that I will publish it on a website somewhere as soon as I get it where I like it. It will be very small. 1K, 2K something like that. It will boot from a floppy or CD-ROM, if you give me your whole hard disk I can boot from hard disk. I doubt if I can live with a partition. Which is why I say this is whole nother opportunity for old computers to come back to life.

Chuck: Good question. What would I do if I didn't have to work for a living? I would do exactly what I am doing now. I am very happy with this. I wish we could get into production. If we get into production there is a disturbing consequence. If we get a chip into production it will be the chip we used last year. It will not be my best chip because it has no history. How can you prove the value of a new chip when nobody will let you make one because they are busy making the old chip that you know doesn't work? And with large production, with the pressures of production, I think that a huge amount of innovation will get lost. This is the conundrum, but even so I'm tired of innovating. I want to see some production.

Color Forth looks to me to be a lot of fun.

I will try to re-address these issues of my wild irresponsible claims as to how small software can be. But I will repeat them lest anyone misunderstand.

I do not think it is intrinsic in the nature of software that it has to be large, bulky, buggy. It is in the nature of our society. We now have millions of programmers who are relying on their paychecks and they have to keep busy. Y2K is a boon for the software industry. You can drag old retired programmers back and put them to work. There is so much economic motivation for producing this ... bloatware, that I kind of feel irresponsible in standing up and saying the emperor has no clothes.

But at least in my small way I will accumulate a library of examples to prove the point to anyone's reasonable satisfaction. I don't expect it will sweep the industry.

There is one way that it could. That is if a small group of people got together and decided to challenge Microsoft. We could replicate their software in 1/10 of 1% of the code. We could feasibly do that with a small team in a year or two. But again why destroy an industry? That is probably not a wise thing to do.

Jet Thomas: I don't think you could do that. To do it effectively you would have to replicate their bugs.

Chuck: Oh no, I'm not running applications. I'm replacing the whole shmear.

Jet: Oh, you could do that.

Chuck: That's much more practical.

Dr. Ting: The Chinese are doing that. They are very much worried about Microsoft controlling every aspect of the industry so they are trying to develop their own OS.

Chuck: You said the evil word. If they are starting from the OS they have made the first mistake. The OS isn't going to fit on a floppy disk and boot in ten seconds.

Actually the greatest problem to having a Forth come up nicely with a picture on the screen before you blink is the damn BIOS. It powers up in BIOS it doesn't power up on the floppy and it takes you ten seconds to get to the floppy. So you have this delay. But I'll give you the PROMs that you can plug into the sockets that will do that. None of this is hard.

There was a book, it was translated from the Danish, written about 1989. I forget the author but the name was, The User Illusion. This should ring a bell for everyone. The User Illusion is what Apple exploited to make the computer look different than it really is. To make it friendlier, easier to use. So the user has the illusion that he is dealing with the software when really he is dealing with the hardware. He viewed this as a good thing. Among many of his good comments this was a misunderstanding I think.

I would like to dispel the user illusion and make the interface transparent and attached to the hardware.

I repeat the claim that I made long ago that it is easier to write a floppy disk driver in machine

language than it is to interface to the BIOS routines. And this is true of every device.

All the devices together, now that is a different problem. But you are not dealing with all the devices together. I am not going to give you every printer driver. I am going to give you the HP inkjet printer driver and if you want a different one you will have the knowledge and tools to build one.

Q: Is Color Forth going to have the same colors as the iMac? (laughter)

Chuck: I have been thinking. comp.lang.forth has been talking about a Forth logo. If you follow that, it kind of got voted down, an anime Jedi warrior girl, scantily clad. But that got me thinking.

What we need is a uniform! (laughter) How much more impressive it would be for me to stand up here looking at you if you all had your cadet uniforms on. I think it is pretty clear that color should be that butternut color that monitors are made of so you could blend in with your computer system. You could have white pipeing on the ANS Forth programmers and red and green pipeing on the Color Forth programmers so that you could tell at a glance as to the essential information about your audience and their level of sophistication. And these would be compatible with Color Forth.

John Rible: FIG Forth programmers could have silver and gray pipeing. (laughter)

Chuck: I wore this tee shirt to get in the uniform mood. But I noticed dearth of tee shirts. Is anyone selling Forth tee shirts?

George Perry: You are the second one to ask.

Chuck: Are we missing a whole marketing opportunity? Let's have a logo contest and everyone will have to produce a prototype tee shirt and wear it next time. If we have a Forth logo then we can have a Forth flag. There is a whole world out there anxious to buy stuff. (laughter)

I do have several talks full of questions backed up that I will not have time to do to here. I did have another thought. The advantages of Forth in a team programming environment. We have spoken the fact that a small group of Forth programmers can do an application that would otherwise take many more and that there are efficiencies of inverse scale at work here. So it a good team approach.

There is another advantage that I haven't heard mentioned, they can do it faster. Time is often a concern in getting out applications. And if you want to minimize the time it takes to do an application move in a team of Forth experts, hopefully they have Forth on your platform already, blitz the problem working twelve hours a day for two weeks and go off with it finished. This is a custom programming model that Forth Inc. has used very effectively and could be applied in a much wider range.

When you are working for a company doing an application it can become a career. And heaven forbid that you finish the application your hanging on the end of loose limb. And that is again a cultural motivation. It may not be the way it ought to work but that is the way the world does work. Moving in a team of a Forth commandos. (question) It depends on your rank. (laughter)

Bob Barr ask a question about efficiency. The natural order isn't so efficient. How efficient is an apple

tree or society?

Chuck: Yes, I agree that economic motives are important but they are not the only ones. There are issues of personal satisfaction which ought to be at least as important as the economic value of what you are doing.

This code that I showed you. I worked on it two or three times as long as I ought to have in order to get it right. I am never going to execute it often enough to get a pay back but it feels good. And that is important especially in this increasingly corporate oriented world where people go to work to earn a living instead of seeking the job satisfaction. I don't know that we should be satisfied with a world full of mediocre programmers. Increasingly our culture is depending on computers in an important way as Y2K is pointing out, perhaps there is room for some professionals to do it right. I don't know how to sell that.

Question about Y2K.

Chuck: I think the answer to the Y2K problem is not the patches that they are doing now. I gather that what they are doing is windowing it so you will have another problem in the future. The answer is to make the specialty embedded microprocessor self-documenting. So that you can walk up to one and ask it what is your take on this particular problem. And the way you do that is with source code instead of binary. And the way you do that is that you code it in Forth and compile it at runtime. And then you know that the program that you are running is the program that the source described and that it works that the way you expect. Especially if you are going to look at that program fifty years from now for the first time and try to figure out what it does.

I think if the Y2K comes out badly there will be regulations about what you can do. (groans) We will just have to get political influence to mandate Forth.

John Carpenter asks a question.
Chuck: Unfortunately this margin is too small.

Question about ADA being demandated.
Chuck: I hear that ever since they demandated ADA that it is more popular than ever. But the pressure will be on compactness because you don't want to store any more in ROM than you have to.

George Perry calls for a hand.
Dr. Ting calls for plans for coffee at Fry's or dinner.
(end of talk)

UltraTechnology
2512 10th St.
Berkeley, CA 94710-2520