eForth as an Arduino Sketch

Last year I decided to retire from electronics and microcontrollers, because after two glaucoma and cataracts operations, I could not see small objects and narrow lines and there was no way that I could work on the surface mounted parts with very narrow line spacing. So I cleaned out my study and my garage, gave away all my tools and spare parts. I realized that I should not be a hardware engineer. I am only a programmer, and should just work on software.

Then when I visited my brother in Denver last summer, I saw that my niece was working on a couple of Arduino Boards. On an Arduino board, there was a microcontroller in a DIP socket! That was very interesting. When I came back, I bought a couple of Arduino Uno Boards, and have been working on them since. I had to buy back tools and many electronic parts and ate my vow to stay away from hardware.

Arduino Uno is a lovely, small, cheap, and readily accessible microcontroller board. The operating system and the programming environment Arduino 0022 is a good match to the Arduino Uno Board. Through a single USB cable, you can upload programs from a PC to Arduino Uno, and then communicate with Uno through the same cable using RS232 protocol. You write programs in C language as sketches in Arduino 0022, and the sketches are compiled and then uploaded to the ATmega328P microcontroller on Arduino Uno for execution. Sketches are C programs greatly simplified to the point that you just have to fill lines of code in the two following routines:

```
 setup()  loop()
```

All intricacies and complications in the C language and its associated compiler and linker are taking care of by Arduino 0022 system. No wonder Arduino is such a huge success.

FORTH is a programming language much better suited for microcontrollers than C. FORTH is really a programming language with a built-in operating system. It has an interpreter and a compiler so that you can write programs in small modules and interactively test and debug them. You can build large applications quickly and debug them thoroughly. FORTH also gives you access to all the hardware components in the microcontroller and all the IO devices connected to the microcontroller.

So, I ported a very simple FORTH model, 328eForth, over to the ATmega328P microcontroller. It was written in AVR assembly language, and had to be assembled in the AVR Studio 4 IDE from Atmel Corp, and then uploaded to ATmega328P through a separated AVRISP mkll programming cable. Once 328eForth is uploaded to ATmega328P, it can communicate with the PC through the Arduino USB cable. 328eForth cannot be uploaded through the USB cable, because Arduino 0022 requires a bootloader pre-loaded in ATmega328P to upload sketches, and 328eForth must use the bootloader section of flash memory in ATmega328P to store commands which writes new code into the application section of the flash memory at run-time.

For the serious FORTH programmers, 328eForth system give you the ultimate control over the ATmega328P microcontroller. For the much larger Arduino user community, we need a FORTH implementation which is compatible with the Arduino 0022 system. Here is my solution: ceForth_328. It is written in C as a sketch. It can be compiled
and uploaded by Arduino 0022. Once it is uploaded to the Atmega328P microcontroller, it communicates with the PC through the Arduino USB cable. However, new FORTH commands are compiled only into the RAM memory in ATmega328P. You have only about 1.5 KB of RAM memory to stored new commands, and when you turn off Arduino Uno, these new commands are lost.

In spite of these limitations, ceForth_328 is still a very useful system. You can learn FORTH and use if to evaluate Arduino Uno for various applications. You can also use it to learn about the ATmega328P microcontroller, because it allows you to read and to write all the IO registers. I specifically added two commands `PEEK` and `POKE` to read and write RAM memory, which includes all the IO registers. The AVR Family Data Book is a huge 566 page documents, and the best way to read it is opening one chapter on an I/O device, reading the register descriptions, using `PEEK` to look at a register, and using `POKE` the change the register.