

Forth Works with Embedded Systems

Jon Titus, Senior Technical Editor



In late 2011 I wrote about the GA144 "multi-computer" integrated circuit from GreenArrays that provides 144 cores on one chip. Engineers and programmers use the company's arrayForth tools to create programs that take advantage of parallel and pipelined processors. Yes, you need to use Forth, which many programmers disdain. But the chip seems so powerful that Forth deserves another look. To find out more, I interviewed Ron Oliver, a consulting engineer at FORTH, Inc.

Q: Why should engineers take a close look at Forth as a language for embedded systems?

A: A Forth cross-compiler can remove many of the headaches associated with embedded C compilers. I work with SwiftX, an incremental compiler. It doesn't need a linker, so you don't deal with a linker configuration file and its syntax and conventions. Here's a code snippet from a SwiftX target configuration.

```
00008000 00017FFF CDATA SECTION PROG \ Code space in flash  
00018000 000187FF IDATA SECTION IRAM \ Initialized data
```

```
00020000 EQU |RAM| \ Total size allocated
```

The Forth words CDATA and IDATA select the current section type. The SECTION defining word creates PROG and IRAM. I can access these section limits while I compile target code without having to export symbols from the linker and pass them to the compiler.

Forth compiles incrementally, so you can compile the debugged part of a program, download it to the target, and leave it unmodified. Then you recompile only the small portion of code under test and reload it. This type of programming lets us maintain the fast turnaround time that Forth programmers know and love, and that C programmers should know and love.

Q: What causes a newcomer the most headaches when they start to learn Forth?

A: First, the stack, because Forth is a stack-based language and the math is like

Forth Works with Embedded Systems

Published on Electronic Component News (<http://www.ecnmag.com>)

that used in a reverse-Polish-notation (RPN) calculator. It takes time to get used to doing $(2 + 3) * 5$ as: $2 3 + 5 *$.

But anyone who can write decent C can handle it. Complicated math in Forth is harder to read than the corresponding math in C. But you will find math operations easier to debug in Forth.

Second, the "word" nature of Forth. New Forth programmers, for example, might look at this line of code:

```
00020000 EQU |RAM|
```

and ask, "Do the vertical bars give an absolute value for RAM?" The names for Forth words require one or more non-blank characters separated by one or more blanks. That's it. So the five characters, `|RAM|`, form a name, although someone else might have chosen `SIZEOF_RAM` instead. Forth programmers frequently use key characters to indicate operations, such as `@ADC` to indicate this word will "fetch" the ADC value.

A third hurdle involves getting programmers to take advantage of the interactive Forth environment. They've never had a way to create software in a fast iterative cycle; code a little, test a little, and do it again. As soon as you have Forth running on a target system you can explore its hardware without writing a single line of code. Reach in and read those I/O lines and memory-mapped status registers using `C@` and `@`. Dump memory to the display using `DUMP` (an ANSI standard word). Modify the memory with `C!` and `!`. Then write a little code and use it.

Q: Can you give an example of stack use in Forth?

A: Given variables `foo` and `bar`, for example, you'll probably find it easier to understand:

```
foo = bar;
```

than

```
bar @ foo !
```

Unlike most programming languages, Forth isn't based on expressions, but rather uses words that serve as functions, constants, variables, and other data structures. As I noted before, Forth uses a stack to hold and pass parameters between words. This "parameter stack" operates independent of the processor stack, or "return stack," programmers associate with other languages. So, here's how the Forth code above operates:

```
bar  
| put the address of variable bar on the stack.
```

```
@
```

Forth Works with Embedded Systems

Published on Electronic Component News (<http://www.ecnmag.com>)

\ @ replaced the address of bar on the stack with the value of bar.

foo

\ put the address for foo on top of stack, with bar's value beneath it.

!

\ store bar's value into foo. Stack is now back to its starting condition.

This stack architecture lets Forth programmers "run" words such as those above from the interpreter and mix and match words interactively to solve a problem. In Forth, the intermediate result remains on the stack and stays there until explicitly removed.

Q: Ron, you've talked about "words" in Forth. Can you explain that concept in more detail?

A: You create a new Forth word - what other languages call a function or procedure--with a "colon definition" in the interpreter. In turn, the interpreter uses the Forth compiler to add new code to the Forth "dictionary;" the list of words available. I can define a word called `set_foo` for the Forth dictionary:

```
: set_foo bar @ foo ! ;
```

And I use it like any other Forth word.

Q: Can you give a practical example of how to use these words in Forth?

A: Say you have a digital-to-analog converter (DAC) at address `DAC0` that drives a motor. You could use a Forth interpreter to start a test this way:

```
10 DAC0 !      \ Start driving the DAC with a value of 10  
10 DAC0 +!    \ add 10 to the DAC value  
10 DAC0 +!    \ and add another 10
```

Suppose I want to press the keyboard spacebar to automatically add 10 to the DAC value as I take measurements. I create a new Forth word `STEP`:

```
: STEP BEGIN KEY BL = ( space?) IF 10 DAC0 +! ELSE ( return to interpreter) QUIT  
THEN ;
```

(The Forth "=" compares values the same way as the "==" operator in C. Forth lets you put comments in parentheses and because the first parentheses also represents a Forth word for "start of a comment," a space must follow it.)

Now I can just run `STEP` and press the spacebar to increase the `DAC0` value by 10. Any other pressed key aborts the test.

Q: Can you give me a few comparisons of strengths and weaknesses in C/C++ and Forth?

A: A great strength of C is the large number of programmers who know it. Everyone programs in C because everyone else programs in C. Conversely, Forth's greatest weakness is the small number of programmers who know it, which makes Forth difficult to get established even for engineers who know it and want to use it. Managers ask, "Who else can program in Forth?" and you have to answer honestly, "Not nearly as many as know C."

To an experienced Forth programmer, Forth isn't just a language; it's a software development and testing environment. In my opinion, the Forth language without the Forth interpreter is pointless. You use Forth to experiment with things interactively, to try things out, and to tie things together. Then you write code that automates what you were doing in the interpreter. The proper Forth software development process closely duplicates how a good Unix system administrator develops complex shell scripts: write a little, test a little, and repeat the process to complete a task. Forth makes it fast and easy to interact with the target system.

Forth's extensibility is noteworthy too. Forth lets programmers control and extend the compiler in ways C programmers can only dream of. You can build complex data structures and tables in Forth more easily than in C.

Q: How do programmers learn more and try Forth?

A: Go to the SwiftForth web site (www.forth.com/swiftoforth/dl.html [1]) and download a free evaluation copy of the software. Then go to the online document, "Starting FORTH," by Leo Brodie (www.forth.com/starting-forth/ [2]), and read it while you use a Forth interpreter. After you go through "Starting FORTH," the jump to SwiftX proves surprisingly easy.

Q: What tools are available to cross compile Forth for embedded systems?

A: TheSwiftX cross compiler from FORTH (www.forth.com [3]) is my choice for Forth programming. MicroProcessor Engineering (www.mpeforth.com [4]) also has a Forth cross compiler. There are many Forth systems out there for various microprocessors and microcontrollers.

Q: Does Forth work well with real-time operating systems? And does Forth support the types of "libraries" found in C/C++?

A: Typically, SwiftX provides the language and real-time operating system. And people usually buy SwiftX with a development board onto which they can install SwiftX and get to work right away. SwiftX provides a simple but effective cooperative "multitasker," small interrupt service routines handle the real-time requirements while the multitasker handles things that can wait a few milliseconds.

From time to time we've had to interface to vendor-provided C libraries. Because Forth has a dual-stack architecture, we assemble C stack frames to make the function calls and get the results. An experienced embedded developer can get it

Forth Works with Embedded Systems

Published on Electronic Component News (<http://www.ecnmag.com>)

done without too much grief.

About Ron Oliver: Ron earned a BSE in Engineering Science from the University of Michigan in 1988. For fun he learned Forth in 1983, working on a friend's single-board 68000-based computer. That experience helped him get a job as a control systems engineer, writing software in Forth, various assemblers, and C for satellite-antenna control systems. Since 2001 he has worked as a consultant in embedded systems programming.

Source URL (retrieved on 11/27/2014 - 7:29am):

<http://www.ecnmag.com/articles/2012/01/forth-works-embedded-systems>

Links:

- [1] <http://www.forth.com/swiftoforth/dl.html>
- [2] <http://www.forth.com/starting-forth/>
- [3] <http://www.forth.com>
- [4] <http://www.mpeforth.com>