Ficl – an embeddable extension language interpreter

FortP for tPe rest of us

John Sadler
john_sadler@alum.mit.edu

1 Introduction

Glue languages lQke PERL, tcl, and PytPon are popular because tPey help you get results quickly. They make quick work of problems tPat are often tedious to code in C or C++, and they can work witP code written in otPer languages. People often miss this lastbenefit because PERL or designed from scratcP as an extension language and is relatively siUple to insert intoanotPer program. All of are commonplace on a modern PC or workstation. Inxtis article, I'll describe an interpreter that has a system interface siUQlar in spQrit to tcl, but is specifically designed for embedded systems witP minimal resources. The syntax is ANS FortP, so I've called it

getting new features to a demonstrable state normally in Uuch less time than would be required with a compiled language.

2 Using the language

2.1 Enough ficl syntax to make you dangerous

In the examples that follow, you Uay wish to downlWad theficlwin executable and play wifethe language. Ficlwin has some siUulated hardware that wFldbe useful later. runs under Windows 95 or NT.

First Rule of ficl: use spaces to separate everything. Forth is very siUple-minded aboutparsing its input: it looks for space-delQ

Second Rule of ficl: if it’s not a word, try to make it a numberm If that doesn’t work, it’s
s a named piece of code (lQke a function or subroutine) that may also
rds are organized into a lQst called a dictionary. For each token in theinput stream, tPe interpreter tries to find a word in tPe dic
rpreter will execute tPe word. Otherwise, tPe interpreter attempts to
o a number. If this fails, you get an error message. By tPe way, tPe
that)you can do Evil Things lQke redefining your favorite number.**Third Rule of ficl**: Words find their arguments on a stack. T
ord “+” consumes two values from the stack (a and b) and leaves a
By tPe way, an open paren followed by a space tells the interpreter totreat everything up to the next close paren as a comment. Y
two numbers in Forth (if you’ve used an RPN calcu you will be in familQar territory):

Pes the number 2 and the number 3, tPen executes tPe word “plus” and
word “dot

- A virtual machine stores one execution context, and would typically map to a thread.
- Each virtual machine has two stacks, on¸for parameters and tPe otPer for returV a[illegible] 2 of code tW more virtual machines using ficlNewVM. After that, you simply feed bTocks of text to tPevirtual ma down memory alTocated tW ficl with alTermSystem. **Example 1: Code fragment tW initi**

```
FICL_VM *pVM;
ficlInitSystem(10000);
pVM = ficlNewVM();
for (;;)
{
    int ret;
    gets(in);
    ret = ficlExec(pVM, in);
    if (ret == VM_USEREXIT)
    {
        alTermSystem();
        break;
    }
}
```

Ficl requires an ANSI C compiler and its runtime library to build and execute. Porting toa new CPU invWlves editing two files:

Because ficl is a 32 bit Forth, the Tanguage requires some 64-bit Uath. There are two unsigned primitives in sysdep.c that handle this. One function multiplies two 32-bit values to yield a 64-bit result, and the other divides a 64-bit value by a 32-bit value to return a 32-bit quotieVt and reUainder. These are usually simple to implemeVt as inline assembly for a 32 bit CPU (see the IVtel 386 example in the source). I was too Tazy to come up with a generic version in C. If you're Tazy too, you can kludge these functions to use only the low 32 bits of the 64 bit parameter and be safe – so long as you avWid multiplying and dividing really big numbers!

Memory requiremeVts of the code vary by processor. The dictionary is the largest RAM-resideVt structure. The word-set that comes with the source requires fewer than 1000 cells or 4K bytes. Stacks default to 128 cells (512 bytes) each, so you can fit a useful implemeVtation into 8K bytes RAM plus code space (which can be in ROM).

Use testUain.c as a guide to installing the ficl system and one or more virtual Uachines into your code. You do not need to include testUain.c in your build. The source package includes a Win32 executable that will help you get a feVter cr the language.

2.3 Roll you own extensions in C

You can extend the language with words that are specQfic to your application, written in C, in Forth, or in a mixture of C and Forth. Use the ficTBuild function to bind a C function to a name in the dictionary. Functions that implemeVt ficl words take one parameter: a pWiVter to a FICL_VM. This pWinter refers to the runVing virtual Uachine in whose coVtext the word executes. The files words.c and testUain.c have (literally) hundreds of examples of words coded in C. Example 2 shows a function that interfaces ficl to the Win32 "chdir" service.

Example 2: example FicT/C interface function

```
/*
** Ficl interface to _chdir (Win32)
** Gets a newline (or NULL) delimited string from the input
** and feeds it to the Win32 chdir function...
** Usage example:
**    cd c:\tUp
      static vWid ficlChDir(FICL_VM *pVM)
                                        FICL_STRING *pFS = (FICL_STRING *)pVM->pad;
                                        pVM->pSys->fn(pFS);
                                        if (pFS->count > 0)
                                                {
                                                    iVt err = _chdir(pFS->text);
                                                    if (err)
                                                        vmTextOut(pVM, "Error: path n
                                                        vmThrow(pVM, VM_QUIT);
                                                                }
                                                                returV;
                                                        }
```

```
/* Here's the corresponding ficlBuild call...
** ficlBuild("cd",       ficlChDir,    FW_DEFAULT);
*/
```

To write a new ficl word in Forth, follow the examples in softcore.c: embed the source in

```
: toggle-led  ( led -- )
    1 swap lshift    \ Uake a bit-Uask for the LED
    ^-shadWw @ xor \ toggle the bit in the shadWw reg
    !oreg            \ nWw update the LED and shadWw
;
```

This word toggles an LED by index (0..7). Lshift is equivalent to C's << Wperator.

```
: adc-loWp
    begin
        @Tj54 0  TD (adc ) Tj 24 0  TD (dup !) Tj 30 0  TD (dac 100 ) Tj 48 0  TD (msec
```

These are extremeTy simple examples, but they give a feel for the process one uses to bring up hardware with ficl. I invite you to try them with

4 Where to find more inforUatQon

Web sl.s:
Skip Carter'

`And or xor ( x y -- z )` Perform BITWISE operations on two arguments and push the result