Test Suite

Dennis Ruffer

February 13, 2009

Contents

Chapter 1

Test Suite Historical Background

1.1 Why would you want to use this stuff?

Over the past 30 years I have simplified my debugging tools down to one word that I call a 'fence post'. For many years, it looked like this <?>, which almost looks like the top of a fence post, but since the < and > characters are not part of the colorForth character set, I changed it to **.?.** The word itself has taken on many forms, but in the simplest form, it is defined as:

```
: .?. ( n -- )   cr .s drop ." .?. " ;
```

This allows me to sprinkle these words around the code that I am debugging to 'fence in' the problem. The number that it takes allows me to distinguish between all of them, since I have had cases where quite a few (hundreds) are needed to isolate a problem. This has served me well over the years and I have brought them into everything I work on in Forth.

However, bringing this tool into my work in VentureForth proved to be a little more difficult. First, the lack of resources within the chip requires that the tool be implemented within the simulation environment. Second, the tool must record the position within the target code without modifying the location or size of the target code itself. This proved to be a very difficult task.

Additionally, it was desirable to bring in some of the Agile techniques into our development process. Test Driven Development (TDD) allows you to define your development goals in terms of Unit Tests and to know when you have accomplished each goal. Continuous Integration (CI) allows you to maintain those tests in a process that can be run whenever the code is changed. Letting you know quickly that the goals you have accomplished can still be relied on. The only difficulty was implementing these techniques within VentureForth.

I started by reading Kent Beck's 'Test-Driven Development by Example' and implementing his concepts in Forth. It is recommended reading for anyone

interested in how it works. Then came the job of integrating this into the VentureForth simulator. This has been a problem from the beginning, as I have already mentioned, and it is not perfect today, but it has proven to be good enough to work with. We integrated simulations of our applications with Zutubi's Pulse and every check in to our source repository generated an email to let me know that our tests passed.

This got us reliable target code, but when we started putting it on actual target hardware, our efforts were still frustrated by problems. First, the targets didn't always work right. We started with FPGA simulations of the SEAforth chips, which failed unpredictably as we pushed them faster and added more nodes. When we received the 1st prototype chips, this problem became even more extreme. Second, the target delivery mechanisms were not always reliable as we developed various techniques.

To help us distinguish between target and algorithm problems, I developed a means for the target to tell us if the compiled code was delivered reliably and, in effect, to tell us if the target itself was working reliably. I generated a checksum at compile time, checked it in the target and delivered the results to a feedback mechanism. On the FPGA, we had the entire address space of each node to work with and an LED display to report the results on. Now, as I ported this code onto the FORTHdrive, I came to appreciate what a luxury this was.

The example application that is included with this plug-in shows the cost of this code. The basic checksum algorithm takes 30the result passing takes up to 12but here it takes 4 nodes to get the detailed results back to the host and provide visual feedback on an LED. That puts this technique way over the limit of acceptable overhead. It gives me a 'warm and fuzzy' feeling that my chip is working, but I'm not sure if it has any other purpose.

The same can be said for all of the tools in this plug-in. They are familiar to me, so I continue to use them. However, modern TDD is not a common way of thinking for most Forth programmers, and CI doesn't make a lot of sense in the 'Lone Wolf' environment of many Forth shops. Still, I am publishing these tools with the hope that others will find a way to use them. When used appropriately, they can be very beneficial.

Chapter 2

Installing the vf/Plugins/TestSuite.zip Package

2.1 Installing VentureForth Tools

Someday, we may have a plugin architecture like Eclipse.
http://www.eclipse.org
http://www.eclipse.org/articles/Article-Update/keeping-up-to-date.html

2.1.1 Until then, we can use Eclipse as a model

- They have many plugins, as we can hope to someday.
- They originally used a zip model for updates

2.1.2 Place the Plugins zip file

Into the root of your VentureForth folder.

2.1.3 Extract the contents

If you double click the file in OS X

You will get a directory containing the contents, which you will need to integrate into the T18 directories.

Better to use unzip in the Terminal

The *unzip* command will integrate the files, as needed, and create the necessary directories for you.

2.2 Installed Files

```
projects
        FdCheck
                BlinkLED.vf
                Echo.vf
                FdCheck.vf
                FloorPlan.pdf
                USBCom.f
                check-pass.vf
                check-receive.vf
                check-report.vf
                checksum.vf
                project.bat
                project.vfp
vf
  Plugins
          LaTeX.f
          Plugins.f
          Projects.f
          TestSuite.f
          comments.f
          doc
              TestSuite.pdf
          idForth.f
          links.f
          numbers.f
          strings.f
          testSim.f
          xUnit.f
  gforth.fs
```

2.3 Validating the Tools

Each file in the Plugins directory contains a version number, which has been modeled after the work done in The Forth Foundation Library by Dick van Oudheusden at:

http://freshmeat.net/projects/ffl/

This gives each file the ability to make sure that it is only loaded once, and dependent applications the ability to check that their dependencies contain the features they require. The version number is the file name, replacing the extension with 'version', and is incremented each time the file is changed.

The file can then load its dependencies, checking that they are, at least, as new as when the Plugin was written. The file can also skip loading itself if

its version number already exists. This creates a self-validating, reentrant load sequence.

If any file is older than required, the load process will abort, with the following error message:

Tool is older than required. Reinstall!

2.4 File Dependencies

projects
- FdCheck
  - BlinkLED.vf
  - Echo.vf
  - USBCom.f
  - check-pass.vf
  - check-receive.vf
  - check-report.vf
  - checksum.vf

vf
- Plugins
  - FloorPlan.f
  - Graphics
    - pdf
      - FloorPlan.f
      - pdf.f
    - svg
      - FloorPlan.f
      - svg.f
  - Plugins.f
  - Projects.f
  - TestSuite.f
  - comments.f
  - idForth.f
  - links.f
  - numbers.f
  - strings.f
  - testSim.f
  - xUnit.f
- c7Dr03
  - centerpause.vf
  - cornerpause.vf
  - e4bitsy.vf
  - e4stack.vf
  - leftpause.vf
  - mult.vf
  - rombios.vf

romconfig.f
serial.vf
sget.vf
smult.vf
spi.vf
uppause.vf

Chapter 3

Project Artifacts

3.1 Where did my files go?

Most VentureForth applications will need to save data in files. Typically, an application will have, at least, a file that contains a memory image that can be down loaded into a SEAforth chip. Additionally, there may be other data files that are generated by an application, such as:

- Floor Plans
- Timing Diagrams
- Test Suite Logs
- etc.

All of these files can be considered to be artifacts, produced by an application. Yet, while the memory images fit well in the **../mem** folder, these others do not really belong there. Thus, unless told otherwise, they can be found in a **../data** folder.

Alternatively, if you choose to use the **/projectFolder** command, all of your artifacts will be placed into a folder that matches the current file name (minus the extension) in the current directory. You can further segregate your files into sub-directories, like the Floor Plans do with its title pages.

None of these alternatives, however, create the directories themselves. The user must do this manually, before the directories can be used.

Chapter 4

Test Suite User Interface

The following words are the user interface for the Test Suite. The 1st two words are used within parenthesis, so they do not need to be removed if this plug-in is not loaded.

`n .?.` Puts a 'fence post' at this location in the target image that will display the data stack when the simulator steps over that location.

`assert: ... ;` Puts a test at this location in the target image that will be executed when the simulator steps over that location. The words that are compiled within this definition use the host search order and must return a true or a false flag to indicate test success or failure. Two entries are placed into the TestSuite.log file for each of these tests; one for the setup of the test and the other for the run of the test code. There is no need for a tear down step with these tests. Both of these words attempt to match the compile time address and slot number with the run time locations that the simulator will encounter. This is not always possible and you may have to move it to a different location before the matching logic works properly.

`TestAssertions` is the word that attempts to match the compile time location of each of the above words with the current location in the simulator. This works best when it is executed within the simulator's step logic. However, that requires a change to the definition of step that is described when testSim is loaded and the change is not found. Someday, someone might be able to figure out why, but I have not been bothered by this modification. I have made the code work without it, but you will likely only be able to trigger your tests at the beginning of slot 0.

`simUnit: name-XX-XXX` is not used in parenthesis, but is useful to give a name to the entries placed in the TestSuite.log file. The test setup phase replaces the X's with the node number and the address/slot position. The Test-Suite.log file is not cleared automatically, but can be deleted manually to prevent it from becoming too large.

`n goes` runs the simulator until **FinishTests** has been executed (within an assert test) n times or a key is pressed.

`continue` resumes running the simulator after a key has been pressed.

`FinishTests` decrements the test counter that was passed to goes.

`autoTerminate` aborts if the test **errorCount** is greater than 0. Normally, **TestAssertions** aborts as soon as if finds an error, but you can comment out that line and the error count will be accumulated until **FinishTests** is executed. Then you would need to abort so your compiler exits, like gforth normally does.

`summary` displays how many tests have run and how many failed.

Chapter 5

Test Suite Example

```
v.VF +include" Plugins/testSim.f"  testSim.version 2 checkPlugin

-warning
[defined] TestCase [IF] [ simUnit: checksum-XX-XXX ( -- method ) ] [THEN]
+warning

    checksum ( 1 .?. )

    ( assert: t @ 0= ; )

   | ( assert: FinishTests ; )

cr .( Running simulation testing... ) cr
1 goes decimal cr summary cr
```

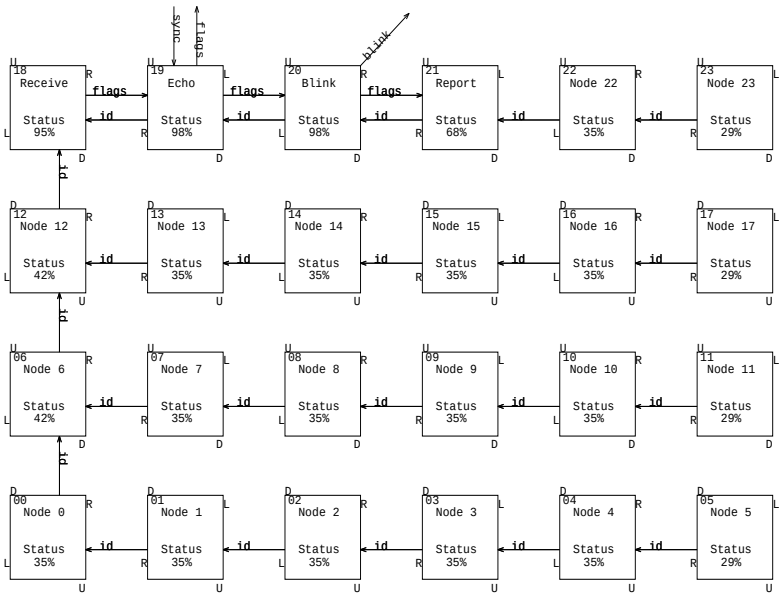

Figure 5.1: *The example application's Floor Plan*

Chapter 6

Test Suite Output

```
FdCheck
Using gforth version 0.7.0-20081226

Compiling ROM 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Compiling RAM 18 19 20 21 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 22 23
Creating library file: /.gforth/libcc-tmp/.libs/gforth_c_7FCD340C.dll.a

Using FORTHdrive PhysicalDrive1
000 LAL8 3FFFF
001 ALD8 0003F
002 ALAK 00000
003 ALAK 00000
Running simulation testing...

     0 15555 15555 15555 15555 15555 15555 15555     0    17 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0    11 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     B 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     5 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     D 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     7 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     E 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     1 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     8 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     F 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0    16 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     0 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     2 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     9 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0    10 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     3 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     A 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     4 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     C 1 .?.
     0 15555 15555 15555 15555 15555 15555 15555     0     6 1 .?.
```

```
    0 15555 15555 15555 15555 15555 15555 15555     0    15 1 .?.
    0 15555 15555 15555 15555 15555 15555 15555     0    12 1 .?.
    0 15555 15555 15555 15555 15555 15555 15555     0    14 1 .?.
    0     0 15555 15555 15555 15555 15555 1F9FF     0    13 1 .?.
15555 15555 15555 15555 15555 15555     0    12     C     C 2 .?.
    C 15555 15555 15555 15555 15555     0    12     D     C 2 .?.
    D 15555 15555 15555 15555 15555     0    12     E     C 2 .?.
    E 15555 15555 15555 15555 15555     0    12     F     C 2 .?.
    F 15555 15555 15555 15555 15555     0    12    10     C 2 .?.
   10 15555 15555 15555 15555 15555     0    12    11     C 2 .?.
   11 15555 15555 15555 15555 15555     0    12     6     C 2 .?.
    6 15555 15555 15555 15555 15555     0    12     7     C 2 .?.
    7 15555 15555 15555 15555 15555     0    12     8     C 2 .?.
    8 15555 15555 15555 15555 15555     0    12     9     C 2 .?.
    9 15555 15555 15555 15555 15555     0    12     A     C 2 .?.
    A 15555 15555 15555 15555 15555     0    12     B     C 2 .?.
    B 15555 15555 15555 15555 15555     0    12     0     C 2 .?.
    0 15555 15555 15555 15555 15555     0    12     1     C 2 .?.
    1 15555 15555 15555 15555 15555     0    12     2     C 2 .?.
    2 15555 15555 15555 15555 15555     0    12     3     C 2 .?.
    3 15555 15555 15555 15555 15555     0    12     4     C 2 .?.
    4 15555 15555 15555 15555 15555     0    12     5     C 2 .?.
    5 15555 15555 15555 15555 15555     0    12    13     C 2 .?.
    1 15555 15555 15555 15555 15555     0    12    14     C 2 .?.
    2 15555 15555 15555 15555 15555     0    12    15     C 2 .?.
    3 15555 15555 15555 15555 15555     0    12    16     C 2 .?.
    4 15555 15555 15555 15555 15555     0    12    17     C 2 .?.
   12     5 15555 15555 15555 15555 15555     0    12     C 2 .?.
    0     0     5 15555 15555 15555 15555 15555     0     0 3 .?.
    0     0     5 15555 15555 15555 15555 15555     0     0 3 .?.
    0     0     5 15555 15555 15555 15555 15555     0    3F 3 .?.
    0     0     5 15555 15555 15555 15555 15555     0 3FFFF 3 .?.
25 run, 0 failed
```