Timing Diagrams

Dennis Ruffer

February 21, 2009

Contents

Chapter 1

Timing Diagrams Historical Background

1.1 The problem it is designed to solve

Normally, timing diagrams are documented as part of a design specification, which is then used as a guideline to meet the inter node communication timing requirements, while developing the multiprocessor program code. The problem with that approach is the actual hardware timing is unknown, so the programmer has to use trial and error techniques to close in on the actual hardware timing that will execute the code correctly, which is a very time consuming process for debugging.

1.2 What it is and how it does it

The inventive method is that the application code itself includes functions that determine the inter node timing, as it executes. The code does this by first capturing data from an event driven simulator that is expected to represent accurate timing information for the hardware. Then the code generates timing diagrams from that data, which are used by an engineer to compare and analyze the code behavior as it executes in the target multiprocessor array hardware. The engineer uses this method to determine if the actual hardware events for a given instruction sequence correlates to the expected events that were simulated. This is a big advantage to reducing debug time, because this method allows the developer to have visibilty of actual timing internal to the chip, which is otherwise not accessible. Also, it is anticipated that the developer will use a standard technique of placing 'dummy' code in nodes while doing design and analysis to see timing in advance, as a part of the design step. That is a novel use of the simulator/chip combination to produce documentation, rather than

just hand drawing these sorts of diagrams as is normally done. [1]

[1] Invention Disclosure 07-0050 by Dave Dalglish 30 Aug 07

Chapter 2

Installing the vf/Plugins/Timing.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 VentureForth 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.htm
  - FloorPlan.pdf
  - FloorPlan.svg
  - Timing.htm
  - Timing.pdf
  - Timing.svg
  - USBCom.f
  - check-pass.vf
  - check-receive.vf
  - check-report.vf
  - checksum.vf
  - project.bat
  - project.vfp

vf
- Plugins
  - Events.f
  - File.fth
  - File
    - Examples
      - Accounts.dbf
      - Accounts.fth
      - Customers.dbf
      - Customers.fth
      - Glossary.dbf
      - Glossary.fth
      - People.dbf
      - People.fth
      - Personnel.dbf
      - Personnel.fth
      - Wines.dbf
      - Wines.fth
    - Index.fth
    - Memory.fth
    - Reports.fth
    - Sort.fth
    - Struct.fth
    - Support.fth
    - csvParser.fth
  - FileNames.f

```
        Graphics
                pdf
                    Timing.f
                    pdf.f
                svg
                    Timing.f
                    svg.f
        LaTeX.f
        Plugins.f
        Projects.f
        Timing.f
        doc
            Timing.pdf
            pfDatabase.pdf
        idForth.f
        links.f
        notail.f
        numbers.f
        strings.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
        Events.f
        File.fth
        File
            Index.fth
            Memory.fth
            Reports.fth
            Sort.fth
            Struct.fth
            Support.fth
            csvParser.fth
        FloorPlan.f
        Graphics
                pdf
                    FloorPlan.f
                    Timing.f
                    pdf.f
                svg
                    FloorPlan.f
                    Timing.f
                    svg.f
        Plugins.f
        Projects.f
        TestSuite.f
        Timing.f
        comments.f
        idForth.f
        links.f
        notail.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

Timing Diagrams User Interface

The only words that are needed to generate Timing diagrams are the words **svgTiming** and **pdfTiming**, which create SVG or PDF diagrams, respectively. However, you do have to set up the VentureForth simulator so that it will save the timing events that you want to see on the diagram. See the Events Help section for those details.

You also may want to change the order of the nodes in the Timing diagram. The value of **NodeOrder** defaults to the the address of an array that containes a sequential list of node number bytes, terminated with a -1. If you want a different order or to only display selected nodes, you can set **NodeOrder** to your own array of node numbers. Remember to always terminate the list with a -1.

4.1 Events Help

```
This plugin requires a hook in VentureForth's port logic like this:
variable 'Event   0 'Event !
: !Event   'Event @ ?dup if  execute  then ;

: !lcl   local !  !Event ;   ( don't ask, don't tell)
...
: /iocs   /clk commit ( n - n')
   adrs @ psel 2@ on? if  1+  !Event
...
```

The Events plug in extends the VentureForth simulator to record timing events in a memory resident database. The database details are not required to use this plug in, but you can see the pfDatabase Help section for documentation.

The commands that you do need to know involve routines that control when data is stored into the Events database and running the VentureForth simulator until some condition is met. Both the storing of event data and the display of simulator information are time consuming, which you may want to avoid when running the simulator. The following commands give you control over those features:

`/Events` initializes the Events database.

`n goes` runs the simulator until n 'finishes' or a key is pressed.

`continue` continues running the simulator after a key is pressed.

`FinishTests drop` an assertion to decrement 'finishes'.

`saveEvents drop` an assertion to store events.

`ignoreEvents drop` an assertion to ignore events.

`lo hi timedEvents` start storing events when the simluation clock is greater than or equal to lo and stop the simulation when the clock is greater than or equal to hi.

The 'assertion' commands are designed to work with the Test Suite plug in, but can be used independently by dropping the flag that they return. Assertions provide more options for controling the simulator, but it is typically sufficient to just do the following:

```
0 10000 timedEvents  1 goes
```

4.2 pFDatabase Help

This plug in is an implementation of the polyFORTH Data Base Support package converted to run on ANS Forth systems. It is documented separately in a 1275 Binding for a Database Package but that binding was never actually published. It is the last time that I worked on the documentation with Elizabeth Rather with the hope to make this tool public, but the effort died when Apple switched to Intel. So, we are free to try again.

While this could be called an ANS Forth Application, you are not left with an ANS Forth System after it loads, due to many naming conflicts. So, I have placed it into a **files-wordlist** that is made current by the word **files**. Thus, all of the naming conflicts, and most of the application usages are buried away from normal user interaction. Only those words which are considered to be application interface words should be exposed to the user. The developer, however, will need to execute **files** before the words in this plug in can be used.

While many application databases are kept entirely in application specific locations, some must be sorted before they can be used. This sorting uses a temporary disk file called TEMP.DBF in the data folder which is a peer to the

folder that your application is loaded from. You will generated an error if this folder does not exist. Plug ins typically create this folder for you, but you may need to create it yourself if your environment does not make this possible.

Chapter 5

Timing Diagrams Example

The following lines are the portions of the FdCheck application that are required to use these features.

```
v.VF +include" Plugins/Timing.f"  Timing.version 4 checkPlugin

: checksum ( -- sum id )     \ assumes it can trash the a pointer
    [defined] saveEvents [if] ( assert: saveEvents ; ) [then]
    dup dup xor dup

[defined] FloorPlan.version [IF]  svgFloorPlan pdfFloorPlan
[THEN] [defined] Events.version [IF]
    /Events \ saveEvents drop \ uncomment to see load
[THEN]  power

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

[defined] Timing.version [IF]
    svgTiming pdfTiming cr
[THEN]
```

The figure that follows shows the resulting Timing diagram. Note that transfers that are written to the right are colored green with the left half of an X appearing at both ends of the line. Transfers to the left use blue and the right half of the X. If you look closely, you will see a double X where node 19 writes to IOCS (actually twice). Using the zoom capability of your viewer is essential with these diagrams.

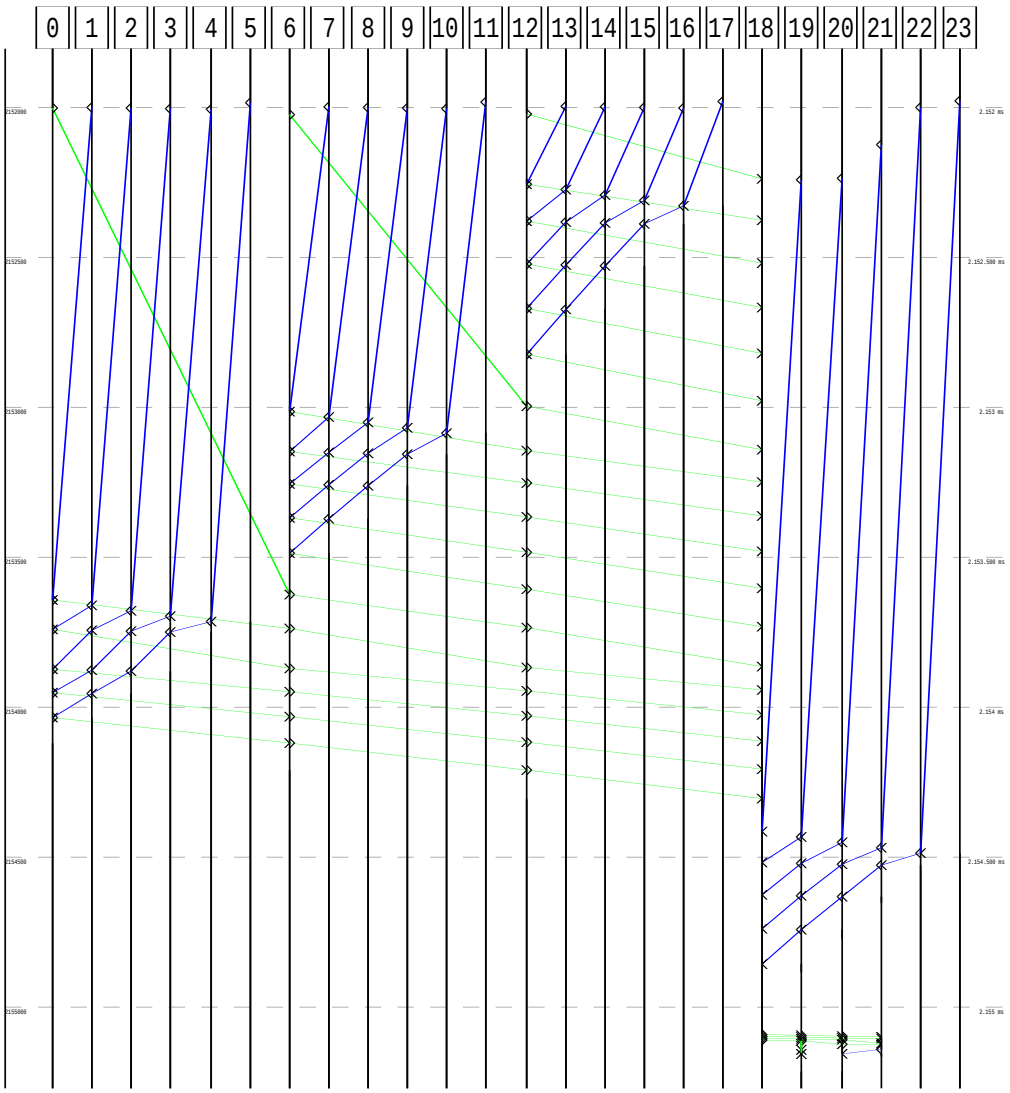

Figure 5.1: *The FdCheck's Timing Diagram*