Floor Plans

Dennis Ruffer

January 15, 2009

Contents

Chapter 1

Floor Plan Historical Background

1.1 The story that started it all

On 05 Feb 07, Les Snively wrote the following story in our XPlanner website

> In the SEAForth world, a floor plan has come to mean the location of specific chunks of code that handle particular processing tasks. It is a natural description of a data flow problem, with pipelining of tasks in both serial and parallel fashion. Since I/O location is fixed relative to nodes on the chip, floor planning describes the location and data flow of the functional processing.

The task that he assigned to me for accomplishing this was described as follows:

> An automatic graphical floor plan generator will be developed as a component of the Radio and RF IDE. The generator will take as it's input a standard VentureForth text file containing the design to be diagrammed. It will use the node definitions, and potentially special tags that will be defined, see below, to produce a graphical description of the floor plan for that file. It is recommended that the description be written in SVG format to make presentation via a server simple and easy to use with any standard browser.
>
> A set of tags should be developed to ease in parsing out the important labels for the graphic. These labels should include, but may not be limited to:
>
> - node name
> - brief functional description
> - active interfaces to other nodes including interface direction
> - names for messages sent from node

- status indicator of node code, such as
  - pending
  - in-progress
  - in-test
  - released
- probably some kind of indicator of the basic layout of the chip, such as
  - 4x4
  - 4x6
  - etc.

The initial release should be done quickly, with no more than 3 or 4 days effort, and probably less, to get an feel for how the technique works. Additional features can be added later as the need arises.

Chapter 2

Installing the vf/Plugins/FloorPlan.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

- vf/Plugins/doc/FloorPlan.pdf
- vf/Plugins/FloorPlan.f
- vf/Plugins/Graphics/pdf/FloorPlan.f
- vf/Plugins/Graphics/pdf/pdf.f
- vf/Plugins/Graphics/svg/FloorPlan.f
- vf/Plugins/Graphics/svg/svg.f
- vf/Plugins/comments.f
- vf/Plugins/idForth.f
- vf/Plugins/LaTeX.f
- vf/Plugins/links.f
- vf/Plugins/Plugins.f
- vf/Plugins/Projects.f
- vf/Plugins/strings.f
- vf/Plugins/numbers.f
- vf/gforth.fs
- projects/FloorPlan/FloorPlan.f
- projects/FloorPlan/FloorPlan.vf
- projects/FloorPlan/exterior/FloorPlan.htm
- projects/FloorPlan/exterior/FloorPlan.pdf
- projects/FloorPlan/exterior/FloorPlan.svg
- projects/FloorPlan/project.bat
- projects/FloorPlan/project.vfp

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!

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

Floor Plan User Interface

The following words are the user interface for the Floor Plan descriptions. They are each used within parenthesis, so they do not need to be removed if this plugin is not loaded. They model HTML tags (somewhat), so that they can be evaluated inside quoted strings. They affect the following lines:

`<noname>` Sets the current node

`<name ...>` Sets the text on the top line and sets the current node

`<rgb r g b>` Sets the background color of the node's box (only in PDF). The r, g and b numbers can have 0, 1 or 2 decimal places.

`<status ...>` Sets the text on the bottom line of the current node

`<function ...>` Adds lines to the middle of the current node

`<title ...>` Adds a named page of arrow entries. Name sets page.

`<read ...>` Adds a named arrow into the current node

`<write ...>` Adds a named arrow out of the current node

The read and write arrow directions are based upon the port address that is on top of the stack when they are executed. Typically, this address will be compiled as a literal immediately following these comments, but the affect will need to be emulated (with a **drop**) if this is not the case.

An alternative to having an address on the stack is also available, which can make these tags less obtrusive. If the tag is immediately followed by a neighbor direction word, such as:

```
<read 'rdlu ...>
```

then that will be used to indicate the direction and the stack will not be checked.

The names and locations of the arrows between ports are designed to line up with each other (e.g. read from one will overlap write from the other). This

should be noticeable as making the text bolder, but may be a blur if both names are not the same.

In the 'iocs, 'addr and 'data cases, the arrows are placed on the diagonal to prevent them from conflicting with the paths between nodes, and they are rotated around the node so they will point away from the other nodes. The 'iocs port also has the option to specify the individual bits, by preceding the description with a bit number. There are 5 positions available, and the chip design uses 2 bits per I/O pin. Therefore, you can specify the even or odd bit for the first 4 pins as 0 through 7 and the 5th pin (the default) is specified as either bit 16 or 17. This 5th pin is the diagonal one used if a bit number is not specified and the other pins use the 2 edges that face away from the other nodes. Thus, specifying 'iocs bits on internal nodes will conflict with the normal communication paths between nodes.

There is, however, only minimal error checking on the use of port or bit numbers on any node. Since the chip capabilities will change, it is left to the programmer to specify the capabilities that are appropriate for the hardware he is using.

Some additional commands:

`n to FpNode` used if you are not setting the name.

`FpRuntime` used to add unlabeled arrows for runtime events. (if the Events Plugin is also installed.)

`true to showUnusedNodes` to show unused nodes (PDF only).

`true to ignoreFloorPlan` to ignore these markers.

All of the preceding words must appear in your source code before the following words that are used to generate the Floor Plan diagrams.

`svgFloorPlan` to generate an SVG Floor Plan.

`pdfFloorPlan` to generate a PDF Floor Plan.

Chapter 5

Floor Plan Example

```
( <title exterior> ) \ Repetitive lines removed

\ Manual measurements (counts-watcher) taken on 30 Dec 08
00 to FpNode ( <function 12020-1> <function 11955-6>  )
: <tick> ( -- )   s" ( <write 'iocs 1 1> <write 'iocs 2 2> )" evaluate ;
: <corner> ( -- )   s" ( <write 'iocs 17 17> )" evaluate
  s" ( <write 'iocs 1 1>   <write 'iocs 3 3> )" evaluate
  s" ( <write 'iocs 5 5>   <write 'iocs 7 7> )" evaluate
  s" ( <write 'addr addr>  <write 'data data> )" evaluate ;
00 {node ( <name Node 00> <rgb 0 1 0>    <status corner> )  <corner>  node}
04 {node ( <name Node 04> <rgb 0 0.7 0>  <status bottom> )    <tick>  node}
16 {node ( <name Node 16> <rgb .92 0 0>  <status interior> )          node}
17 {node ( <name Node 17> <rgb 1.00 0 0> <status right> )     <tick>  node}
19 {node ( <name Node 19> <rgb 0 1 0>    <status uploader> )  <tick>  node}
22 {node ( <name Node 22> <rgb 0 1 1>    <status top> )       <tick>  node}
23 {node ( <name Node 23> <rgb 1 1 1>    <status corner> )  <corner>  node}

( exterior ) svgFloorPlan pdfFloorPlan
```

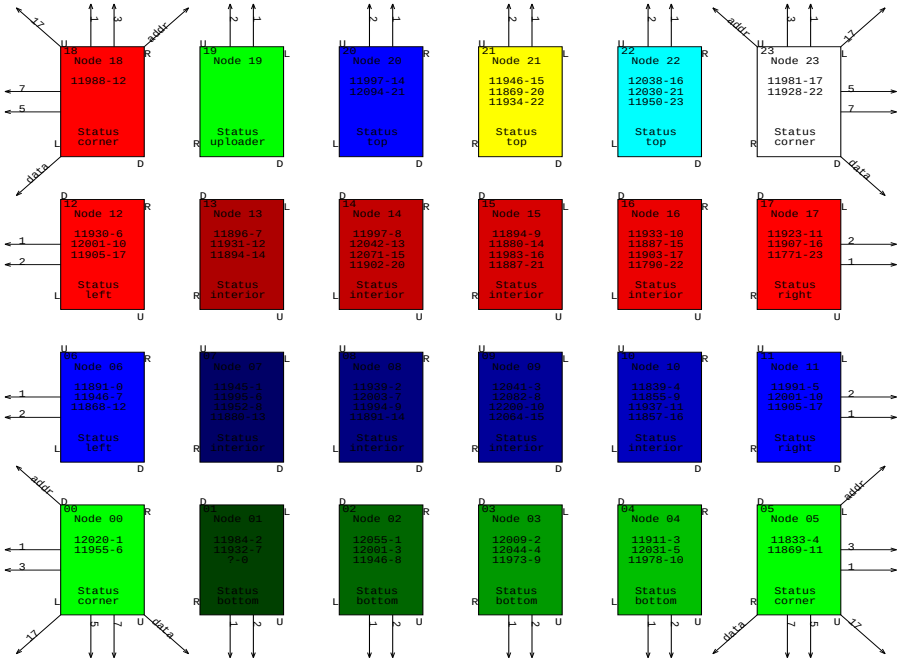

Figure 5.1: *The example application's loader Floor Plan*