# 14

# *Appendix A: Project Directory and Vim Editor*

In Chapter 5, we introduce the use of Verilog to design a basic testbench, which includes a bus functional model, design-under-test instantiation, and a monitor. This tutorial focuses on the introduction of an efficient project directory, basic Unix/Linux commands for using the simulation environment, and some important commands for using the Vim editor. We use the ddot design project, which was introduced in Chapter 10, as an example for this tutorial.

## 14.1 Project Directory

To promote consistency across projects and development teams, it is recommended to create a standardized project directory. This allows for an organized view of all design files, simulation testbenches, scripts, synthesis results, filelists, golden files, and other relevant materials, making it easier for designers to manage tasks, programs, and processes in a single dynamic platform.

As an example, Fig. 14.1 shows the project directory structure for the "ddot" design project. The top-level folder is named "basic-ddot", under which seven subfolders are created for managing different files. The **constraint** folder contains the constraint file (in this case, "Nexys-4-Master.xdc") used for the FPGA demonstration. The **dut** folder contains all the design files (".v" files), including the top module – "basic_ddot.v", the submodules – "FP_adder.v" and "FP_multipiler.v", and the "binary_lib.v" file which contains all the binary submodules for constructing FP adders and multipliers. The **filelist** folder contains the "filelist.f" file, which maintains all design files and testbenches to be compiled using the simulator. The **sim** folder contains the TCL script used for running the simulation, while the **syn** folder is used for directing all synthesis and implementation-related projects and files. The **tb** folder contains the testbench, and the **golden** folder stores all input data and golden output results.

To track and manage changes to the project, a version control system (VCS) such as Git or SVN (subversion) is typically used by IC design teams.

VCS platforms provide an efficient way to collaborate, share, and maintain the project within a development group. This is particularly useful as design projects grow in size and complexity, but even simple projects can benefit from tracking code changes with a VCS. SVN has been commonly used in IC design companies for years, while Git has recently become popular due to its use by developers collaborating on open-source projects.

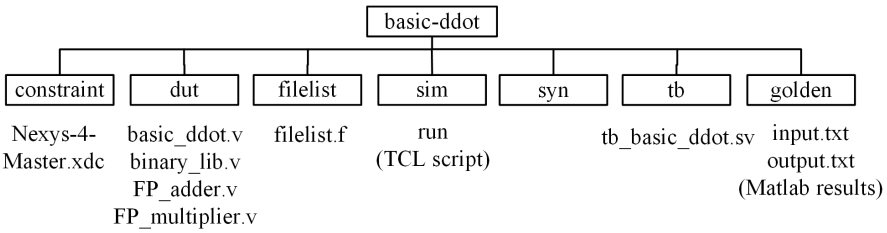



**FIGURE 14.1**
A Typical Project Directory

## 14.2 Basic Unix/Linux Commands

In the field of IC design, the majority of EDA vendors such as Siemens EDA, Synopsys, and Cadence offer a comprehensive portfolio of tools for IC design, verification, and manufacturing on Unix/Linux operating systems, but not for Windows OS. In addition, the software in Windows OS typically supports Unix/Linux commands. Therefore, it is necessary for IC designers to have knowledge of basic Unix/Linux commands. Although it may seem tedious at first, learning these commands can greatly improve efficiency. This section introduces only the most basic commands as examples, including:

- . : represents the current directory.
- .. : represents the parent directory.
- cd : changes the current directory.
- pwd: prints the current working directory.
- ls : lists all the folders and files in the current directory.

Fig. 14.2 shows an example of a project directory hierarchy in Unix.

```
xiaokunyang@~/basic-ddot$ ll
total 36
drwxrwxr-x 9 xiaokunyang xiaokunyang 4096 Dec 10 18:29 ./
drwxr-xr-x 5 xiaokunyang xiaokunyang 4096 Dec 10 16:13 ../
drwxrwxr-x 3 xiaokunyang xiaokunyang 4096 Dec 10 16:13 constraint/
drwxrwxr-x 2 xiaokunyang xiaokunyang 4096 Dec 10 16:48 dut/
drwxrwxr-x 2 xiaokunyang xiaokunyang 4096 Dec 10 16:13 filelist/
drwxrwxr-x 2 xiaokunyang xiaokunyang 4096 Dec 10 18:29 golden/
drwxrwxr-x 3 xiaokunyang xiaokunyang 4096 Dec 10 16:13 sim/
drwxrwxr-x 4 xiaokunyang xiaokunyang 4096 Dec 10 18:28 syn/
drwxrwxr-x 2 xiaokunyang xiaokunyang 4096 Dec 10 16:13 tb/
```

**FIGURE 14.2**
Project Directory in Unix

## 14.3 Vim Editor

When building a simulation environment, various files such as scripts, Verilog codes and SystemVerilog testbenches, golden models/results, etc., need to be worked on using a text editor. Vim, one of the most popular editors in industry, is widely used for several reasons.

Firstly, Vim is an open-source text editor created in 1976. Although it is popular in the Unix/Linux world, it is also available on Mac OS and Windows OS. It is very fast and lightweight, even when editing large files and/or source code. Secondly, Vim supports several programming languages and file formats, including ".v" and ".sv" files. It can detect file extensions and recognize specific text contents and syntax. Thirdly, Vim is configurable using the ".vimrc" source file. For example, the following configuration sets the font to Consolas and the font size to 11.

```
1 set guifont=Consolas:h11
```

One of the most significant advantages of using Vim is the ability to use many useful commands to increase programming efficiency. Although it has a bit of a learning curve, once you learn the commands, you will find it helpful and productive to use the editor. Many Vim commands are available, but in this book, only the most frequently used ones are introduced below.

### Two Modes Using Vim

Vim has two modes: command mode and insert mode. In the command mode, users can execute various commands such as navigating the file, copying, changing, or deleting code, and more. In the insert mode, users can type text into the file. To switch from the command mode to insert mode, users

need to type "i". To switch from insert mode back to command mode, users can type "Esc" or the "escape key". Below are the steps for writing and saving code with Vim:

- Step 1: Open or create a file with a chosen filename.
- Step 2: Type "i" to switch to the insert mode.
- Step 3: Enter your code in the file.
- Step 4: Hit "Esc" key to switch back to the command mode.
- Step 5: Type ":w!" to write the file, or "wq!" to write the file and exit Vim.

The write and exit commands also have the following variations:

- :w : Write changes to the original file.
- :w! : Force write changes (no warning is given).
- :wq : Write changes to the file and quit the editor.
- :q : Quit the editor (a warning is printed if a modified file has not been saved).
- :q! : Quit the editor without saving any changes (no warning is given).

### Navigating within a File

Vim's highly effective key-bindings allow you to write code without lifting your fingers from the keyboard. Instead of using arrow keys to navigate within a file, Vim provides the "h", "l", "k", and "j" keys in the command mode for navigation. Here's what each key does:

- h: Moves the cursor one character to the left.
- l: Moves the cursor one character to the right.
- k: Moves the cursor up one line.
- j: Moves the cursor down one line.

While moving fingers on the keyboard and using a mouse may only take seconds, using key-bindings can save hours and keep your fingers in the center of the keyboard. This can help keep programmers in flow-states and ultimately help them focus on writing code. Additionally, there are several additional keyboard shortcuts to navigate within a file:

- :set nu : Shows the number of lines in the current file.
- :N : Move cursor to the specified line (for example, :10 goes to line 10).

- :0 : Move cursor to the beginning of the current line.
- Shift+$ : Move cursor to the end of the current line.
- 1 followed by Shift+G: Jump to the first line in the file.
- Shift+G : Jump to the last line in the file.

### Commands for Copy, Paste and Delete

Here are some useful commands for copying, pasting, and deleting text in Vim. Each command can be repeated a certain number of times by typing a number before the command, and using the "." command can repeat the previous action. Additionally, the "u" command is essential for undoing changes made to the text.

- x : Deletes the character where the cursor is.
- dw : Deletes the word from where the cursor is.
- dd : Deletes the line where the cursor is.
- Ndd: Deletes N lines from where the cursor is.
- yy : Copies the current line where the cursor is.
- Nyy: Copies N lines starting from the current line.
- p : Pastes the copied or deleted text below the current line where the cursor is.
- u : Undoes the last change made to the text.

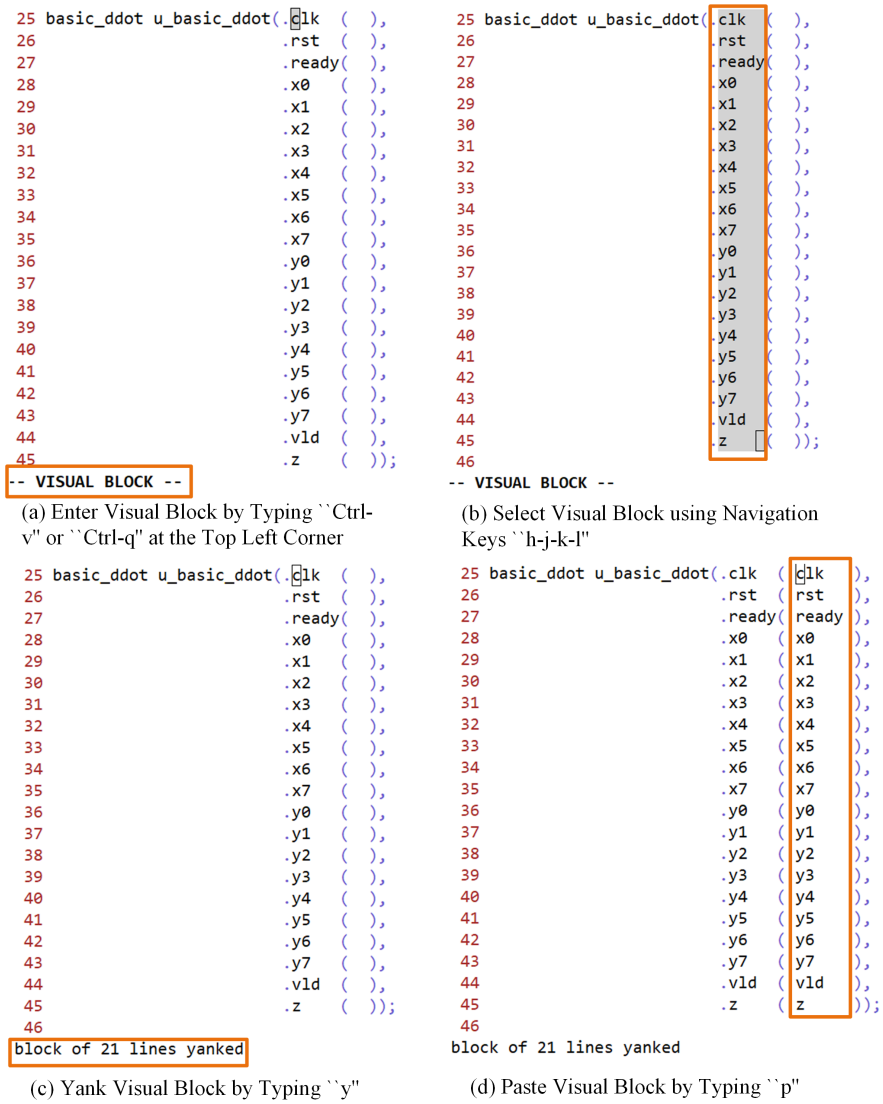
By knowing these commands, you can easily manipulate text in Vim for increased efficiency.

### Visual Block: Editing Text by Block/Row/Column

One of the main benefits of using Vim is the ability to edit text in block/row/-column in the visual block. In Unix/Linux, typing "Ctrl-v" enters block-wise visual block mode to highlight rows and columns. In Windows, the command is "Ctrl-q".

Once in visual block mode, the keyboard navigation keys "h-j-k-l" can be used to select and highlight text by blocks. Then, batch commands can be used to manipulate the selected text, such as delete ("d"), yank ("y"), or paste ("p"). To exit visual block mode, press "Esc".

For example, Fig.14.3 shows an example of copying and pasting a visual block (all the IO ports of the design) into parentheses for the design




(a) Enter Visual Block by Typing ``Ctrl-v'' or ``Ctrl-q'' at the Top Left Corner

(b) Select Visual Block using Navigation Keys ``h-j-k-l''

(c) Yank Visual Block by Typing ``y''

(d) Paste Visual Block by Typing ``p''

**FIGURE 14.3**
Copy and Paste a Visual Block (all the IO ports of the design) into the Parenthesis (IO ports connection for the design instantiation and connection)

instantiation and connection. First, enter visual block mode by typing "Ctrl-v" (Unix/Linux) or "Ctrl-q" (Windows) at the top left corner of the visual block.

Fig.14.3(a) shows that the first letter "c" of the "clk" signal is selected with a gray background. The Vim editor also displays "VISUAL BLOCK" in the bottom left corner of the screen. Next, select the visual block by moving the navigation keys. As shown in Fig.14.3(b), all the ports in lines 25-45 are selected as the visual block with a gray background. To copy the selected visual block, type "y", and Vim will display "block of 21 lines yanked" in the bottom left corner, as shown in Fig.14.3(c). After the visual block is copied, move the cursor to the top left position of the block where the copy will be pasted. As shown in Fig. 14.3(d), the cursor is moved to line 25 and into the parentheses, which is the top left corner of the block. Finally, type "p" to paste the copied visual block into the parentheses in lines 25-45.

The visual block feature in Vim is not limited to copying and pasting. It can be used with any other Vim command. Therefore, vertical alignment for writing Verilog is strongly recommended as it can make programming very productive and improve the readability of the code.

### Searching and Substitution

Searching and substitution are two of the most useful commands in Vim for reading and modifying text. Below are the commands for searching and substitution in Vim:

- /word: search and highlight all instances of the string "word" in the text.
- %s/original/new/g: substitute all instances of "original" with "new" in the text.

To search for a word in Vim, type the forward-slash (/) key followed by the search term in the command mode. For example, if you want to find all instances of the word "input" in the text, type "/input" and Vim will highlight all of them. You can navigate to the next instance of the word by typing "shift+n", or go to the previous instance by typing "n".

The global substitution command is similar to the command used in Perl scripts and is commonly used in programming. To substitute all occurrences of a word in the text, type "%s/original/new/g" in the command mode. The % symbol allows you to access all the content in the file and replace all occurrences in each line.

For example, in Fig.14.4(a), we type the command "%s/input/output/g" in the command mode of Vim to replace all instances of "input" with "output" in the text. Fig.14.4(b) shows the result of the substitutions on all the five lines. In each line, all the "input" words are replaced with "output". The "g" at the end of the substitution command stands for "global", which means that Vim will replace all occurrences of the original word in the entire text.

Overall, searching and substitution are powerful commands that can help you quickly find and modify text in Vim.

```
 1 module basic_ddot (input         clk  ,
 2                    input         rst  ,
 3                    input         ready,
 4                    input  [31:0] x0,x1,x2,x3,x4,x5,x6,x7,
 5                    input  [31:0] y0,y1,y2,y3,y4,y5,y6,y7,
 6                    output reg    vld  ,
 7                    output [31:0] z    );
 8 //*********************************
:%s/input/output/g
```

(a) Substitution Command.

```
 1 module basic_ddot (output         clk  ,
 2                    output         rst  ,
 3                    output         ready,
 4                    output  [31:0] x0,x1,x2,x3,x4,x5,x6,x7,
 5                    output  [31:0] y0,y1,y2,y3,y4,y5,y6,y7,
 6                    output reg    vld  ,
 7                    output [31:0] z    );
 8 //*********************************
5 substitutions on 5 lines                              5,20
```

(b) Substitution Result.

**FIGURE 14.4**
Substitution Command in Command Mode of Vim Editor.

### Highlighting

Highlighting lines or words of code can be very helpful for reading and understanding design code. In Vim, we can use the command "shift+v" to select and highlight the entire line where the cursor is. Once the entire line is selected, we can use the "j" key to move down and select more lines, or the "k" key to move up and select lines in the opposite direction.

Similarly, to select and highlight a word where the cursor is, we can use the command "shift+#". This command is commonly used for tracing design signals. After the word/signal is selected, we can use "shift+n" to find the next occurrence, or "n" to find the previous occurrence, to help us locate all instances of the selected word/signal in the design logic.

Here are the keyboard shortcuts for these commands:

- Shift+v : Select and highlight the entire line where the cursor is.
- shift+#: Select and highlight the word where the cursor is.

### Other Useful Commands

Additional useful commands in the Vim editor include the ability to compare and manipulate multiple files simultaneously. To split two files vertically, type "vsp" and for horizontal splitting, type "sp", in the command mode of the Vim editor. To switch between the two files, press "Ctrl + w" twice.

- vsp/sp : Split two files vertically or horizontally.
- Ctrl+w : Switch between the two open files.

These commands are particularly useful when writing testbenches. For example, in Fig.14.5(a), the command ":vsp ../dut/basic_ddot.v" is used to open the design file named "basic_ddot.v" while in the testbench file. Notice that the location of the design file is in the parent directory (specified using ".." to find the parent directory) and then the "dut" folder, so the path "../dut/basic_ddot.v" is needed. Fig.14.5(b) shows the result of this command, with the design file on the left window and the testbench in the right window.

Other useful commands include:

- gf : Open the file path under the cursor.
- Ctrl+o: Return to the previous file.

The "gf" command is used to check the file path in the filelist. For example, in Fig. 14.6(a), it shows a filelist contains four design files and one testbench. Move the cursor to the "basic_ddot.v" file or path and type the "gf" command to check the path. If the file path is correct, the file will be opened as shown in Fig. 14.6(b). Otherwise, if the file path is incorrect, an error message such as "Can't find file ../tb/basic_ddot.v" in path" will be displayed, as shown in Fig. 14.6(c). To return to the filelist, type "Ctrl+o".

```
25 ddot_streaming u_ddot_streaming(.clk  (clk  ),
26                                 .rst  (rst  ),
27                                 .ready(ready),
28                                 .x0   (x0   ),
29                                 .x1   (x1   ),
30                                 .x2   (x2   ),
31                                 .x3   (x3   ),
32                                 .x4   (x4   ),
33                                 .x5   (x5   ),
34                                 .x6   (x6   ),
35                                 .x7   (x7   ),
36                                 .y0   (y0   ),
37                                 .y1   (y1   ),
38                                 .y2   (y2   ),
39                                 .y3   (y3   ),
40                                 .y4   (y4   ),
41                                 .y5   (y5   ),
42                                 .y6   (y6   ),
43                                 .y7   (y7   ),
44                                 .vld  (vld  ),
45                                 .z    (z    ));
vsp ..\dut\basic_ddot.v
```

(a) Split Window Command.

```
 1 module basic_ddot (input         clk  ,                  | 25 ddot_streaming u_ddot_streaming(.clk  (clk  ),
 2                    input         rst  ,                  | 26                                 .rst  (rst  ),
 3                    input         ready,                  | 27                                 .ready(ready),
 4                    input  [31:0] x0,x1,x2,x3,x4,x5,x6,x7,| 28                                 .x0   (x0   ),
 5                    input  [31:0] y0,y1,y2,y3,y4,y5,y6,y7,| 29                                 .x1   (x1   ),
 6                    output reg    vld  ,                  | 30                                 .x2   (x2   ),
 7                    output [31:0] z    );                 | 31                                 .x3   (x3   ),
 8 //*********************************                     | 32                                 .x4   (x4   ),
 9 //****    Datapath Design *********                     | 33                                 .x5   (x5   ),
10 //*********************************                     | 34                                 .x6   (x6   ),
11 wire [31:0] p0, p1, p2, p3, p4, p5, p6, p7;             | 35                                 .x7   (x7   ),
12 wire [31:0] s0, s1, s2, s3, s4, s5;                     | 36                                 .y0   (y0   ),
13                                                         | 37                                 .y1   (y1   ),
14 // The first stage of multipliers                       | 38                                 .y2   (y2   ),
15 FP_multiplier u0_fp_mul(.clock   (clk  ),               | 39                                 .y3   (y3   ),
16                         .reset   (rst  ),               | 40                                 .y4   (y4   ),
17                         .io_in_a (x0   ),               | 41                                 .y5   (y5   ),
18                         .io_in_b (y0   ),               | 42                                 .y6   (y6   ),
19                         .io_out_s(p0   ));              | 43                                 .y7   (y7   ),
20                                                         | 44                                 .vld  (vld  ),
21 FP_multiplier u1_fp_mul(.clock   (clk  ),               | 45                                 .z    (z    ));
..\dut\basic_ddot.v                    1,1          Top tb_basic_ddot.v [+]              31,18-46
```

(b) Split Window Result.

**FIGURE 14.5**
Split Windows in Command Mode of Vim Editor.

```
  1 ../dut/binary_lib.v
  2 ../dut/FP_multiplier.v
  3 ../dut/FP_adder.v
  4 ../dut/basic_ddot.v
  5 ../dut/tb_basic_ddot.v
~
~
"filelist.f" 5L, 109B
```

(a) Go To File Command.

```
module basic_ddot (input         clk   ,
                   input         rst   ,
                   input         ready,
                   input  [31:0] x0,x1,x2,x3,x4,x5,x6,x7,
                   input  [31:0] y0,y1,y2,y3,y4,y5,y6,y7,
                   output reg    vld   ,
                   output [31:0] z     );
<elist\..\dut\basic_ddot.v" 144L, 4505B           4,2-16
```

(b) Go To File Result - Correct Path.

```
  2 ../dut/FP_multiplier.v
  3 ../dut/FP_adder.v
  4 ../tb/basic_ddot.v
  5 ../dut/tb_basic_ddot.v
~
~
~
E447: Can't find file "../tb/basic_ddot.v" in path
Press ENTER or type command to continue
```

(c) Go To File Result - Incorrect Path.

**FIGURE 14.6**
Go To File in Command Mode of Vim Editor.