Verilog for Digital Design©

By

Raymond L. Barrett, Jr., PhD, PE

CEO, American Research and Development, LLC

Raymond L. Barrett, III

IT, American Research and Development, LLC

Abstract & Preamble

Engineering Practices - With the recognition of the Professional Engineering status for the practice of Computer Engineering in April of 2009, the practice of Control Systems Engineering in October of 2011, and the practice of Software Engineering in April of 2013, there has been the need for specialized continuing education courses related to these practices. Computer Engineering majors may have taken a course with some component of digital design without using Verilog, or may have had a course using the VHDL language. Control Systems engineers will find Verilog to be a useful tool for modeling and simulating real-time hardware and embedded systems for control applications, Software engineers can apply the principles of the concurrent parallel system representations available in Verilog for numerous applications. Other disciplines, including Electrical Engineers will also find the digital design practices encapsulated in Verilog useful all the way to the device level representations available in the language.

For this course, we assume that all readers are familiar with digital design concepts, but are interested in an introduction to the capabilities of the Verilog Design Language and how to use it in the design practice. Likewise, we assume that readers generally have rudimentary experience with computer languages, text editors, and general concepts of computer program compilation.

Digital Design Methodologies – Nearly all of today's digital design approaches find a basis in early design practices inherited from analog circuit design. A circuit schematic was drawn using electrical or logic-gate symbols and interconnected to represent the design intent. Analysis and expected behavior was annotated to the connections, often in the form of Boolean equations and dynamic behavior represented using timing diagrams. At first, nodal and loop descriptive equations were written and hand-analyzed. As soon as computer-aided differential equation solvers became available, they were employed in time-domain simulators to verify the behavior of essentially analog circuit representation of digital logic. For digital design practice, however, the continuous-time , continuous-magnitude signal representation used in analog design practice proves non-optimal.

In early practice, the collection of circuit schematics, timing diagrams, simulation results, and narrative descriptive text was also used as design documentation as a deliverable. The US military funded the Very High-Speed Integrated Circuit (VHSIC) initiative (https://en.wikipedia.org/wiki/VHSIC) and as part of that effort, mandated use of the VHSIC Hardware Description Language (https://en.wikipedia.org/wiki/VHDL) as a standardized form of design documentation. Initially, the designers attempted to document designs from device level through higher levels of complexity to full system descriptions. In various scopes, that same VHDL is used in Europe and often still mandated by the US military. Compared to earlier practice, VHDL is superior to the ad-hoc prior practice. Verilog has many of the features inherited from VHDL practices, but has added strengths.

As computers and their languages evolved, the Verilog language (http://www.verilog.com) was developed that combined features borrowed from the “C” language, but maintained a focus on logic design. Verilog is less verbose than VHDL that was implemented on an “ADA” language backbone basis, and has been adopted by a wide range of digital designers for its ability to support additional design emphasis as well as for supporting roles in design description and documentation.

Predecessor methodologies are still in use to some degree, but Verilog has become a popular tool for design capture, as well design synthesis. Verilog tools supporting implementation capabilities targeted to numerous technologies, including: discrete devices at a board level, Field-Programmable Gate Array (FPGA) devices, Application-Specific Integrated Circuit (ASIC) custom devices, and even embedded “C” programs in devices using the Verilator (http://www.veripool.org/projects/verilator) and other translator applications.

There are several textual and graphical design entry mechanisms available for Verilog tool suites. This course employs a simple text-editor for alphanumeric program entry and a compiler that produces a textual output. More complex GUI support is available including some proprietary schematic capture methodologies with symbol generators and other tools that are not discussed here. We introduce the simplest tools we can find to get hands-on experience with the language, but we later direct the reader to somewhat more complicated tools that may be used to increase productivity.

1.0 Introduction to Verilog Design

1.0.1 The Verilog Language Abstraction Features - Verilog is a parallel and concurrent computer language, in addition to supporting more common sequential programming constructs, and it can be used at several levels of design abstraction. At the highest level, Verilog can be used as a general-purpose numerical computation platform to capture some of the essence of digital system behavior.

In another, more-restrictive environment, it can be used as a "Register Transfer Level" (RTL) digital design-capture tool for preparing a design for synthesis into target digital hardware.

In its most detailed usage, it can be used to describe digital components at the device level, including MOS devices and components, along with the interconnecting wiring of the components. To some extent, we can choose and interchange the level of abstraction for component sub-systems in a design. At all levels, though, Verilog maintains a relationship with its roots in logic design by defining signals as "wires" and "reg" register bit value assignments, primarily for simulation.

In the signal representations, individual signal bits can be represented a "1", "0", "X" (unknown), or "Z" (un-connected) value with other representations available as collections. Yes, we can represent signals as floating point entities, but only as Behavioral Level constructs (not currently synthesizable to digital logic).

We explore each of the levels of abstraction, starting from the highest level and progressing to a detailed structural level. At each level, we enter a design using a text-processor, compile that design to an intermediate file format, and display the results in the time-domain using a waveform display software tool. All tools are Free, Open-Source Software (FOSS) and can be obtained from on-line repositories at the links provided.

1.0.2 The Verilog Language Simulation Features - Verilog employs a "discrete-event" methodology for digital signal representation that is based on the four-valued binary signal representation and an event-queueing, time-sequence based process model.

Most physics-based analog design systems construct models with their signals as continuous-time and continuous magnitude representations. In such analog systems all signal levels are valid as well as all time intervals; in contrast to the discrete-event signal representation.

Some Digital-Signal Processing (DSP) systems use different discrete-time approach with magnitude representations as integer or floating-point signals but with those values only valid at discrete times, evaluated at all clock instances. Verilog does not require evaluation at every time instance, although it is common in many systems to have a regular clock signal.

The discrete-event approach treats the time variable as an index into its event queue. Component "modules" are stimulated by input signals and produce output signals that are scheduled for action during the execution of the Verilog program.

The component modules are relatively free to respond in any logical fashion but must obey a causal relationship; outputs are related to inputs, but with their output not preceding inputs. A rudimentary software tool is included in the compiler to detect non-causal attempts.

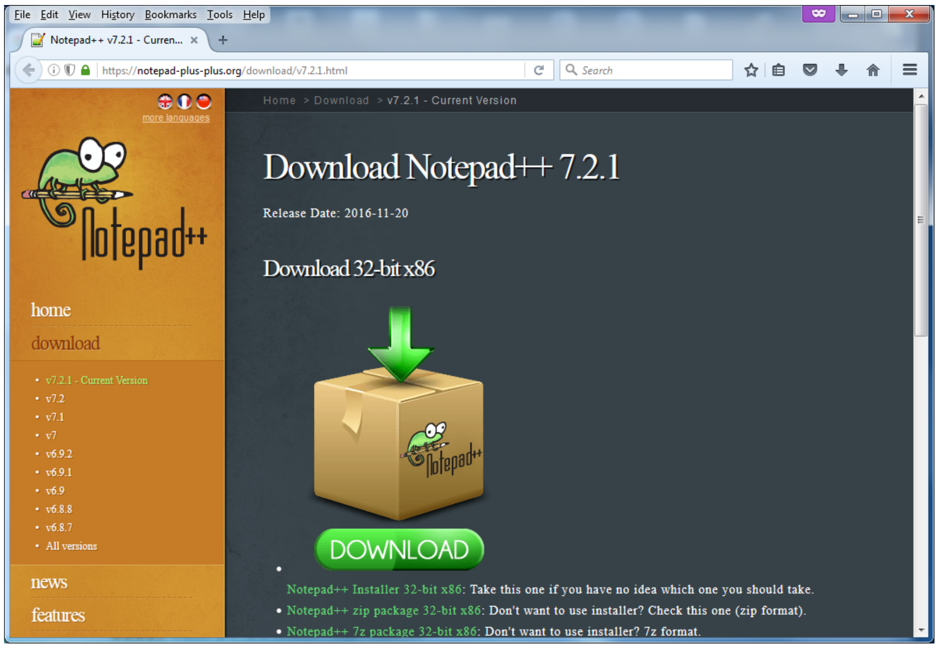
The discrete-event approach is most-often the most simulation time-efficient, as well as simulation space efficient for digital design because intermediate signal and time values are not computed or stored; they are displayed as continuous between events for ease of interpretation. Later additions to the language re-capture some of the analog capabilities in super-set versions like Verilog-A and System-Verilog.

1.0.3 Verilog Language Design Entry - As a computer language, Verilog can be entered as a single text-based document or set of documents and translated using many alternate proprietary and Free-Open-Source Software tools (FOSS). The tools range from very simple single-function programs to complex multi-function integrated design environments. We employ only the FOSS tools throughout, but begin using the tools in the simplest form and at the highest level of abstraction to produce an introductory "Hello Word" message to the designer. Admittedly this is a limited use Verilog program but the entry, compilation, and running the program instills confidence for the experiences that follow.

The particular text editors are recommended as compatible with the "**verilog.v**" file notation checking language syntax and signals errors they encounter. Each text editor provides line-number references that the compiler uses to indicate where it finds certain errors during compilation. Two such text editors are introduced and compared, one associated with Linux and the other with the Microsoft Operating system environments.

The alpha-numeric representation of variables can be case-sensitive and many of the word-processor software that is used for human-language text-processing often injects what it is programmed to interpret as typing errors in a "correction" that creates Verilog language errors. Please do not attempt to use a word-processor for program entry and stick with those introduced here; these text processors will reduce avoidable frustrations.

1.0.4 Recommended Text Editor Tools – The Microsoft Windows Operating System (OS) environment is widely available and we begin by introducing the “notepad++” text editor. You can download the editor from: https://notepad-plus-plus.org/download/v7.2.1.html in either 32-bit or 64-bit versions. You will probably find that the 32-bit version works equally well in either environment, and also has no problem in Windows7 and Windows10 (both were tested by us and work well). Again, we recommend that you avoid trying the use of a “word processor” application because many of them inject “hidden” characters and automatic features interact adversely with the Verilog language syntax.


File Edit View History Bookmarks Tools Help
Notepad++ v7.2.1 - Curren...
https://notepad-plus-plus.org/download/v7.2.1.html
Search
more languages
Notepad++
home
download
v7.2.1 - Current Version
v7.2
v7.1
v7
v6.9.2
v6.9.1
v6.9
v6.8.8
v6.8.7
All versions
news
features
Home > Download > v7.2.1 - Current Version
DOWNLOAD

Download Notepad++ 7.2.1

Release Date: 2016-11-20

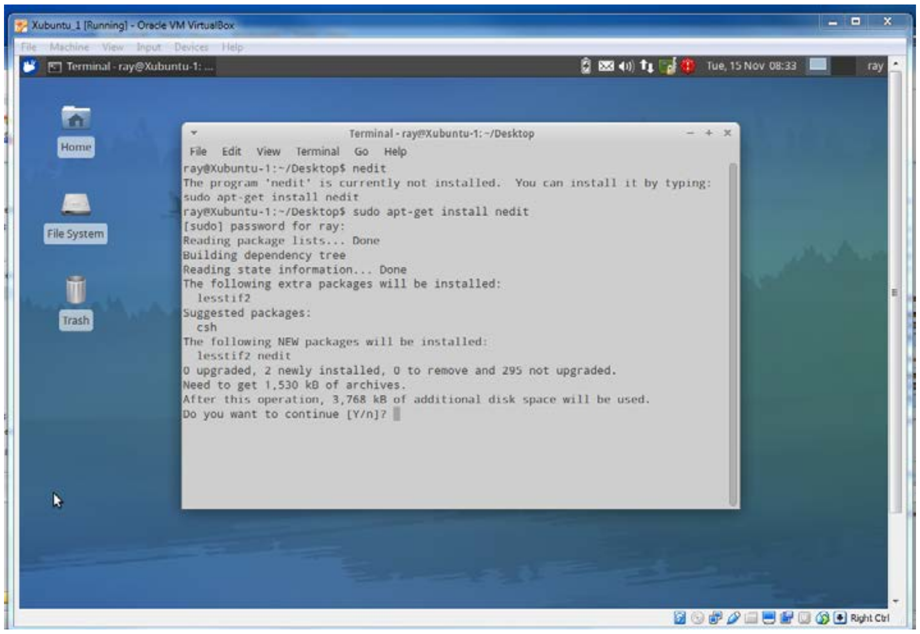
Download 32-bit x86

- Notepad++ Installer 32-bit x86: Take this one if you have no idea which one you should take.
- Notepad++ zip package 32-bit x86: Don't want to use installer? Check this one (zip format).
- Notepad++ 7z package 32-bit x86: Don't want to use installer? 7z format.

Figure 1.0 - Installing the notepad++ program in Microsoft Windows

In a Linux environment, we have tested and illustrate examples using the “nedit” text editor in a Xubuntu installation. You can manually download the editor from sourceforge at: https://sourceforge.net/projects/nedit, but often just following your system’s installation procedure works well; usually:

```
sudo apt-get nedit install
```

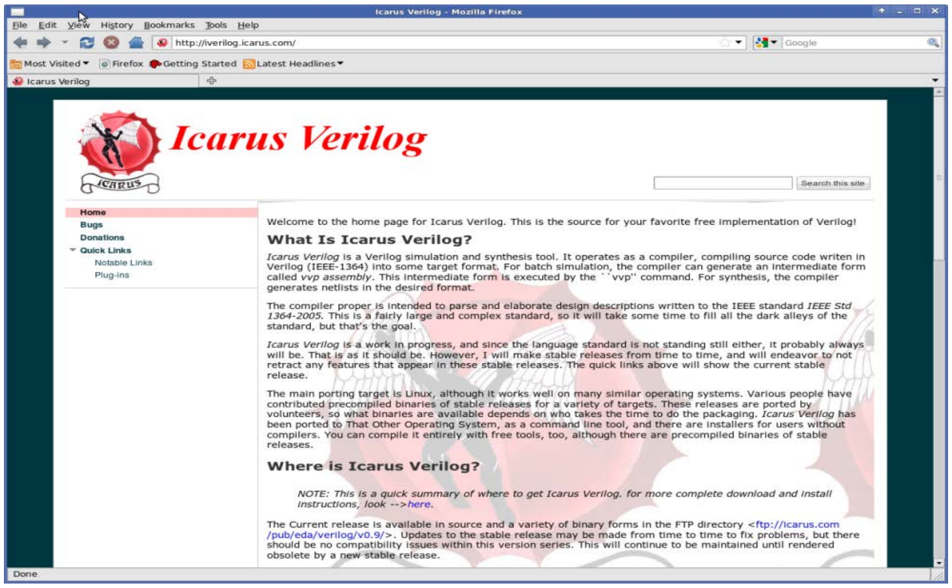

Xubuntu_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal - ray@Xubuntu-1: ...
Tue, 15 Nov 08:33
ray
Home
File System
Trash
Terminal - ray@Xubuntu-1: ~/Desktop
File Edit View Terminal Go Help
Right Ctrl

```
ray@Xubuntu-1:~/Desktop$ nedit
The program 'nedit' is currently not installed.  You can install it by typing:
sudo apt-get install nedit
ray@Xubuntu-1:~/Desktop$ sudo apt-get install nedit
[sudo] password for ray:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  lesstif2
Suggested packages:
  csh
The following NEW packages will be installed:
  lesstif2 nedit
0 upgraded, 2 newly installed, 0 to remove and 295 not upgraded.
Need to get 1,530 kB of archives.
After this operation, 3,768 kB of additional disk space will be used.
Do you want to continue [Y/n]?
```

Figure 1.1 - Installing the nedit program in Xubuntu Linux

If you are using the Linux environment, hopefully you are already fairly familiar with your version and its installation procedures. Most of the early development of Verilog was done in a Linux environment and there are many resources to support the installation of the tools, but we have only the Xubuntu environment actually installed and working. Where there are important points to be made about the Linux environment, we attempt to point out special considerations.

1.0.5 Verilog Language Compiler Tool – As with many languages, there are standards for the language (https://en.wikipedia.org/wiki/Verilog) and various revisions, and more than one accepted set of software tools are available to use each version of the language. We have chosen the **Icarus Verilog** version of Verilog, in part because it is the same in both Windows and Linux environments.


Icarus Verilog - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://iverilog.icarus.com/
Most Visited Firefox Getting Started Latest Headlines
Icarus Verilog
Search this site
Home
Bugs
Donations
Quick Links
Notable Links
Plug-ins
Done

Icarus Verilog

Welcome to the home page for Icarus Verilog. This is the source for your favorite free implementation of Verilog!

What Is Icarus Verilog?

Icarus Verilog is a Verilog simulation and synthesis tool. It operates as a compiler, compiling source code writen in Verilog (IEEE-1364) into some target format. For batch simulation, the compiler can generate an intermediate form called *vvp assembly*. This intermediate form is executed by the ``vvp'' command. For synthesis, the compiler generates netlists in the desired format.

The compiler proper is intended to parse and elaborate design descriptions written to the IEEE standard *IEEE Std 1364-2005*. This is a fairly large and complex standard, so it will take some time to fill all the dark alleys of the standard, but that's the goal.

Icarus Verilog is a work in progress, and since the language standard is not standing still either, it probably always will be. That is as it should be. However, I will make stable releases from time to time, and will endeavor to not retract any features that appear in these stable releases. The quick links above will show the current stable release.

The main porting target is Linux, although it works well on many similar operating systems. Various people have contributed precompiled binaries of stable releases for a variety of targets. These releases are ported by volunteers, so what binaries are available depends on who takes the time to do the packaging. *Icarus Verilog* has been ported to That Other Operating System, as a command line tool, and there are installers for users without compilers. You can compile it entirely with free tools, too, although there are precompiled binaries of stable releases.

Where is Icarus Verilog?

NOTE: This is a quick summary of where to get Icarus Verilog. for more complete download and install instructions, look -->here.

The Current release is available in source and a variety of binary forms in the FTP directory <ftp://icarus.com/pub/eda/verilog/v0.9/>. Updates to the stable release may be made from time to time to fix problems, but there should be no compatibility issues within this version series. This will continue to be maintained until rendered obsolete by a new stable release.

Figure 1.2- Icarus Verilog Web Page

We introduce Verilog compilation using the Icarus Verilog (**iVerilog**) tool. **iVerilog** (http://iverilog.icarus.com) can be employed equally well in many Operating System (OS) environments but you may need to try more than one installer to get it working; for Windows there is a list (http://bleyer.org/icarus) of installers, but for Linux, the best approach is to use the installer capability of your “dialect” of Linux trying your supported approach, perhaps something like: (http://askubuntu.com/questions/217555/how-to-use-verilog-hdl-on-ubuntu)

```
sudo apt-get install iverilog
```

It is not uncommon for the installation to inform you that some other software is also needed to complete the iVerilog installation (most often a “C” compiler for your particular environment). You will probably not need both Windows and Linux, so choose the one you are most familiar working with and add the other as your interest takes you.

Take heart, getting the software tools installed is probably the most difficult part of this course and we have made the introduction as painless as possible. There will be more software tools introduced but the process builds on increments that help you gain confidence.

1.0.6 Icarus Verilog Compiler Tool Installation – In the Microsoft Windows environment, from the list of installers at: http://bleyer.org/icarus, you should choose the latest setup.exe installer version for your version of Windows. Usually, the 32-bit versions work as well as 64-bit versions, but there may be differences between Intel and AMD processor hardware, particularly if you do choose 64-bit software. Occasionally there are also issues with the PATHNAME variable in Windows that must be set for the program to be “found.” Follow the recommendations of your Windows version and help files for setting the installation-specific variables.

In a Linux environment, you should visit: http://iverilog.icarus.com and follow links to the source code to compile, or you can use your software registry for Linux by executing the “sudo apt-get iverilog install” instruction as the super-user and supplying your password.

1.0.7 Icarus Verilog Programming Example – We introduce the “Hello World” Program in Windows as a simple example of program entry, compilation and execution. On your desktop, right-click and select New-> folder, and name the folder **Verilog**, and open so that you see the following window:

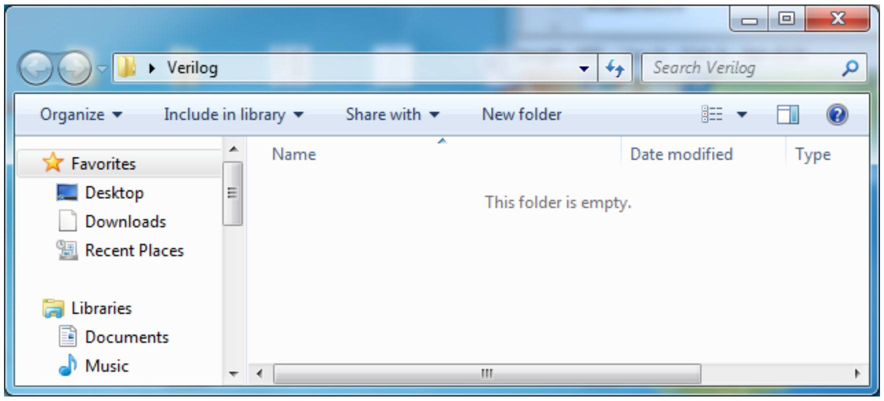


Figure 1.3 - Verilog File Folder in Windows

In your Start list, select “notepad++” with a single-click, and the notepad++ window should appear on your desktop looking like:

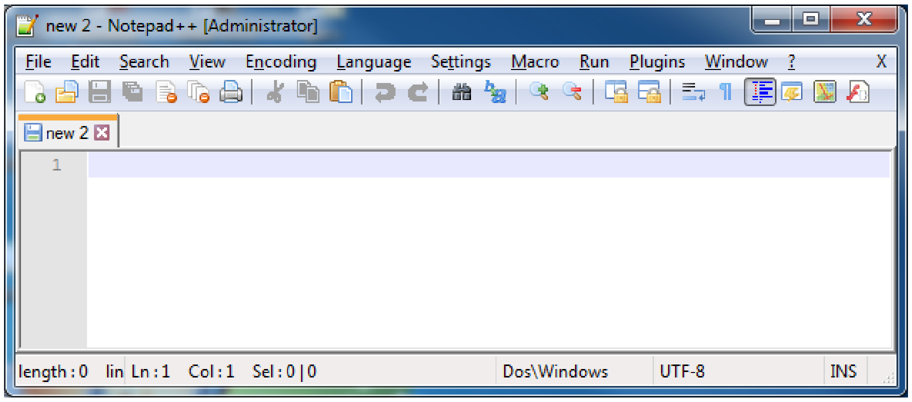

new 2 - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
new 2
length : 0 lin Ln : 1 Col : 1 Sel : 0 | 0 Dos\Windows UTF-8 INS

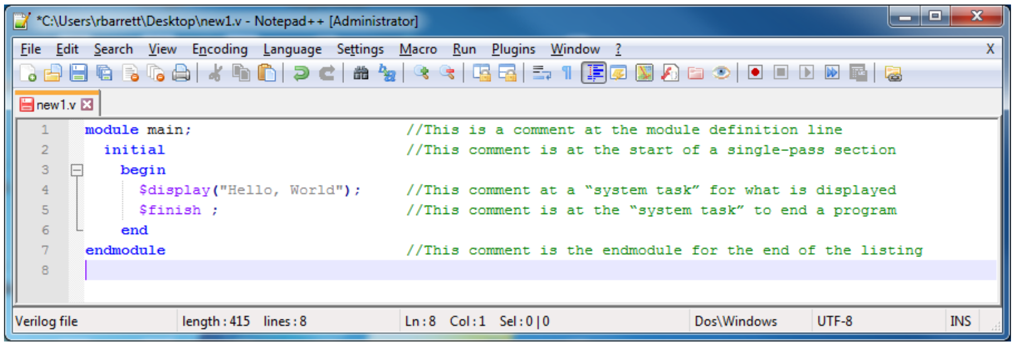
Figure 1.4 – Notepad++ Text Editor in Windows

In the window, type the following Verilog program-listing text:

```
module main;                        //This is a comment at the module definition line
  initial                           //This comment is at the start of a single-pass section
    begin
      $display("Hello, World");     //This comment at a “system task” for what is displayed
      $finish ;                     //This comment is at the “system task” to end a program
    end
endmodule                           //This comment is the endmodule for the end of the listing
```

Program Listing 1.0

If you are careful typing an pay close attention to punctuation (be careful about where the semicolons are located), your text window should now appear with the program listing text as:


*C:\Users\rbarrett\Desktop\new1.v - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
new1.v
Verilog file length : 415 lines : 8 Ln : 8 Col : 1 Sel : 0 | 0 Dos\Windows UTF-8 INS

```
module main;                        //This is a comment at the module definition line
  initial                           //This comment is at the start of a single-pass section
    begin
      $display("Hello, World");     //This comment at a “system task” for what is displayed
      $finish ;                     //This comment is at the “system task” to end a program
    end
endmodule                           //This comment is the endmodule for the end of the listing
```

Figure 1.5 – Verilog Program in Notepad++ Text Editor in Windows

Using the text window File menus, select Save As, and use the “Save in” box to direct the save to Desktop, and under that, the Verilog directory/folder that you just created, and it should appear just as it was created. In the File name, type “hello” without the quotes, and in the Save as type, scroll down to the Verilog file designation, and you should see:

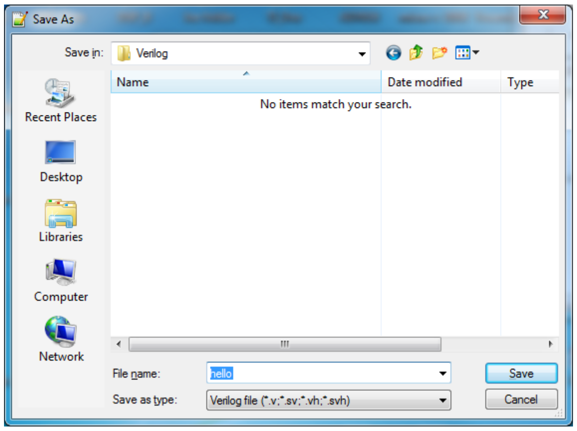


Figure 1.6 – Saving the Verilog Program in Notepad++ Text Editor in Windows

Press the “Save” button so the Verilog window should now appear as:

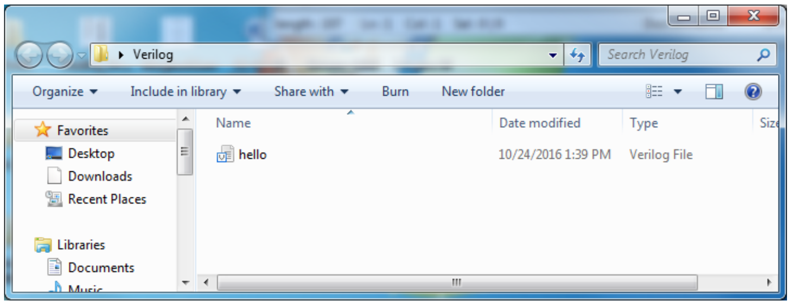


Figure 1.7 – The Verilog “hello” program in the Verilog File Folder in Windows

Type the Windows-R key and you should get a DOS command window like:

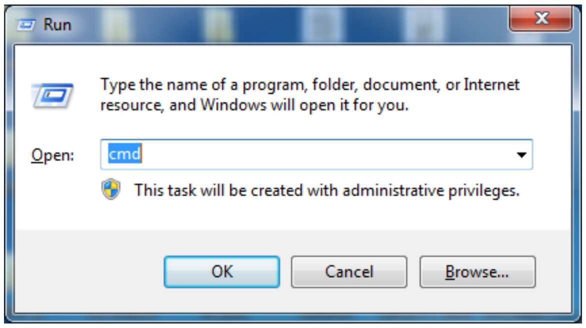
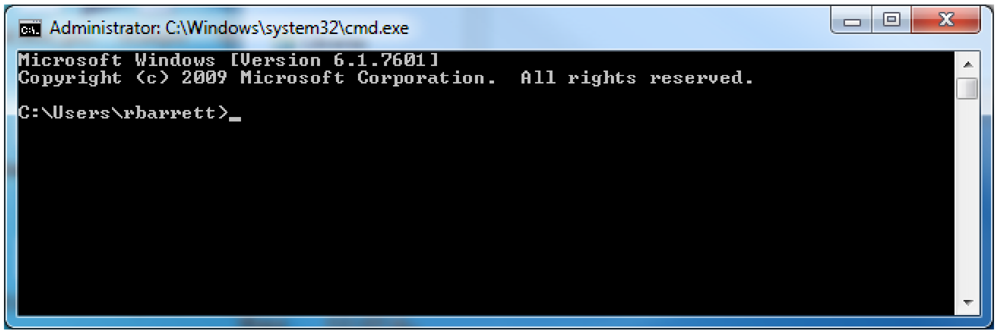

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
cmd
This task will be created with administrative privileges.
OK
Cancel
Browse...

Figure 1.8 – Obtaining a Command Dialog Box in Windows

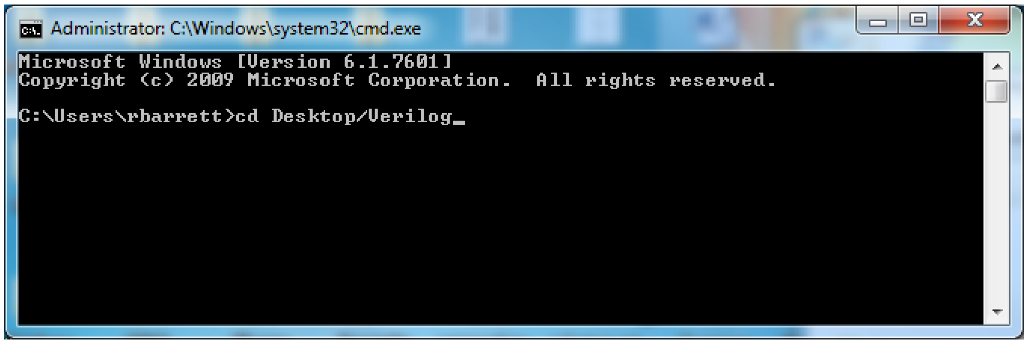
Press OK, and you should get a command window like:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\rbarrett>_
```

Figure 1.9 – An Empty log-in Command Window in Windows

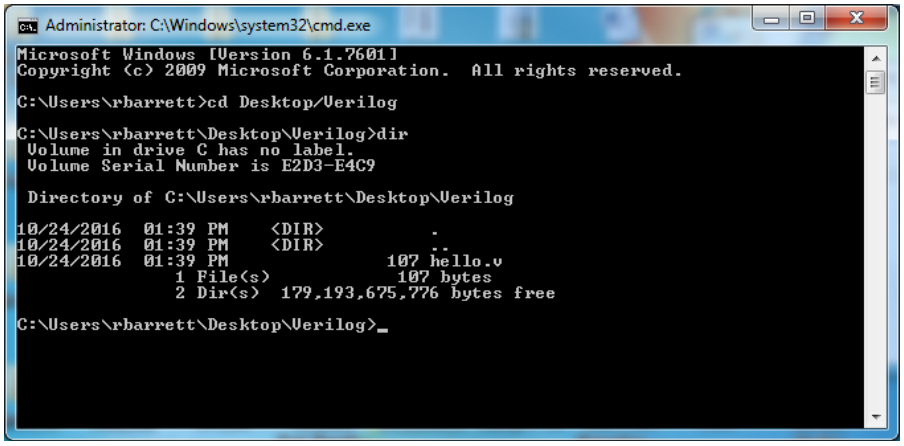
In this window, type **cd Desktop/Verilog** as shown below:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\rbarrett>cd Desktop/Verilog_
```

Figure 1.10 – Linking the Command Window to the Verilog Folder in Windows

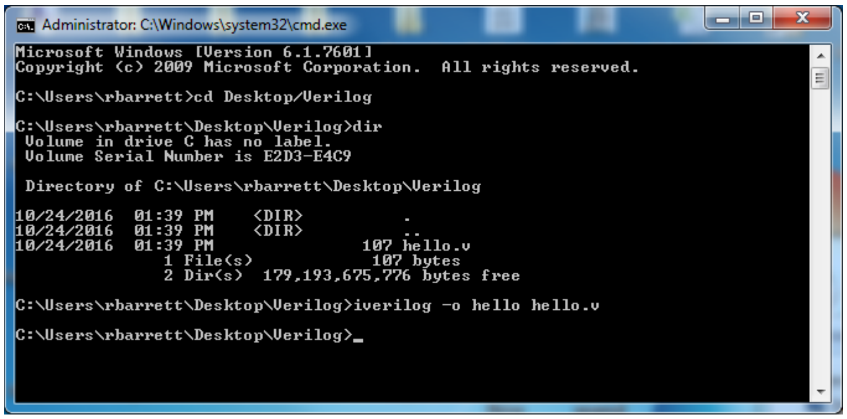
Type Enter, followed by "**dir**" and the window should reflect the view shown below:



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\rbarrett>cd Desktop/Verilog

C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

10/24/2016  01:39 PM    <DIR>          .
10/24/2016  01:39 PM    <DIR>          ..
10/24/2016  01:39 PM               107 hello.v
               1 File(s)            107 bytes
               2 Dir(s)  179,193,675,776 bytes free

C:\Users\rbarrett\Desktop\Verilog>_
```

Figure 1.11 – Listing the Contents of the Verilog Folder in Windows

Note that the view shows the contents of the Verilog directory on your desktop, including the **hello.v** file. Type: "**iverilog –o hello hello.v**" and after a few seconds, you should see:



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\rbarrett>cd Desktop/Verilog

C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

10/24/2016  01:39 PM    <DIR>          .
10/24/2016  01:39 PM    <DIR>          ..
10/24/2016  01:39 PM               107 hello.v
               1 File(s)            107 bytes
               2 Dir(s)  179,193,675,776 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o hello hello.v

C:\Users\rbarrett\Desktop\Verilog>_
```

Figure 1.12 – Compiling the hello Program in the Verilog Folder in Windows

At this point, you have entered your Verilog program and saved it, and then compiled it, although the result of the compilation doesn not become visible until you explicitly list the folder contents.

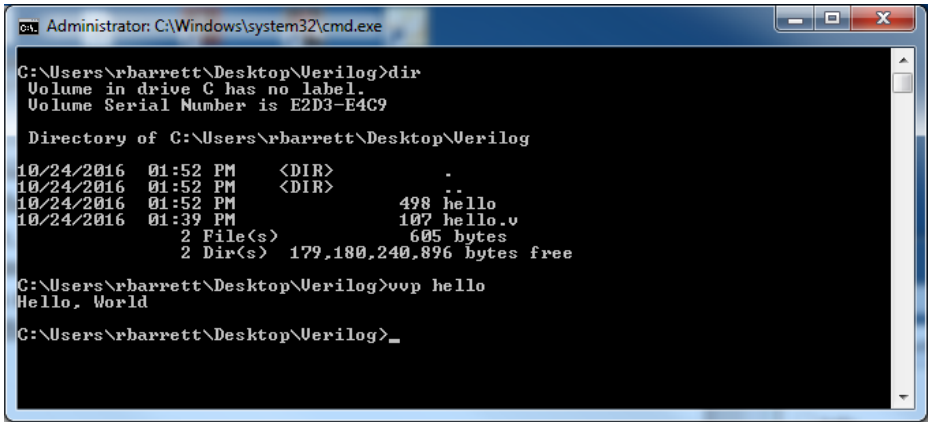
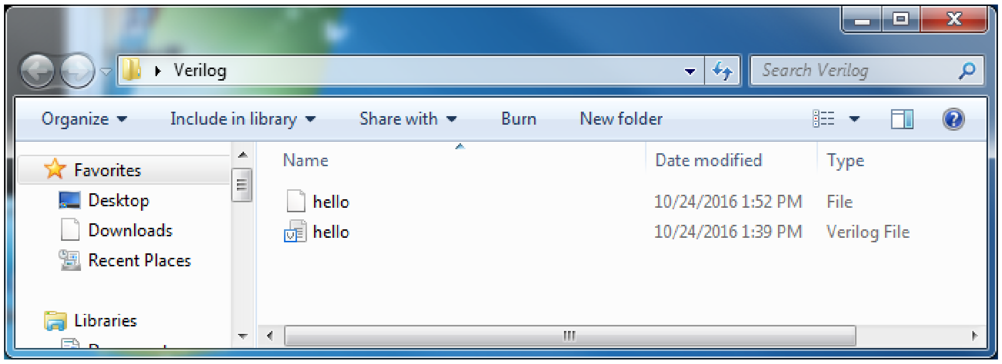

```
Administrator: C:\Windows\system32\cmd.exe

C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

10/24/2016  01:52 PM    <DIR>          .
10/24/2016  01:52 PM    <DIR>          ..
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
               2 File(s)            605 bytes
               2 Dir(s)  179,180,240,896 bytes free

C:\Users\rbarrett\Desktop\Verilog>vvp hello
Hello, World

C:\Users\rbarrett\Desktop\Verilog>_
```

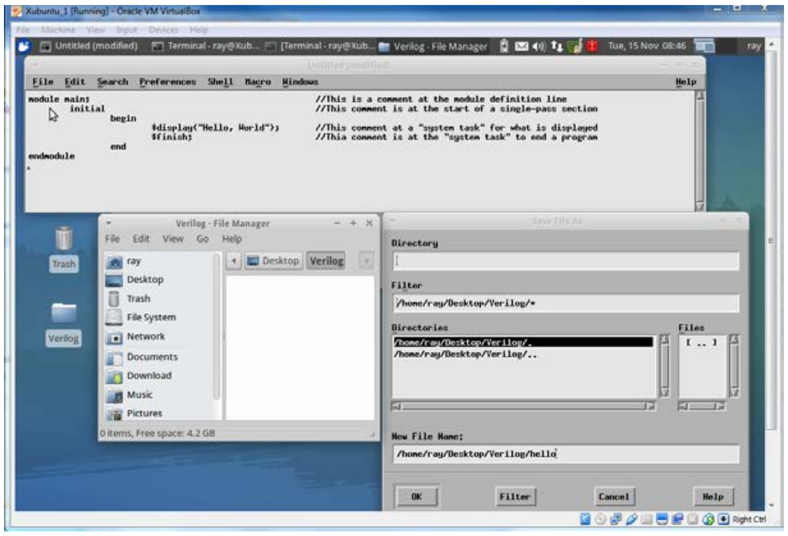
Figure 1.13 –Program files in the Verilog Folder in Windows (Command View)



Name	Date modified	Type
hello	10/24/2016 1:52 PM	File
hello	10/24/2016 1:39 PM	Verilog File

Figure 1.14 –Program files in the Verilog Folder in Windows (GUI View)

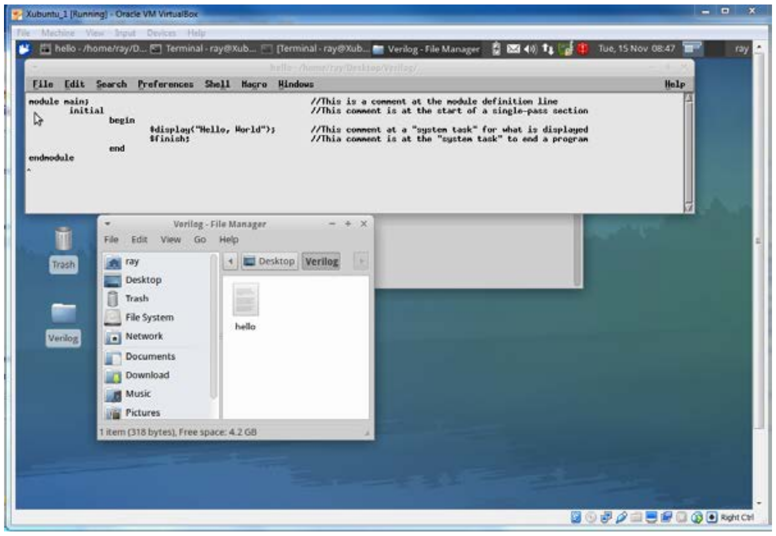
These tools and procedure are recommended because notepad++ performs rudimentary lexical analysis on your program and highlights keywords and the compiler performs syntax checking and refers to a line number in notepad++ where it identifies problems.



```
module main;                                     //This is a comment at the module definition line
        initial                                  //This comment is at the start of a single-pass section
                begin
                        $display("Hello, World");       //This comment at a "system task" for what is displayed
                        $finish;                        //This comment is at the "system task" to end a program
                end
endmodule
```

```
Directory

Filter
/home/ray/Desktop/Verilog/*
Directories                                      Files
/home/ray/Desktop/Verilog/.                      [ .. ]
/home/ray/Desktop/Verilog/..

New File Name:
/home/ray/Desktop/Verilog/hello
```

Figure 1.15 – Program Entry Using nedit in the Verilog Folder in Xubuntu Linux



```
module main;                                     //This is a comment at the module definition line
        initial                                  //This comment is at the start of a single-pass section
                begin
                        $display("Hello, World");       //This comment at a "system task" for what is displayed
                        $finish;                        //This comment is at the "system task" to end a program
                end
endmodule
```

Figure 1.16 – Saving the hello program in the Verilog Folder in Xubuntu Linux

Again, be aware that it is common to require several iterations, entering and editing the program before all syntax issues are resolved. It is also a good strategy to write small modules, name them uniquely, and compile them enough times to obtain syntactically correct code. Later, cut and paste techniques can be used to construct combinations of modules.

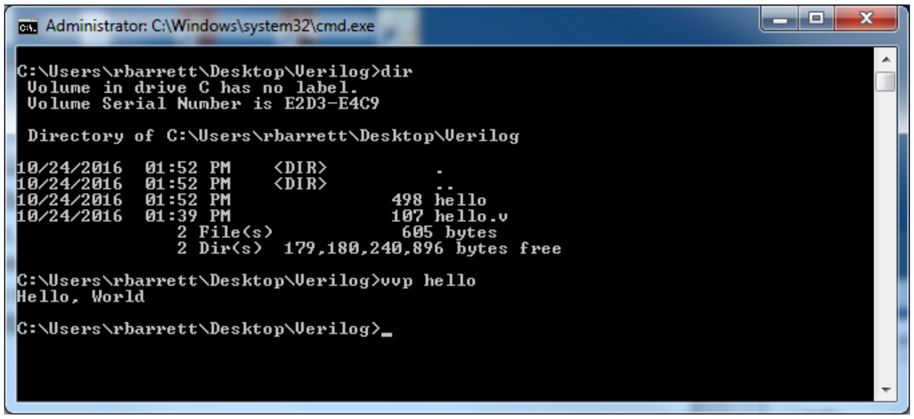
The Verilog language has many syntactical requirements like any computer language. Many universities employ Verilog as an integral part of their programs and support its use in their laboratories. One such program at the University of Maryland, Baltimore, supplies a very useful "help" index at:
https://www.csee.umbc.edu/portal/help/VHDL/verilog/summary.html#Top

As a language reference, purchase the IEEE standard:
http://ieeexplore.ieee.org/document/1620780/?reload=true&arnumber=1620780; or use the on-line version provided by UC Berkeley to help fix syntax errors.
https://inst.eecs.berkeley.edu/~cs150/fa06/Labs/verilog-ieee.pdf

If you get more deeply involved with Verilog, you may want to secure "The Verilog Hardware Description Language," by Thomas and Moorby; the authors of this course have found this book to be invaluable as a well-organized reference:
http://www.springer.com/us/book/9780387849300

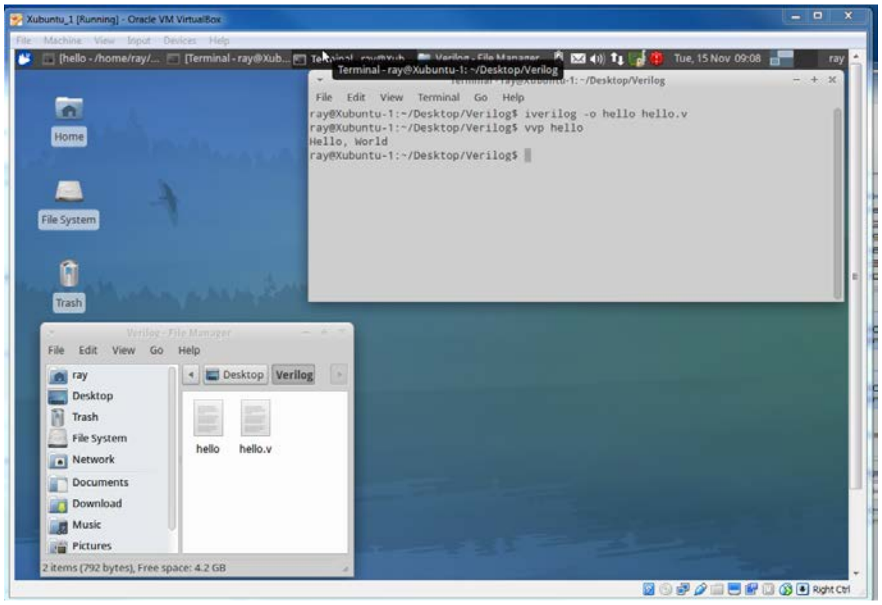
After compilation, you should see two indications that it has compiled correctly with the completion following the command above in the DOS command window (without error messages), and with the change of the Verilog window to include both the source hello.v and compiled program hello:

Once the hello program is compiled without errors, you must run the program by typing the command "**vvp hello**" as below:



```
C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

10/24/2016  01:52 PM    <DIR>          .
10/24/2016  01:52 PM    <DIR>          ..
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
               2 File(s)            605 bytes
               2 Dir(s)  179,180,240,896 bytes free

C:\Users\rbarrett\Desktop\Verilog>vvp hello
Hello, World

C:\Users\rbarrett\Desktop\Verilog>_
```

Figure 1.17 –Executing the hello Program in Windows (Command View)


Xubuntu_1 [Running] - Oracle VM VirtualBox
Terminal - ray@Xubuntu-1: ~/Desktop/Verilog
File Edit View Terminal Go Help
Tue, 15 Nov 09:08
Home
File System
Trash
Verilog - File Manager
ray
Desktop
Trash
File System
Network
Documents
Download
Music
Pictures
Verilog
hello
hello.v
2 items (792 bytes), Free space: 4.2 GB

```
ray@Xubuntu-1:~/Desktop/Verilog$ iverilog -o hello hello.v
ray@Xubuntu-1:~/Desktop/Verilog$ vvp hello
Hello, World
ray@Xubuntu-1:~/Desktop/Verilog$
```

Figure 1.18– Compiling and Executing the hello Program in Xubuntu Linux

You see the "Hello, World" response in the command window indicating that the message is a result of invoking the iverilog simulator to execute the hello program. Congratulations, you have just entered, compiled, and executed a Verilog program.

There are shortcut operations possible in Windows, but they essentially perform the same operations. For example, if you hold the shift-key and right-click on the Verilog folder containing the files on your desktop, you will be shown a list that includes the option to "open command window here" bypassing the selection from the Windows-R key and DOS command window as discussed above. Also, if repeated edits are needed to correct syntax errors, the arrow keys can be used to repeat prior instructions and avoid extra typing.

1.0.8 Icarus Verilog Program Structure – The first line of the "**hello**" program shown below in **Listing 1.1**, is the naming declaration for the entire program "**module**" and all programs must be declared in this way with a "**module**" declaration and they must end with the "**endmodule**" declaration. These are language "**keywords**" that identify the beginning and end of the program. The **"//"** character pair denotes a comment from the pair to the end of the line.

```
module main;                          //This is a comment at the module definition line
 initial                              //This comment is at the start of a single-pass section
  begin
   $display("Hello, World");          //This comment at a "system task" for what is displayed
   $finish ;                          //This comment is at the "system task" to end a program
  end
endmodule                             //This comment is the endmodule for the end of the listing
```

Program Listing 1.1

Next, we encounter an "**initial**" declaration that indicates a part of the program that executes sequentially just once, and includes all statements between the "**begin**" and "**end**" keywords.

As you might guess, the "**initial**" declaration is mostly used to initialize variables and do the program setup, but can be used, as we have, to execute the program one single time, in the way we do here.

In the body of the "**initial**" portion between the "**begin**" and "**end**" keywords are two system tasks **$display** and **$finish** that tell the system what to send to the display, and to end the system instructions. There are more such system tasks and they all begin with the **$** character.

As we shall see later, there are Integrated Development Environment (IDE) software tools that simplify the program entry, compilation, and execution, but setting these tools up is itself an involved process and deferred until the ideas of Verilog programming have become firmly established.

2.0 Verilog Logic Design Levels of Abstraction – The Verilog Language supports three general levels of abstraction: Behavioral, RTL, and Structural Levels. Verilog programs can employ these levels seamlessly but there are attributes to each level that are best employed to each level's advantage.

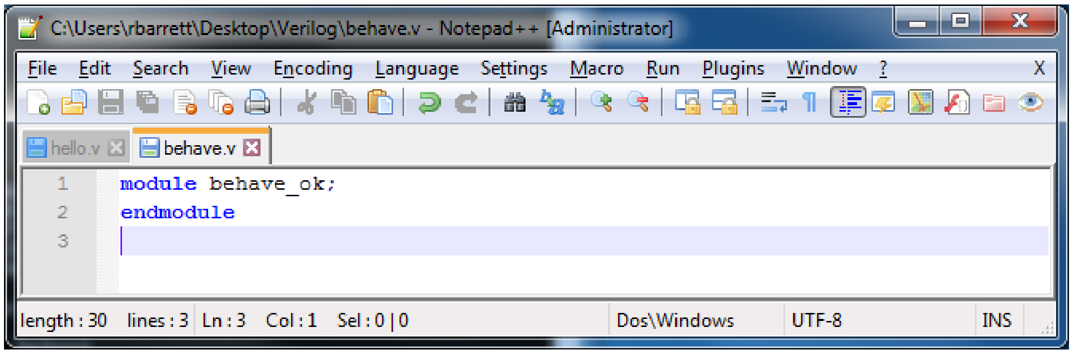
There are also tools available that can translate between levels of abstraction with differing degrees of success. One such tool is a Verilog synthesis tool that produces an output file describing the digital logic suitable for Field Programmable Gate Array (FPGA) or Application Specific IC (ASIC) implementation. Synthesis is beyond the scope of this course but the reader is encouraged to follow an interest deeper into the subject.

2.1 Verilog Logic Design Behavioral Level of Abstraction – Designs at the Behavioral Level of abstraction support design-capture of system behavior, as the name states, but has little or no implementation information. The Behavioral Level of abstraction is closest to a general-purpose programming language and can be used to capture features without bothering with detail. One such usage is the construction of Test-Bench programs that provide simulation behaviors for external systems to determine how a more detailed design will function.

One price of the flexibility and generalized functionality of a Behavioral Level model is that it is rarely "synthesizable" using a synthesis programming tool to convert the program to a more detailed expression. Progress has been made in behavioral system synthesis, but generally Behavioral Level cannot reliably be synthesized to an implementable level.

We will explore an example of a Behavioral Level model by generating a "**Clock**" waveform and displaying the result using the GTKWave tool. We will expand that example, also as a Behavioral Level model by generating other waveforms in a similar fashion.

A Behavioral Level program must still follow the syntax rules of the Verilog language and have the module and endmodule declarations, as well as a module name. An example that shows this minimum requirement is:

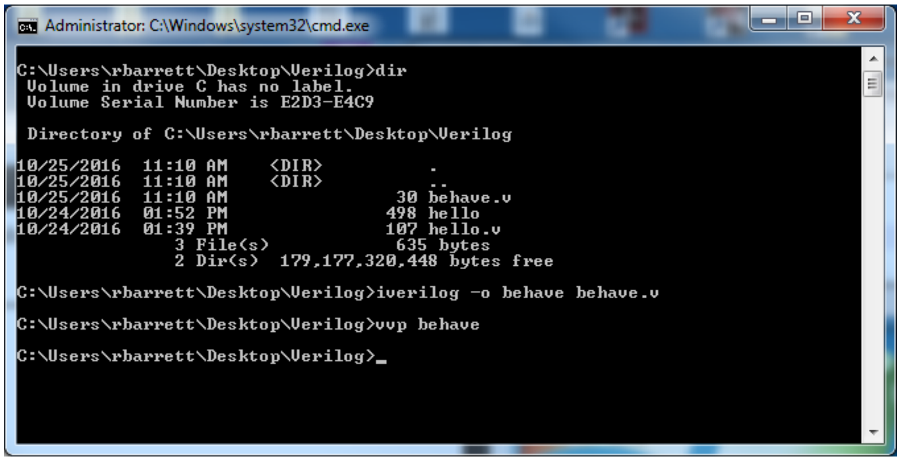

C:\Users\rbarrett\Desktop\Verilog\behave.v - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hello.v
behave.v

```
module behave_ok;
endmodule
```

length : 30 lines : 3 Ln : 3 Col : 1 Sel : 0 | 0
Dos\Windows
UTF-8
INS

Figure 2.0 – Entering the behave Program in notepad++ in Windows

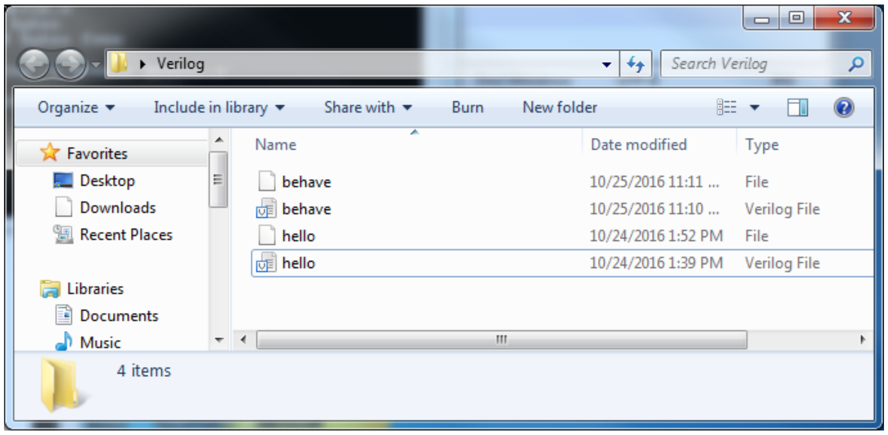
We follow the steps outlined above for program entry, compilation, and execution, and obtain:



```
C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

10/25/2016  11:10 AM    <DIR>          .
10/25/2016  11:10 AM    <DIR>          ..
10/25/2016  11:10 AM                30 behave.v
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
               3 File(s)            635 bytes
               2 Dir(s)  179,177,320,448 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o behave behave.v

C:\Users\rbarrett\Desktop\Verilog>vvp behave

C:\Users\rbarrett\Desktop\Verilog>_
```

Figure 2.1 – Entering, Compiling, and Executing behave in Windows (Command)

And in the Verilog folder, we have the files:

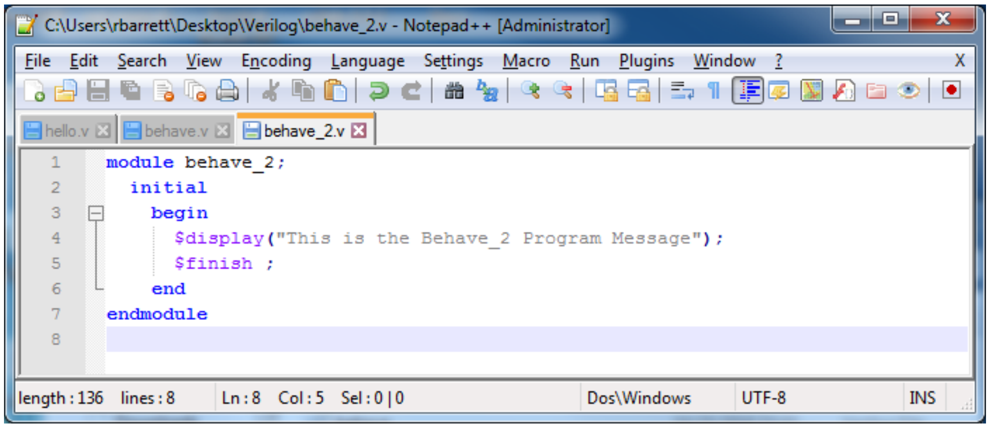


Name	Date modified	Type
behave	10/25/2016 11:11 ...	File
behave	10/25/2016 11:10 ...	Verilog File
hello	10/24/2016 1:52 PM	File
hello	10/24/2016 1:39 PM	Verilog File

Figure 2.2 – Entering and Compiling behave in Windows Verilog (GUI)

But, following the "**vvp behave**" command, there is no message or apparent action because our minimum correct program does nothing, as it was instructed to do (not do?).

We can add a message, just as we did in the "**Hello, World**" program with:

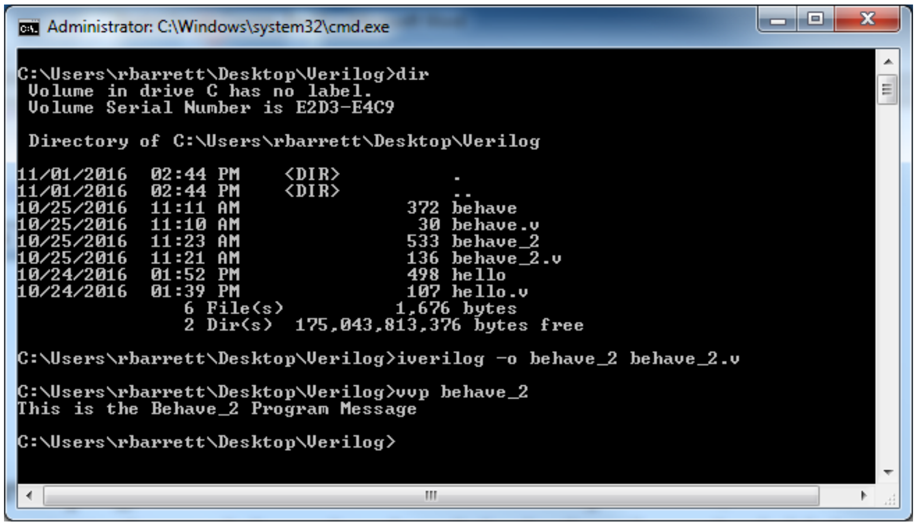

C:\Users\rbarrett\Desktop\Verilog\behave_2.v - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hello.v
behave.v
behave_2.v

```
module behave_2;
  initial
    begin
      $display("This is the Behave_2 Program Message");
      $finish ;
    end
endmodule
```

length : 136 lines : 8 Ln : 8 Col : 5 Sel : 0 | 0 Dos\Windows UTF-8 INS

Figure 2.3 – Editing the behave_2 Program in notepad++ in Windows

And obtain the expected result:



```
C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/01/2016  02:44 PM    <DIR>          .
11/01/2016  02:44 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
10/25/2016  11:23 AM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
               6 File(s)          1,676 bytes
               2 Dir(s)  175,043,813,376 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o behave_2 behave_2.v

C:\Users\rbarrett\Desktop\Verilog>vvp behave_2
This is the Behave_2 Program Message

C:\Users\rbarrett\Desktop\Verilog>
```

Figure 2.4 –Compiling and Executing behave_2 in Windows (Command)

We now have three Behavioral Level programs that have been successfully entered, compiled, and executed and demonstrate the module and endmodule declarations, the initial control construct declaration, the system tasks **$display** and **$finish**, and the practice of punctuation with semicolons at the **module** declaration and the system tasks, but not at the **initial** declaration or **begin** or **end** statements.

We recommend that you become familiar with the program entry using the notepad++ text editor, and the iverilog compiler steps repetitively because they work together to perform a first-level of syntax checking for you.

Some other constructs that can be employed in a Behavioral Level program include the **always** control construct declaration. **Always** is distinguished from the **initial** control construct declaration because its procedural body is expected to execute at least once but may repeat indefinitely, whereas the **initial** control construct declaration executes its procedural body (usually between begin & end delimiters) only once.

We could continue a discussion of the procedural uses of the **always** control construct declaration but is cleared in a context with time-dependent waveforms and those are best left until after the **GTKWave** viewer is introduced.

The Behavioral Level program structures are often used to construct "**Test Bench**" programs that invoke other modules. We use this approach to illustrate both **RTL** and **Structural Level** programs in the following discussions.

2.2 Verilog Logic Design RTL Level of Abstraction – The **RTL Level of Abstraction** takes its name from the use of data register declarations and the manipulations performed on the contents of those registers.

With suitable restrictions on the data constructs and operations, the RTL programs can be synthesizable to logic-level implementations. Some target implementations include discrete devices at a board level, Field-Programmable Gate Array (**FPGA**) devices, Application-Specific Integrated Circuit (**ASIC**) custom devices, embedded "**C**" programs in devices using the Verilator (http://www.veripool.org/projects/verilator) translator, as well as other discrete circuit applications that are possible.

Portability of designs is one very important consideration supported by RTL designs. It is common, for example, to target the same design to several of the technologies mentioned, as well as different generations of technologies. Some products are introduced to a market as an FPGA for early introduction while a full-custom ASIC is being fabricated. The FPGA and ASIC use the same program Verilog code despite the target implementation differences, especially when early introduction has important time-to-market economic impacts.

For those designs that are implemented in logic gates, wires and transistors, a suitably more detailed expression is the **Structural Level of Abstraction**. The higher level of abstraction of RTL, combined with synthesis software, makes the design process more robust, more compact, and easier for a human to read. Sometimes, a combination of RTL with a Structural Level library of cell modules is used to support ASIC implementations.

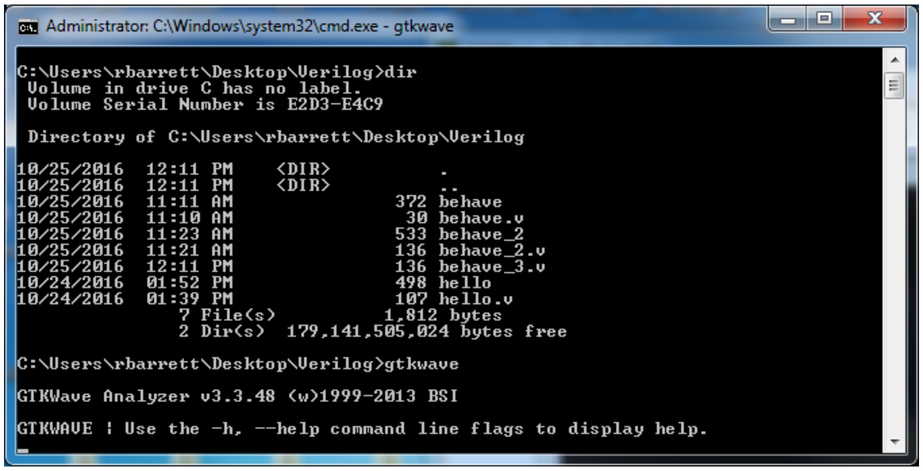
2.3 Verilog Logic Design Structural Level of Abstraction – The Structural Level of Abstraction takes its name from the logic structures supported by Verilog; the gates, devices, and their interconnections. Structural level programs are offer less portable programs than synthesizable RTL, but may be used for design capture for concepts that are best expressed at the gate and transistor level. Detailed timing behaviors can be obtained from structural level cell module circuits that must be estimated at higher levels. Those cell modules are "characterized" for important power and delay parameters that are appended and available to the RTL synthesis software for speed/power tradeoff decisions.

It is a common practice to design implementation-specific ASIC cell-libraries of logic gate cells at the structural level and characterize their behavior as attachments to higher-level cell descriptions. For instance, delays are often obtained from other environments (analog SPICE simulation is common) and attached as "Standard Delay Format" (.sdf files) accessed by synthesis tools. Re-targeting Verilog RTL designs from FPGA to an ASIC technology is important, but also to support advances in IC technology between ASIC generations and implementation foundry suppliers.

In a sense, the synthesis tool substitutes the structural level information into a higher level expression to obtain timing and power dissipation estimates at that higher level. Substitutions and variations in the synthesis operations support optimization of designs in a target technology. One common approach offers a trade-off of speed performance for power performance while retaining common core functionality for technology comparisons.

3.0 Introduction to Verilog Logic Design Waveform Viewing – To enter a Verilog program, and compile it to obtain the executable form, we have used the notepad++ text editor and the Iverilog compiler. For viewing digital design results, we add another tool that displays time-domain digital waveform information produced by the executable Verilog files.

When you installed your Icarus Verilog, the GTKWave program may have been included in the installer. You can test to see if it is present by opening the command window and typing the "gtkwave" command:


Administrator: C:\Windows\system32\cmd.exe - gtkwave

```
C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

10/25/2016  12:11 PM    <DIR>          .
10/25/2016  12:11 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
10/25/2016  11:23 AM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
10/25/2016  12:11 PM               136 behave_3.v
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
               7 File(s)          1,812 bytes
               2 Dir(s)  179,141,505,024 bytes free

C:\Users\rbarrett\Desktop\Verilog>gtkwave

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

GTKWAVE | Use the -h, --help command line flags to display help.
_
```

Figure 3.0 – Invoking GTKWave in Windows (Command Window Test)

The command window above shows that GTKWave is installed and opens its interface as shown below:

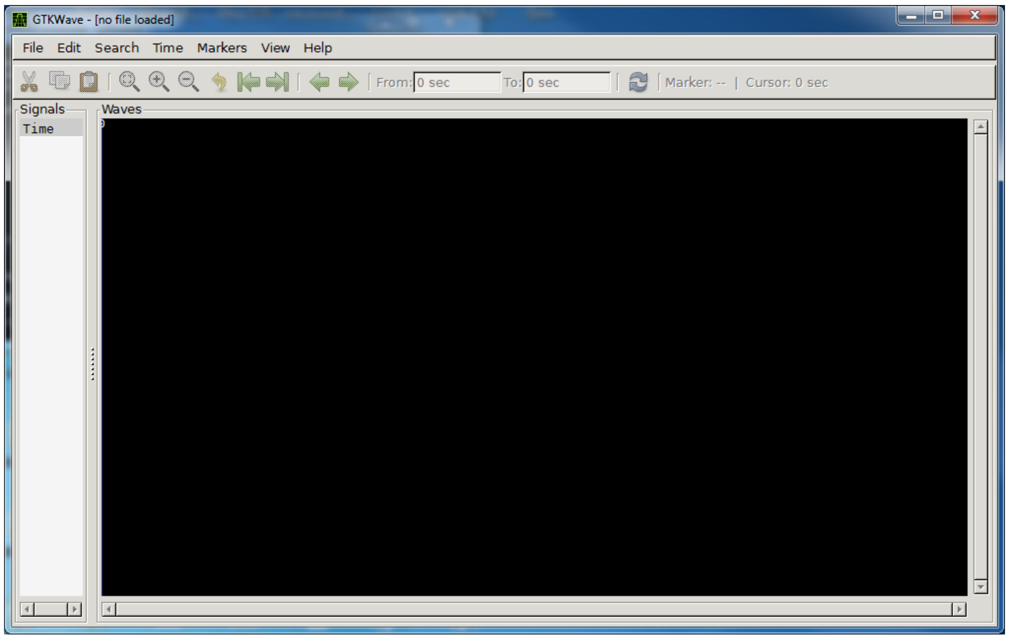

GTKWave - [no file loaded]
File Edit Search Time Markers View Help
From: 0 sec
To: 0 sec
Marker: -- | Cursor: 0 sec
Signals
Time
Waves

Figure 3.1 – GTKWave Waveform Viewer GUI in Windows

If the tool is not already installed, an installer can be obtained at: http://gtkwave.soft112.com/

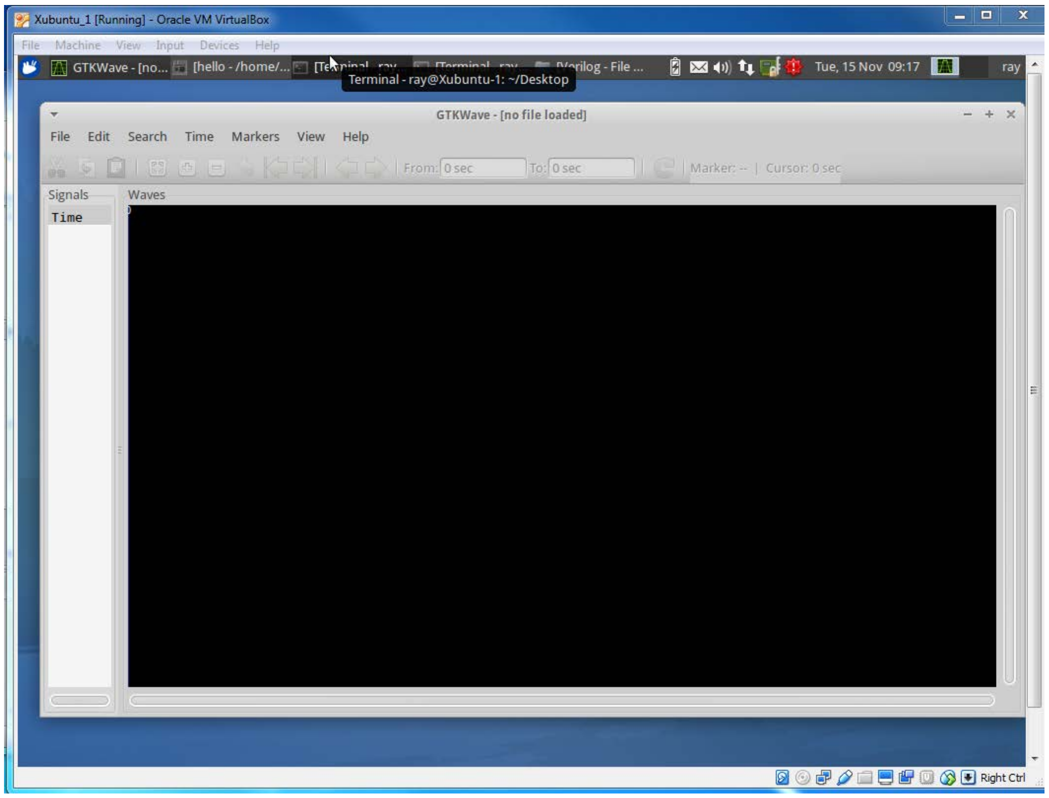
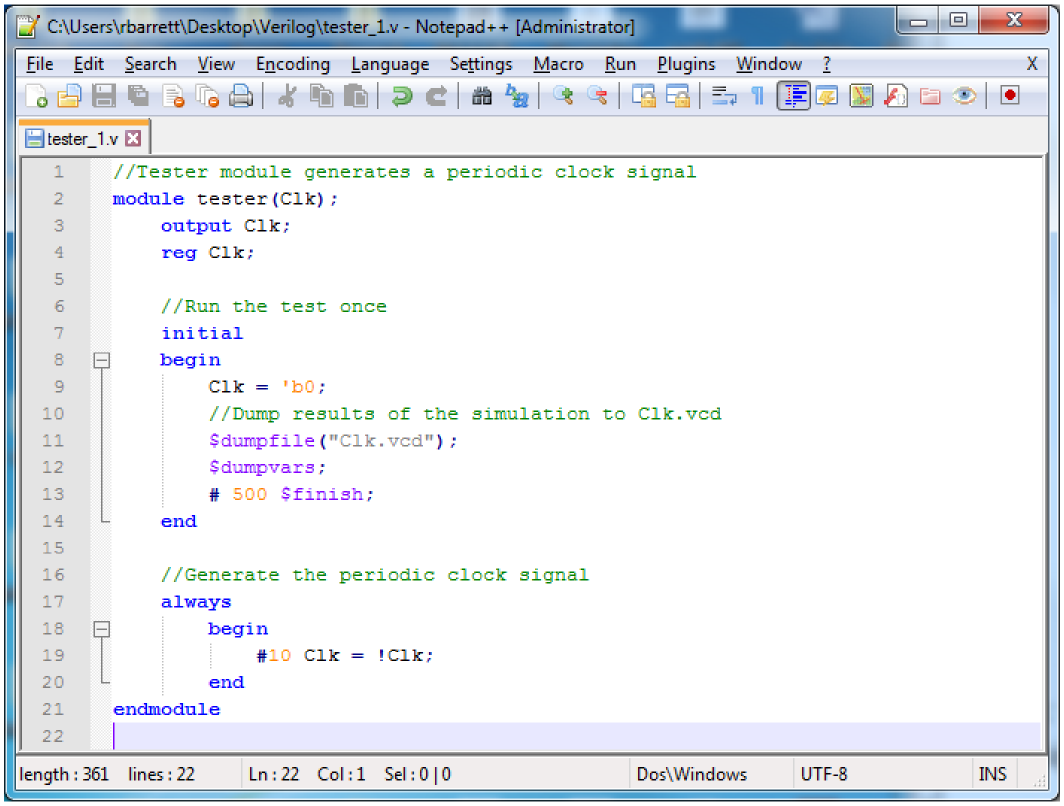

Xubuntu_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
GTKWave - [no...
[hello - /home/...
Terminal - ray@Xubuntu-1: ~/Desktop
Tue, 15 Nov 09:17
ray
GTKWave - [no file loaded]
File Edit Search Time Markers View Help
From: 0 sec
To: 0 sec
Marker: -- | Cursor: 0 sec
Signals
Time
Waves
Right Ctrl

Figure 3.2 – GTKWave Waveform Viewer GUI in Xubuntu Linux

3.1 Verilog Logic Design Waveform Viewing Example – You can find a detailed “GTK Wave User’s Guide at: http://gtkwave.sourceforge.net/gtkwave.pdf for additional reference, and we present a behavioral level program then showing the clock waveform in GTKWave.

We begin by defining the program module tester_1 as follows:


C:\Users\rbarrett\Desktop\Verilog\tester_1.v - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
tester_1.v

```
//Tester module generates a periodic clock signal
module tester(Clk);
    output Clk;
    reg Clk;

    //Run the test once
    initial
    begin
        Clk = 'b0;
        //Dump results of the simulation to Clk.vcd
        $dumpfile("Clk.vcd");
        $dumpvars;
        # 500 $finish;
    end

    //Generate the periodic clock signal
    always
        begin
            #10 Clk = !Clk;
        end
endmodule

```

length : 361 lines : 22 Ln : 22 Col : 1 Sel : 0 | 0 Dos\Windows UTF-8 INS

Figure 3.3 – Editing the tester_1 Program in notepad++ in Windows

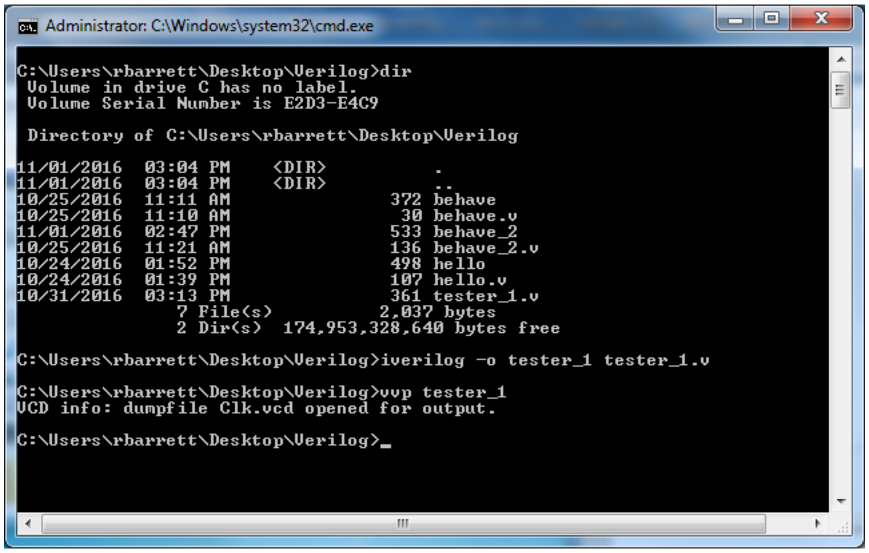
Note: we included lines in the always block near the end of the module to define operation of the periodic **Clk** signal, and we have also modified the initial section. At the top, we have a new module name "**tester**," and we will save under the filename **tester_1** to help distinguish the file from its contents.

The module tester has an output terminal defined by the "**tester(Clk);**", the "**output (Clk);**" and the "**reg Clk;**" declarations. We have associated the **Clk** with tester, defined it as an **output** signal, and retained its value in a single-bit "**Clk**" register.

To retain the simulation waveform data, we use a system command "**$dmpfile("Clk.vcd);**" to tell the compiler which file to use for the data, and then **"$dumpvars**;" to actually write the simulation results into that file.

After we issue the command to write to the file, we use the "**#500 $finish;**" instructions to delay, then finish the program. The delay is added so that the program executes for the non-zero duration of 500 arbitrary unit-delay time steps. We will see the result on the GTKWave display.

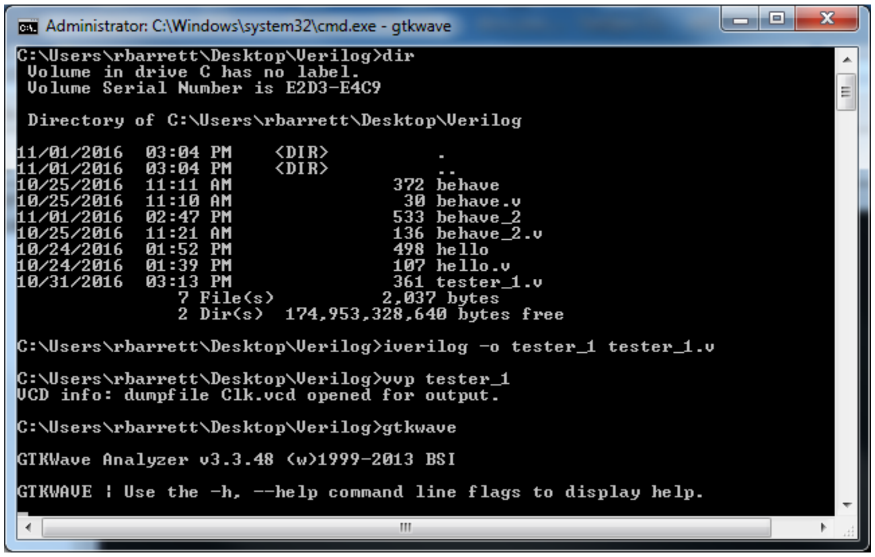
We use a similar delay in the "**#10 Clk = !Clk;**" so that the **Clk** signal is delayed 10 units and then complemented to its logic inverse. The implication from this structure is that we should see a clock with a period of 20 units (10 high = 1, and 10 low = 0) repeating until we reach 500 units (500/20 = 25 clock cycles).



```
C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/01/2016  03:04 PM    <DIR>          .
11/01/2016  03:04 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
11/01/2016  02:47 PM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
10/31/2016  03:13 PM               361 tester_1.v
               7 File(s)          2,037 bytes
               2 Dir(s)  174,953,328,640 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o tester_1 tester_1.v

C:\Users\rbarrett\Desktop\Verilog>vvp tester_1
VCD info: dumpfile Clk.vcd opened for output.

C:\Users\rbarrett\Desktop\Verilog>_
```

Figure 3.4 –Compiling and Executing tester_1 in Windows (Command)

Again, we compile and execute the program with the **vvp** command so that the Clk.vcd file is written, and invoke the GTKWave program as follows:



```
Administrator: C:\Windows\system32\cmd.exe - gtkwave
C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/01/2016  03:04 PM    <DIR>          .
11/01/2016  03:04 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
11/01/2016  02:47 PM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
10/31/2016  03:13 PM               361 tester_1.v
               7 File(s)          2,037 bytes
               2 Dir(s)  174,953,328,640 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o tester_1 tester_1.v

C:\Users\rbarrett\Desktop\Verilog>vvp tester_1
VCD info: dumpfile Clk.vcd opened for output.

C:\Users\rbarrett\Desktop\Verilog>gtkwave

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

GTKWAVE | Use the -h, --help command line flags to display help.
```

Figure 3.5 – Invoking GTKWave in Windows (Command Window Test)

The gtkwave window opens and we get an empty screen because the program does not yet have any data loaded:

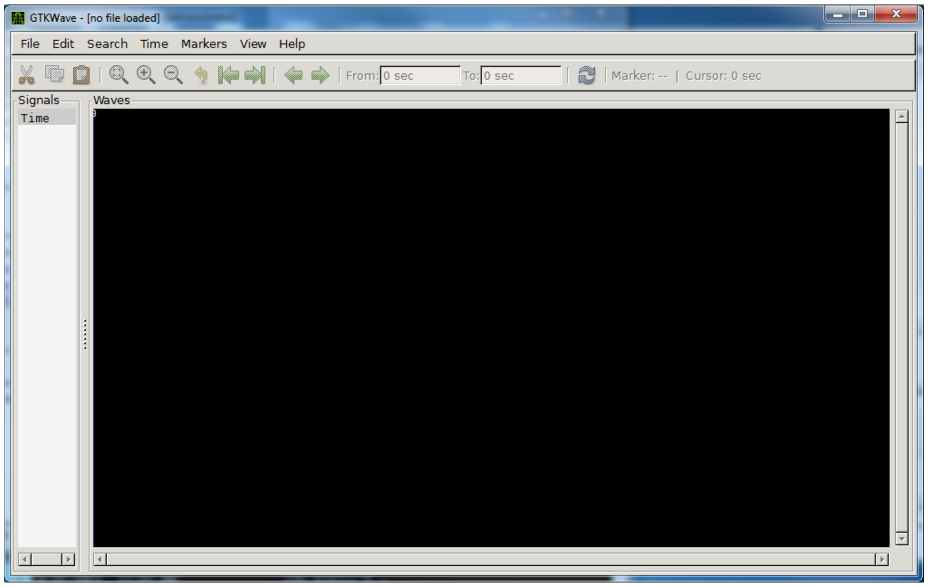

GTKWave - [no file loaded]
File Edit Search Time Markers View Help
From: 0 sec
To: 0 sec
Marker: --
Cursor: 0 sec
Signals
Time
Waves

Figure 3.6 – Empty GTKWave Waveform Viewer GUI in Windows

Under the File header, we select the Open New Tab, get the wave selection from the list as **Clk.vcd**:

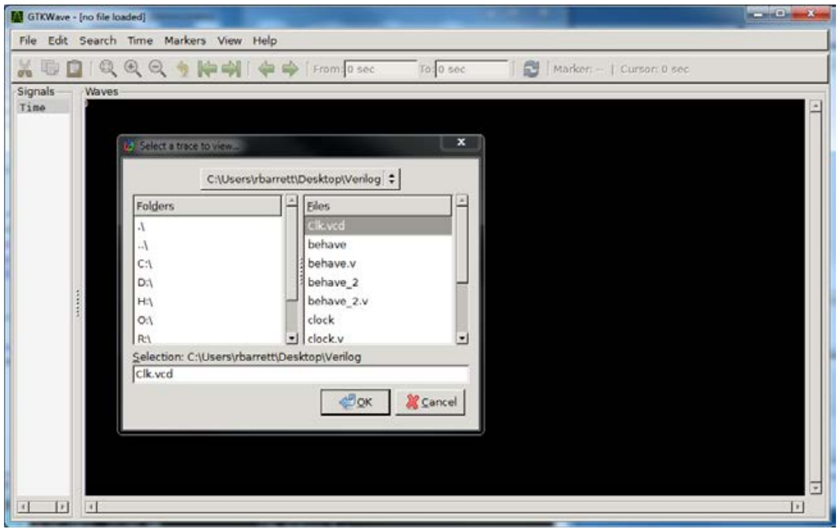

GTKWave - [no file loaded]
File Edit Search Time Markers View Help
From: 0 sec
To: 0 sec
Marker: -- | Cursor: 0 sec
Signals
Time
Waves
Select a trace to view...
C:\Users\rbarrett\Desktop\Verilog
Folders
.\
..\
C:\
D:\
H:\
O:\
R:\
Files
Clk.vcd
behave
behave.v
behave_2
behave_2.v
clock
clock.v
Selection: C:\Users\rbarrett\Desktop\Verilog
Clk.vcd
OK
Cancel

Figure 3.7 – Opening New Tab GTKWave Waveform Viewer GUI in Windows

After we select OK, we now have another selection with the data from the tester_1 program module loaded. We see that the time at the top is from 0 sec to 500 sec, as we expected. :

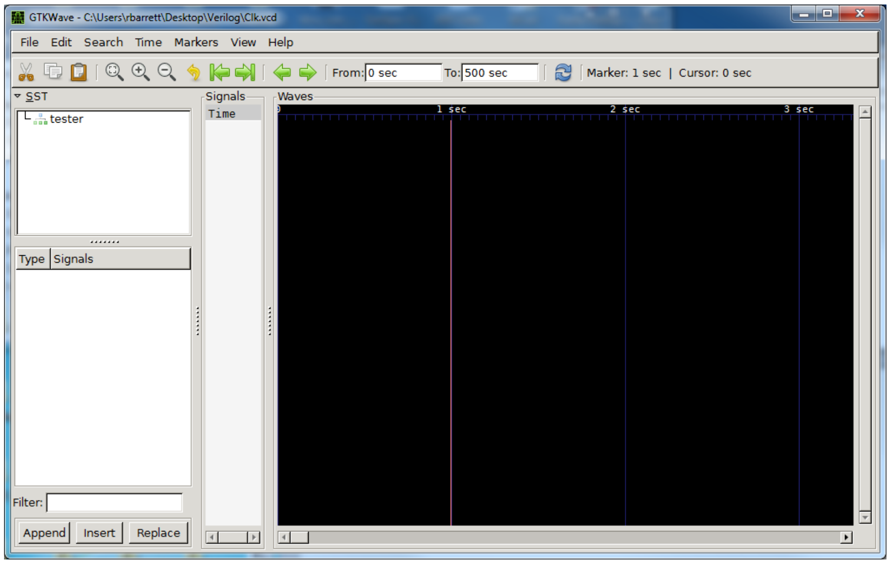

GTKWave - C:\Users\rbarrett\Desktop\Verilog\Clk.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 500 sec
Marker: 1 sec | Cursor: 0 sec
SST
tester
Type Signals
Signals
Time
Waves
1 sec
2 sec
3 sec
Filter:
Append
Insert
Replace

Figure 3.8 – Open tester data in GTKWave Waveform Viewer GUI in Windows

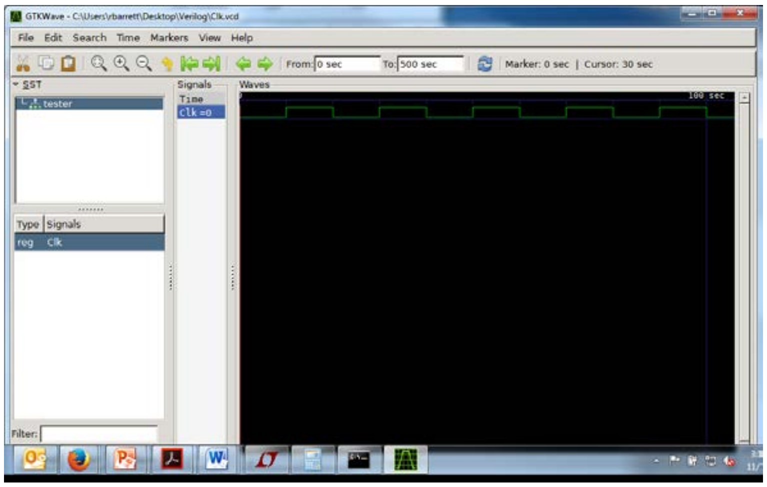

GTKWave - C:\Users\rbarrett\Desktop\Verilog\Clk.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 500 sec
Marker: 0 sec | Cursor: 30 sec
SST
tester
Type Signals
reg Clk
Signals
Time
Clk =0
Waves
100 sec
Filter:

Figure 3.9 – Clock waveform data in GTKWave GUI in Windows

We select the waveforms by clicking on "tester" and drag the signal "**Clk**" onto the Time pane. At first, the waveform appears blank, but clicking on the "Time Zoom Out" several times, we get the display above.

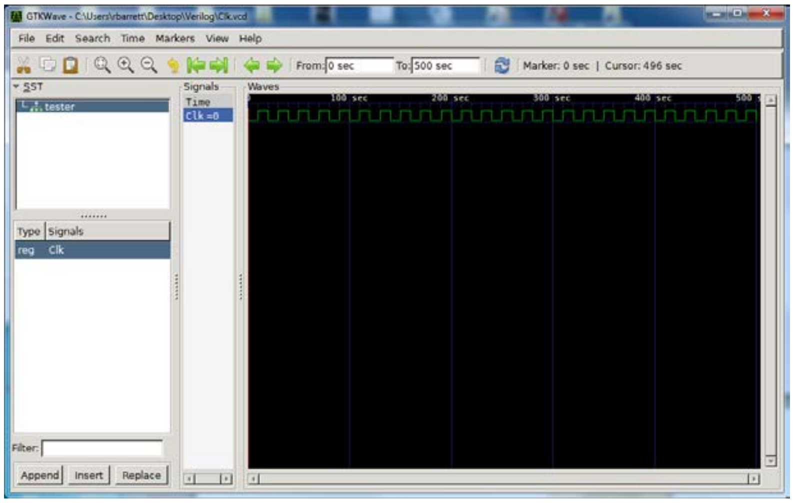


Figure 3.10 – Clock waveform full data in GTKWave GUI in Windows

Time "Zoom Full" produces the entire 500 second interval of 25 cycles.

3.2 Another Verilog Logic Design Waveform Viewing Example – We note that the first example defines a simple Behavioral Level repetitive clock waveform. We expand the single clock signal by adding three more clock definitions; Clk1, Clk2, and Clk3 in similar fashion to the existing definitions, each as an **output**, and each as a "**reg**" variable..

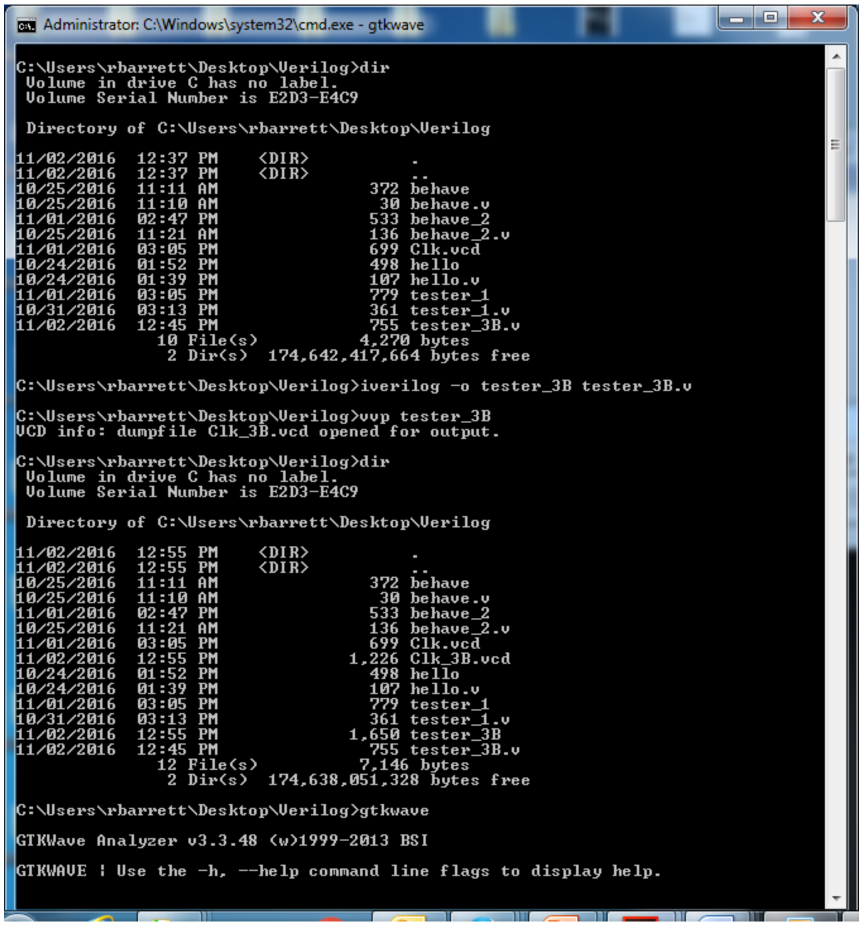
We rename the module text-file as **tester_3B**, as well as the module name, too. As the program listing continues on the next page, you will see that we have re-named the "dumpfile," also as "**Clk_3B.vcd**", then added event related sections to the "**always**" that relate to the "**posedge**" transitions on each preceding clock signal.

```
module tester_3B(Clk, Clk1, Clk2, Clk3); //Tester module generates multiple periodic clock signals

        output Clk, Clk1, Clk2, Clk3;
        reg Clk, Clk1, Clk2, Clk3;

        initial                                  //Run the test once
        begin
                Clk = 'b0;
                Clk1 = 'b0;
                Clk2 = 'b0;
                Clk3 = 'b0;
                $dumpfile("Clk_3B.vcd");         //Dump results of the simulation to Clk_3B.vcd
                $dumpvars;
                # 500 $finish;
        end

        always                                  //Generate the original periodic clock signal
                begin
                        #10 Clk = !Clk;
                end

        always @(posedge Clk)                   // Generate event related periodic clocking
                begin
                        #2 Clk1 = !Clk1;
                end
        always @(posedge Clk1)
                begin
                        #2 Clk2 = !Clk2;
                end
        always @(posedge Clk2)
                begin
                        #2 Clk3 = !Clk3;
                end
endmodule
```

Program Listing 3.0

After the text-file is saved, we open the command window, show the files listed, noting that the "**tester_3B.v**" file is present, compile the file to the "**tester_3B**" file, and issue the "**vvp tester_3B**" to obtain the "**Clk_3B.vcd**" results file. We ensure that they are all present by obtaining another "**dir**" listing and then invoke the viewer using the "**gtkwave**" command.

You will see the results below, followed by the GTKWave window after the file is opened as a new tab, the **tester_3B** project is "clicked," and the reg type signals **Clk**, **Clk1**, **Clk2**, and **Clk3** are "dragged" into the signals Time pane. As before, you will need to Zoom Full to see the entire duration as it appears in the illustration below.



```
Administrator: C:\Windows\system32\cmd.exe - gtkwave

C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/02/2016  12:37 PM    <DIR>          .
11/02/2016  12:37 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
11/01/2016  02:47 PM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
11/01/2016  03:05 PM               699 Clk.vcd
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
11/01/2016  03:05 PM               779 tester_1
10/31/2016  03:13 PM               361 tester_1.v
11/02/2016  12:45 PM               755 tester_3B.v
              10 File(s)          4,270 bytes
               2 Dir(s)  174,642,417,664 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o tester_3B tester_3B.v

C:\Users\rbarrett\Desktop\Verilog>vvp tester_3B
VCD info: dumpfile Clk_3B.vcd opened for output.

C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/02/2016  12:55 PM    <DIR>          .
11/02/2016  12:55 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
11/01/2016  02:47 PM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
11/01/2016  03:05 PM               699 Clk.vcd
11/02/2016  12:55 PM             1,226 Clk_3B.vcd
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
11/01/2016  03:05 PM               779 tester_1
10/31/2016  03:13 PM               361 tester_1.v
11/02/2016  12:55 PM             1,650 tester_3B
11/02/2016  12:45 PM               755 tester_3B.v
              12 File(s)          7,146 bytes
               2 Dir(s)  174,638,051,328 bytes free

C:\Users\rbarrett\Desktop\Verilog>gtkwave

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

GTKWAVE | Use the -h, --help command line flags to display help.
```

Figure 3.11 –Compiling and Executing tester_3B in Windows (Command)

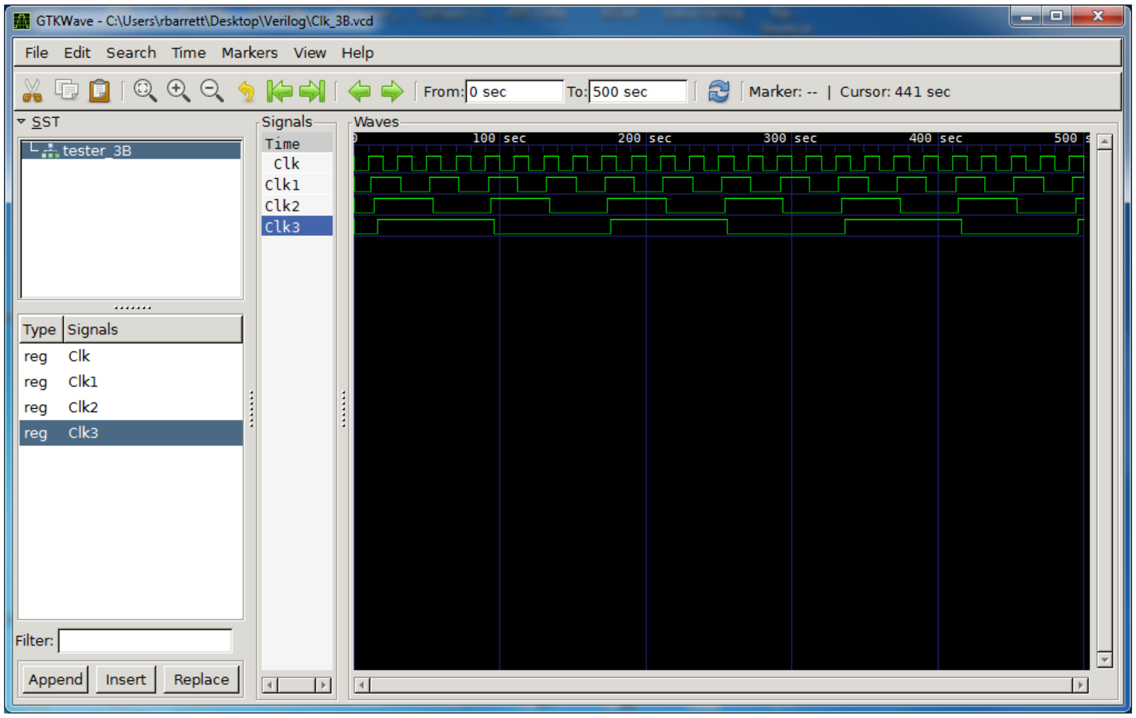


Figure 3.12 – Clock waveform full data in GTKWave GUI tester_3B in Windows

We note that we have the original clock signal and three other signals derived from it. All the signals are Behavioral Level representations and cannot be converted to an implementation using synthesis software tools.

We next build an RTL model with three similar clocking signals derived from the timing reference that provides similar results but is amenable to using synthesis software tools.

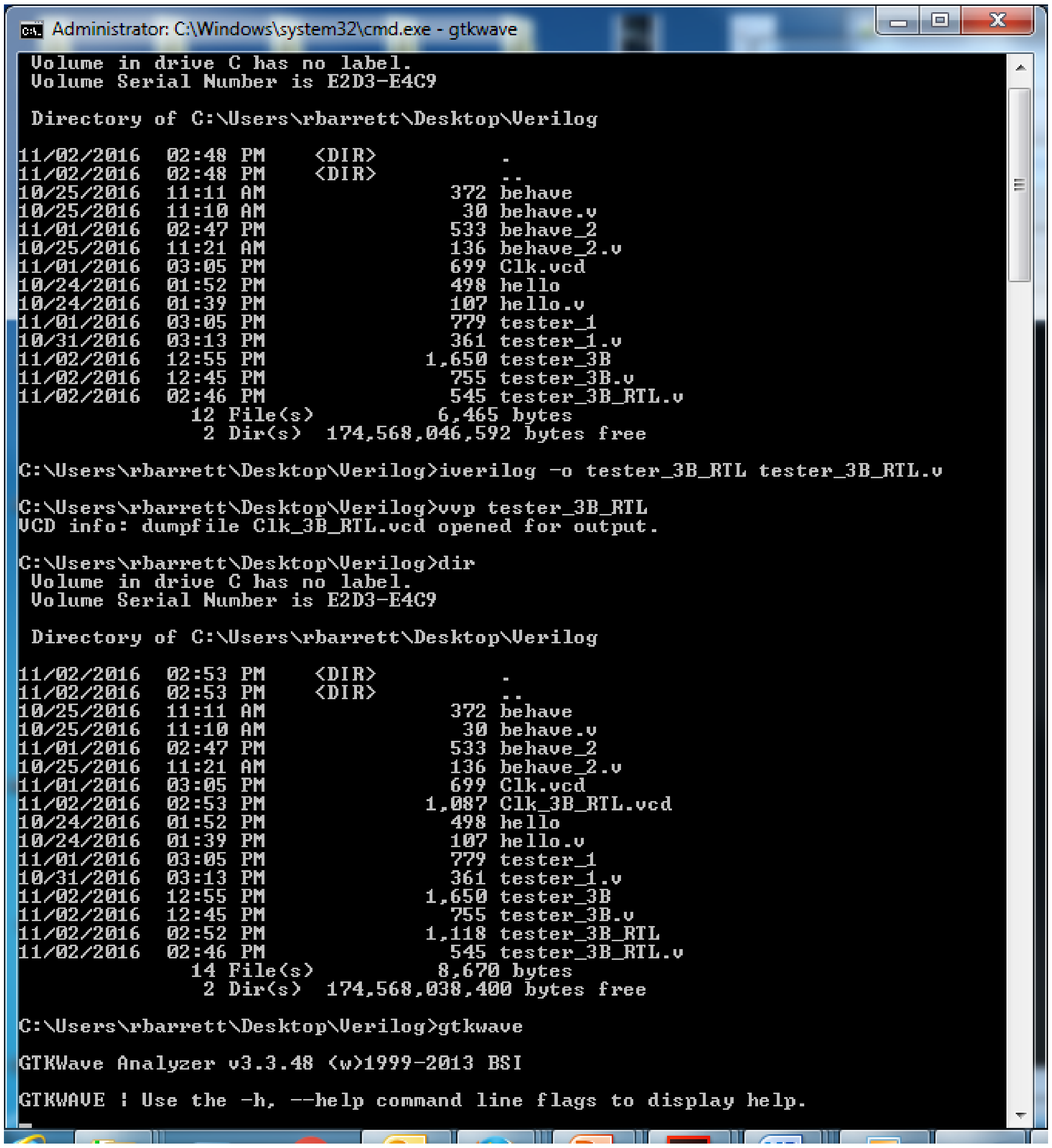
3.2 An RTL Verilog Logic Design Waveform Viewing Example – We note that we added the three clock waveforms in three "always" constructs. We revisit that program listing and modify its structure using a single register variable with three binary bits. We make that variable count over the range of zero to seven in binary fashion with the result that each register bit is equivalent to one of the prior clock signals.

Use the text editor to enter and save the following file under the "**tester_3B_RTL**" name.

```
//Tester module generates multiple periodic clock signals
module tester_3B_RTL(Clk);
        output Clk

        reg Clk;
        reg [2:0] temp;

        //Run the test once
        initial
        begin
                Clk = 'b0;
                temp = 3'b0;

                //Dump results of the simulation to Clk.vcd
                $dumpfile("Clk_3B_RTL.vcd");
                $dumpvars;
                # 500 $finish;
        end

        //Generate the original periodic clock signal
        always
                begin
                        #10 Clk = !Clk;
                end
        // Event related periodic RTL register clocking
        always @(posedge Clk)
                begin
                        #2 temp = temp + 1;
                end
endmodule
```

Program Listing 3.1

As before, we get similar results from the listings, compile, **vvp**, and listings again before the **gtkwave** command to invoke the viewer:



```
Administrator: C:\Windows\system32\cmd.exe - gtkwave
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/02/2016  02:48 PM    <DIR>          .
11/02/2016  02:48 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
11/01/2016  02:47 PM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
11/01/2016  03:05 PM               699 Clk.vcd
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
11/01/2016  03:05 PM               779 tester_1
10/31/2016  03:13 PM               361 tester_1.v
11/02/2016  12:55 PM             1,650 tester_3B
11/02/2016  12:45 PM               755 tester_3B.v
11/02/2016  02:46 PM               545 tester_3B_RTL.v
              12 File(s)          6,465 bytes
               2 Dir(s)  174,568,046,592 bytes free

C:\Users\rbarrett\Desktop\Verilog>iverilog -o tester_3B_RTL tester_3B_RTL.v

C:\Users\rbarrett\Desktop\Verilog>vvp tester_3B_RTL
VCD info: dumpfile Clk_3B_RTL.vcd opened for output.

C:\Users\rbarrett\Desktop\Verilog>dir
 Volume in drive C has no label.
 Volume Serial Number is E2D3-E4C9

 Directory of C:\Users\rbarrett\Desktop\Verilog

11/02/2016  02:53 PM    <DIR>          .
11/02/2016  02:53 PM    <DIR>          ..
10/25/2016  11:11 AM               372 behave
10/25/2016  11:10 AM                30 behave.v
11/01/2016  02:47 PM               533 behave_2
10/25/2016  11:21 AM               136 behave_2.v
11/01/2016  03:05 PM               699 Clk.vcd
11/02/2016  02:53 PM             1,087 Clk_3B_RTL.vcd
10/24/2016  01:52 PM               498 hello
10/24/2016  01:39 PM               107 hello.v
11/01/2016  03:05 PM               779 tester_1
10/31/2016  03:13 PM               361 tester_1.v
11/02/2016  12:55 PM             1,650 tester_3B
11/02/2016  12:45 PM               755 tester_3B.v
11/02/2016  02:52 PM             1,118 tester_3B_RTL
11/02/2016  02:46 PM               545 tester_3B_RTL.v
              14 File(s)          8,670 bytes
               2 Dir(s)  174,568,038,400 bytes free

C:\Users\rbarrett\Desktop\Verilog>gtkwave

GTKWave Analyzer v3.3.48 (w)1999-2013 BSI

GTKWAVE | Use the -h, --help command line flags to display help.
```

Figure 3.13 –Compiling and Executing tester_3B_RTL in Windows (Command)

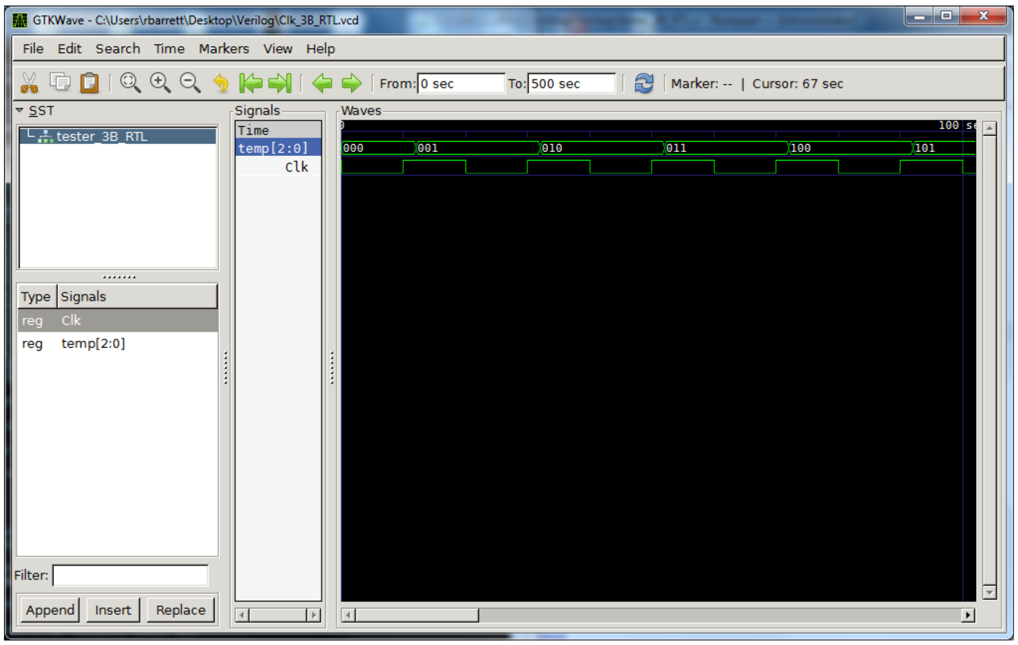


Figure 3.14 – Clock waveform in GTKWave GUI tester_3B_RTL in Windows

With GTKWave opened, the "**Clk_3B_RTL.vcd**" file loaded, the **tester_3B_RTL** project module selected, and both signals dragged to the Time pane, you will see traces, but you should select the **temp[2:0]** signal with the right mouse button and make the data format **binary**. You will see all three bits each with their waveforms changing following the rising edge of the **Clk** variable. **Temp[0]** is equivalent to the prior Behavioral **Clk1** signal, **Temp[1]** is equivalent to the prior Behavioral **Clk2** signal, and **Temp[2]** is equivalent to the prior Behavioral **Clk3** signal, without the additional delays. We could make explicit assignments with delays if we wish. The major point of this exercise is the procedure similarity of the tool use and the changes to generate the RTL Level portion of the program.

3.3 A Structural Verilog Logic Design Waveform Viewing Example – We note the similarity of forming three related clock waveforms using Behavioral Level and RTL Level approaches. Next, we contrast a Structural Level approach using three flip-flops to perform the counting achieved with an increment in the RTL example. We maintain portions of the prior module definitions to employ as a Test-Bench module. We also construct other modules with differing uses developed and introduced as needed.

To achieve a counter structure using flip-flops to perform the counting, we introduce a synthesizable model of a "D" flip-flop. Structural elements are introduced by connecting these flip-flops modules to perform the counting.

A text description of the flip-flop nodule follows:

```
module dff(q, d, ck);                          //D flip-flop with d, & ck inputs for q output
 output q;
 input          d, ck;
 reg            q;

  always @(posedge ck)                         // Positive edge clocking
   q <= d;                                     // Transfer d value to q register

endmodule
```

Program Listing 3.2

The only thing new in this description id the use of the "<=" defined as a "non-blocking assignment." This particular assignment means that the event completion is not entirely instantaneous, but is scheduled for the end of the current time step. We plan to use three dff modules and want them all to change state on a common clock signal. The non-blocking assignment causes all three to synchronize during one time step but change at the next step.

Because we need to initialize the dff module, we define another version with a reset input "r" defined with a reset "**r**" input as follows:

```
module dffr(q, d, r, ck);                      //D flip-flop with d, r, & ck inputs for q output
 output q;
 input          d, r, ck;
 reg            q;

  always @(posedge ck or negedge r) begin      // Positive edge clocking and Negative edge reset
                if(~r)                         // After Negative edge reset, r is binary false
                        q <= 0;                // Reset implies q goes to binary false or "0"
                else                           // Remaining test must be posedge ck
                        q <= d;                // Transfer d value to q register
                end

endmodule
```

Program Listing 3.3

You can save your module definition as "**dffr.v**" and follow the procedure for compiling to the "**dffr**" file without running the procedure using the "**vvp**" command. The save and compile supports syntax checking and debug and is recommended prior to the next steps.

In our next steps, we form a counter module constructed by "wiring" three dffr modules into a counter:

```
module counter(R, C);                   //Construct a counter - Collection of DFFR modules
        input R, C;                     // Reset and Clock inputs from the Test Benck
        reg [2:0] value;                //Define the Counter Value
        initial
                begin
                        value = 3'b0;    //Initialize the Counter Value
                end

        dffr Y1(q1, !value[0], R, C);   //Wire the LSB Counter Bit to Reset & Clock input
        dffr Y2(q2, !value[1], R, q1);  //Wire the Next bit to Reset and LSB as its Clock
        dffr Y3(q3, !value[2], R, q2);  //Wire the MSB to the Reset and Prior Bit

        always #1                       // Use the value as a Buffered Copy of the Counter Bits
                begin
                        value[2] <= q3;
                        value[1] <= q2;
                        value[0] <= q1;
                end
endmodule

module dffr(q, d, r, ck);               //Definition of DFFR D-Flip/Flop with Reset
        output  q;
        input   d, r, ck;
        reg             q;

        always @(posedge ck or negedge r) begin
                if(~r)
                        q <= 0;
                else
                        q <= d;
                end
endmodule
```

Program Listing 3.4

You should be able to enter this text into notepad ++ and compile until the syntax is correct. Note that the modules three instances of the dffr modules employ the Y1, Y2, and Y3 reference designators. The wiring between these instances is indicated by the position of the signal designators in the relative module lists.

Add to your notepad ++ file (above the prior module definitions), the following text:

```
module tester_3B_counter;               //Module test-bench generates counted signals
        reg Clk;
        reg Rst;
        reg [2:0] temp;

        initial                         // The single sequence below generates a Reset Pulse
        begin
                        Rst = 'b1;
                #1      Rst = !Rst;
                #1      Rst = !Rst;
        end

        initial                                         //Run the test once
        begin
                Clk = 'b0;
                temp = 3'b0;
                $dumpfile("Clk_3B_Count.vcd");          //Dump results of the simulation to Count.vcd
                $dumpvars;
                # 200 $finish;
        end

        always                                          //Generate the periodic clock signal
                begin
                        #10 Clk = !Clk;
                end
                        counter X1(Rst, Clk);           //Invoke the counter within the Test-Bench
        endmodule
```

Program Listing 3.5

To debug the Test-Bench, you can use a "//" comment designation at the beginning of the next-to-last line that invokes the "**counter X1(Rst, Clk)**" to temporarily deactivate it, and get the GTKWave test waveforms to verify the Test Bench as shown below:

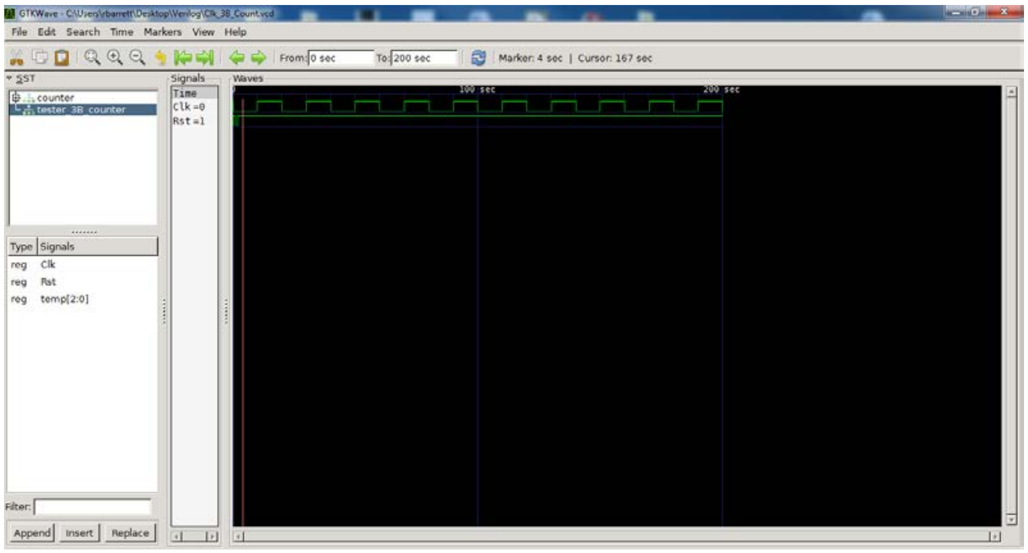


Figure 3.15 – Clock waveform in GTKWave GUI tester_3B_counter in Windows

Remove the temporary comment designator and you invoke the entire structure with the counter and **dffr** modules to get (See notes below, though):

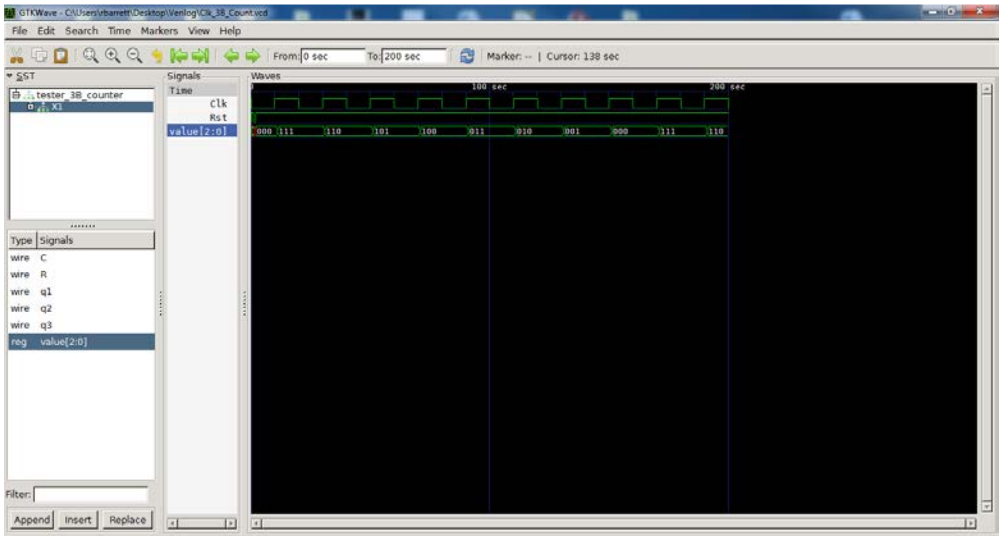


Figure 3.16 – Clock waveform in GTKWave GUI tester_3B_counter in Windows

Note that the waveform for **value[2:0]** is internal to the instance counter **X1 (Rst, Clk)** with the consequence that it is only visible within that scope. You must "click" on the hierarchy symbol at **X1** to make its signals visible. Drag **value[2:0]** to the Time pane and you will see the entire trace above. Further, if you sequentially click on the hierarchy symbols for **Y1**. **Y2**, and **Y3**, below **X1**, you can plot the traces associated with the "**q**" values in each **dffr** structure as below:

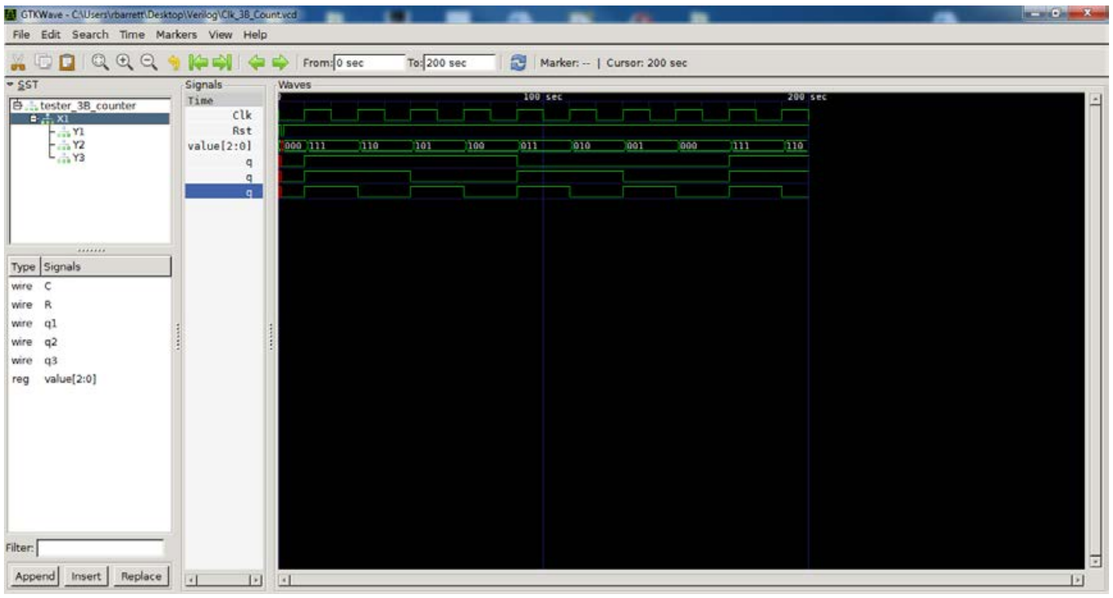


Figure 3.17 – Clock waveforms in GTKWave GUI tester_3B_counter in Windows

With this program, we have added a Structural Level representation of three clocking signals derived from the behavioral test-bench clock to the prior RTL Level, and the simpler Behavioral Level examples.

4.0 A Structural Verilog Logic Transistor-Level Design – The Test-Bench Modules we have been using all assume a Behavioral Level clocking signal. We now introduce a Structural Level example using all NMOS transistors to achieve a clock signal. This Structural Level example illustrates an old NMOS technology that is rarely used today, but serves to show the principles of Structural Level design using semiconductor switching devices. Although it is still a discrete-event digital design, it is also very close to the level that would usually require an analog simulation environment.

4.1 An NMOS Ring Oscillator Example - The “**Osc.v**” module below shows a 5-stage Ring Oscillator implemented in NMOS logic.

```
module Osc;
        supply0 gnd;
        supply1 vdd;
        reg Rst;                                //Define the Reset to initialize the Ring
        tri w1, w2, w3, w4, w5;                 //Define five Tri-state internal wires
        pullup (w1), (w2), (w3), (w4), (w5);    //Define Depletion-Load pullup on the wires

        initial
        begin
                        Rst = 'b1;              //Cycle the Reset
                #5      Rst = !Rst;
        end

 initial
  begin
                $dumpfile("Osc.vcd");           //Dump results of the simulation to Osc.vcd
                $dumpvars;
                #200 $finish ;
  end

        nmos #1                                 //Define five NMOS Ring Pull-Down NMOS
                a1 (w1, gnd, w2),
                a2 (w2, gnd, w3),
                a3 (w3, gnd, w4),
                a4 (w4, gnd, w5),
                a5 (w5, gnd, w1),
                a6 (w1, gnd, Rst),              //Define two NMOS Reset Pull-Down NMOS
                a7 (w4, gnd, Rst);
endmodule
```

Program Listing 4.0

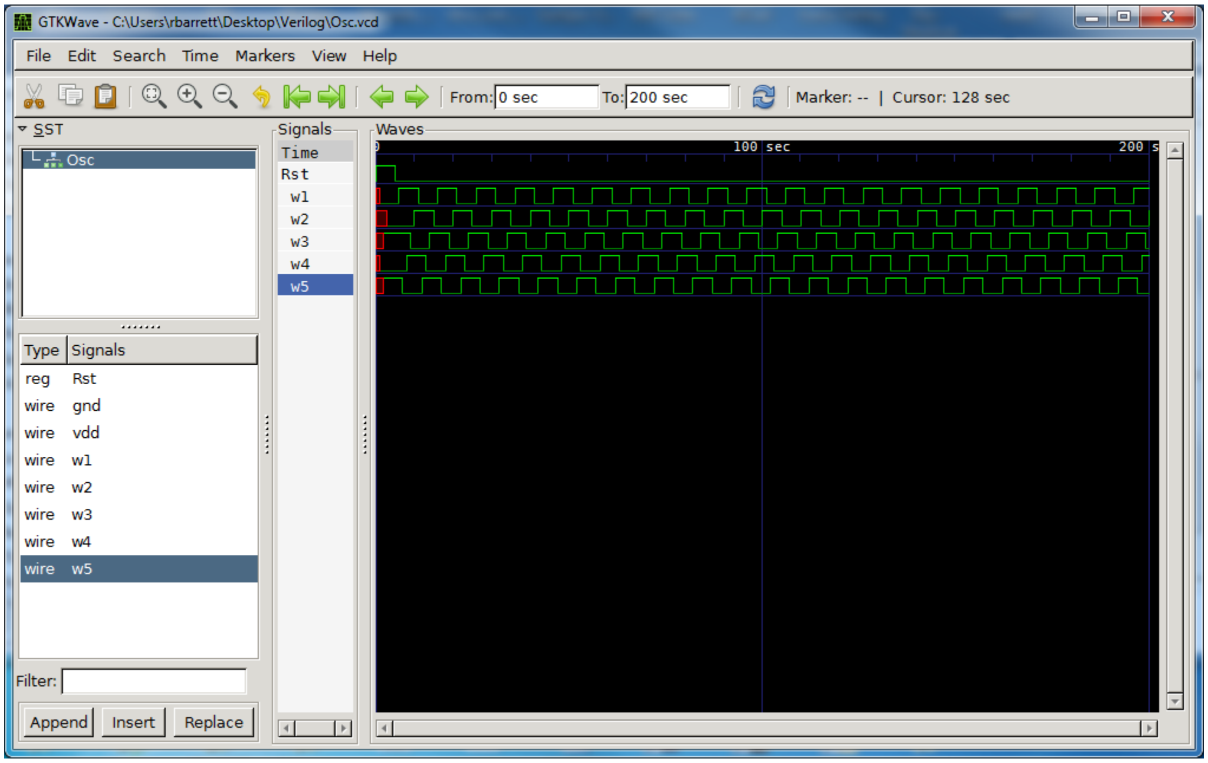


Figure 4.0 – Clock waveforms in GTKWave GUI Osc in Windows

For our example of the five-stage Ring Oscillator, "**nmos**" devices **a1**, **a2**, **a3**, **a4**, and **a5**, form NMOS inverters with their outputs respectively at "**tri**" wires **w1**, **w2**, **w3**, **w4**, and **w5**. They form a ring with the connection of the inverter gate inputs at adjacent outputs.

The addition of "**nmos**" devices **a6** and **a7**, with their inverter gate inputs at the "**Rst**" reset input serves to hold at "**tri**" wires **w1**, and **w4** at a "**0**" level while the "**Rst**" reset is high at its "**1**" level.

Removal of the "**Rst**" reset sets all stages to assume an inverter output after the "**#1**" delay associated with the activation of the "**nmos**" devices **a1**, **a2**, **a3**, **a4**, and **a5**, forming the oscillation from the initialized state.

The use of Structural Verilog Logic Transistor-Level Design tends to be rather specialized and primarily employed to verify specialized digital logic topologies. It, too, can be intermixed with Behavioral and RTL Levels, just as other Structural Level modules can.

The example, using "**tri**" wires, though, is often used to provide an interface on shared-bus designs, particularly at the "board-level" to interconnect disparate technologies.

5.0 Other Software with Verilog Support – The software tool suite we have introduced above is FOSS and a reasonable environment for investing the time to learn about Verilog in an affordable environment. There are several alternative environments built around other tool suites, some of which are also FOSS, or minimal cost. As a practicing Professional Engineer, you may find yourself delivering work-product in any number of such environments. We list and briefly describe below several of these alternatives.

5.1 The Eclipse IDE – Eclipse https://eclipse.org/ is a FOSS general-purpose software Integrated Development Environment (IDE) that supports a large number of languages, including Verilog. Eclipse can be obtained from an installer location: https://www.eclipse.org/downloads/eclipse-packages/

During installation of Eclipse, you will be asked to select an OS environment, including 32/64 bit versions, a default set of packages including C/C++ or Java support. You will probably be required to install several other packages including Java.

After a successful installation of the Eclipse IDE, you can obtain Verilog support from the site below and "drag: it into the installation to install: https://marketplace.eclipse.org/category/free-tagging/verilog

5.2 ModelSim Software – ModelSim https://en.wikipedia.org/wiki/ModelSim is a development environment from Mentor Graphics that has been "bundled" into other environments to support varying digital design products. One version can be obtained for free from Intel: https://www.altera.com/products/design-software/model---simulation/modelsim-altera-software.html as part of the starter edition obtained in the Intel acquisition of Altera. Both Linux and Windows versions are available. There are restrictions on the "free" license, but the tools are excellent in support of Altera/Intel FPGA designs.

5.3 Verilator Software – Verilator is a Verilog compiler that produces C/C++/Systemc outputs. http://www.veripool.org/projects/verilator/wiki/Installing

5.4 Verilog-AMS Software – Verilog-AMS is a super-set of Verilog and is used extensively in ASIC design environments including tools from Cadence https://www.cadence.com/ and Synopsys https://www.synopsys.com among others. Generally, these are high-end environments requiring large annual licensing fees.

6.0 FOSS Verilog Projects – There a number of FOSS repositories of Verilog code available, as well as a few restricted, but accessible repositories. Each project is managed according to the repository rules and are often collections provided by individual contributors. There are varying levels of quality between and among contributors so it is advised that the projects be used first as learning templates before committing the examples to a deliverable product.

6.1 GitHub Verilog Projects – The GitHub https://github.com/trending/verilog repository contains another wide range of projects that can be accessed and used with differing constraints, Numerous microprocessor and peripherals projects are available, including the 32 bit MIPS http://g-karthik.github.io/32BitMIPSProcessor/ processor..

6.2 SourceForge Verilog Projects – The SourceForge repository is vast and has a section with Verilog project https://sourceforge.net/directory/language:vhdl_verilog/os:windows/ support, as well as other HDL languages.

6.3 OpenCircuitsDesign Projects – The OpenCircuitsDesign site is focused primarily on VLSI design. Tim Edwards http://opencircuitdesign.com/verilog/ maintains tools and links to s large number of specialized tools to support all phases of IC design, including a digital design flow. Using the tools at this site, both FPGA and ASIC projects can be produced. By also establishing an account at MOSIS https://www.mosis.com/ , custom ASIC designs can be fabricated on a low-volume, shared-cost basis, or full production runs as the needs require.

7.0 Verilog for Digital Design: Summary and Conclusions – We have introduced the Verilog language in the context of Digital Design, introduced text editors, the iVerilog compiler, and the GTKWave waveform viewer. We have illustrated and contrasted the use of these Free Open-Source Software (FOSS) tools in both a Windows and Xubuntu Linux environment. We have introduced examples of Behavioral, RTL, and Structural Level programs with some discussion of the similarities and differences between the levels. This course is not exhaustive on the topic but makes a case for the value and usage of Verilog in Digital Design.