ForgeFPGA Software Simulation User Guide

Simulation is a technique of applying different input stimulus to the design at different times to check if the RTL code behaves the intended way. It is used to verify the robustness of the design. Simulation allows the user to view the timing diagram of the related signals to understand how the design description in the design file behaves.

Testbenches are pieces of code that are used for simulation. A simple testbench will instantiate the Unit Under Test (UUT) and drives the input. GoConfigure software uses Icarus Verilog (iVerilog) and GTKWave to observe the simulation waveforms with the stimulus provided in the testbench.

This document describes the steps that need to be taken while installing Icarus on your system and how to run a successful simulation.

Contents

1. Installing Icarus Verilog 3

2. Testbench 5

2.1 Understanding a Testbench 6

2.2 Timescale Definition in Testbench 7

2.3 Module Declaration 7

2.4 DUT Instantiation 8

2.5 Always & Initial Block in a Testbench 8

2.5.1 Initial block 8

2.5.2 Dump Files 9

2.5.3 Always Block 10

2.6 Self-Checking Testbench 10

3. GTKWave Software 12

3.1 Signal Values 14

3.2 GTKWave Toolbar 15

3.3 Menu Options 15

3.3.1 File 16

3.3.2 Edit 16

3.3.3 Search 16

3.3.4 Time 16

3.3.5 Marker 16

3.3.6 View 18

3.3.7 Help 18

4. Conclusion 18

5. Revision History 19

© 2024 Renesas Electronics

Figures

Figure 1. Go Configure Software Hub User Interface 3
Figure 2. Select SLG47910 3
Figure 3. FPGA Editor Button 4
Figure 4. Configure Tools 4
Figure 5. Parts of a Testbench 6
Figure 6. DUT and initial block relation 9
Figure 7: Self-checking testbench example statements 11
Figure 8: Simulation Log of printed values 12
Figure 9: Signal selection 14
Figure 10: GTKWave Toolbar 15
Figure 11: Menu Options 16
Figure 12: Marker Options 17
Figure 13: Collecting data from Markers 18

Reference

For related documents and software, please visit our website:

Download our free ForgeFPGA Designer software [1] and follow the steps in this user guide. User can reference [2] for the datasheet. Use Configuration Document to understand the different modes of configuration [3]. Renesas Electronics provides a complete library of application notes [4] featuring design examples as well as explanations of features and blocks within the Renesas IC.

[1] Go Configure Software Hub, Software Download, Renesas Electronics

[2] ForgeFPGA SLG47910 Datasheet, Renesas Electronics

[3] SLG47910, Configuration Document, Renesas Electronics

[4] Application Notes, ForgeFPGA Application Notes & Design Files, Renesas Electronics

[5] ForgeFPGA Development Board User Manual, Renesas Electronics

[6] ForgeFPGA Socket Adapter User Manual, Renesas Electronics

[7] ForgeFPGA Evaluation Board User Manual , Renesas Electronics

[8] ForgeFPGA Software Simulation User Manual, Renesas Electronics

1. Installing Icarus Verilog

a. Install the latest version of Icarus Verilog (IVerilog) from https://bleyer.org/icarus/

b. Be sure to add IVerilog to the PATH and let it install GTKWave (See Figure 1)

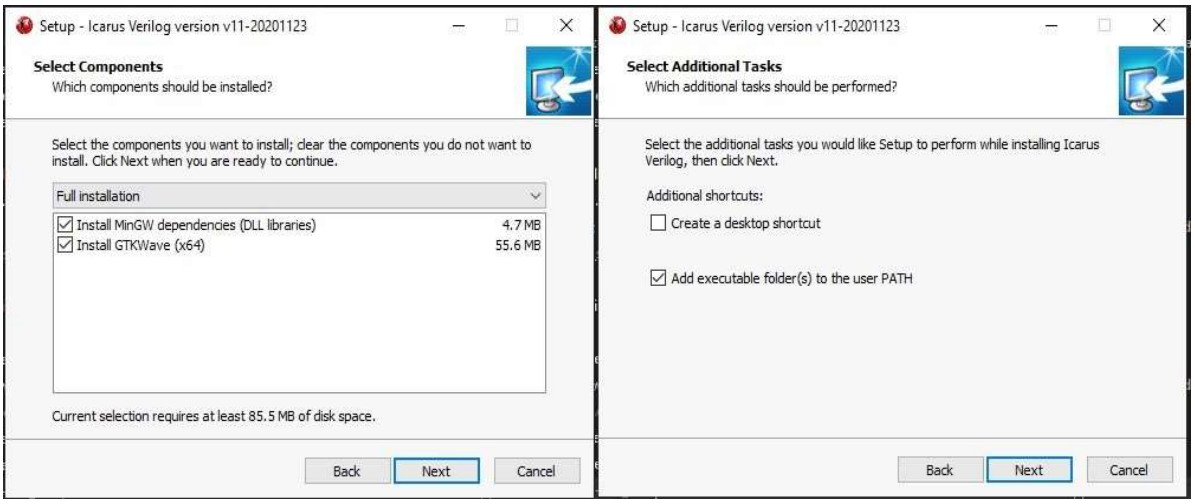

Figure 1. Go Configure Software Hub User Interface

c. Open the Go Configure Software and select the part: SLG47910(Rev BB) to open the Forge Workshop (see Figure 2).

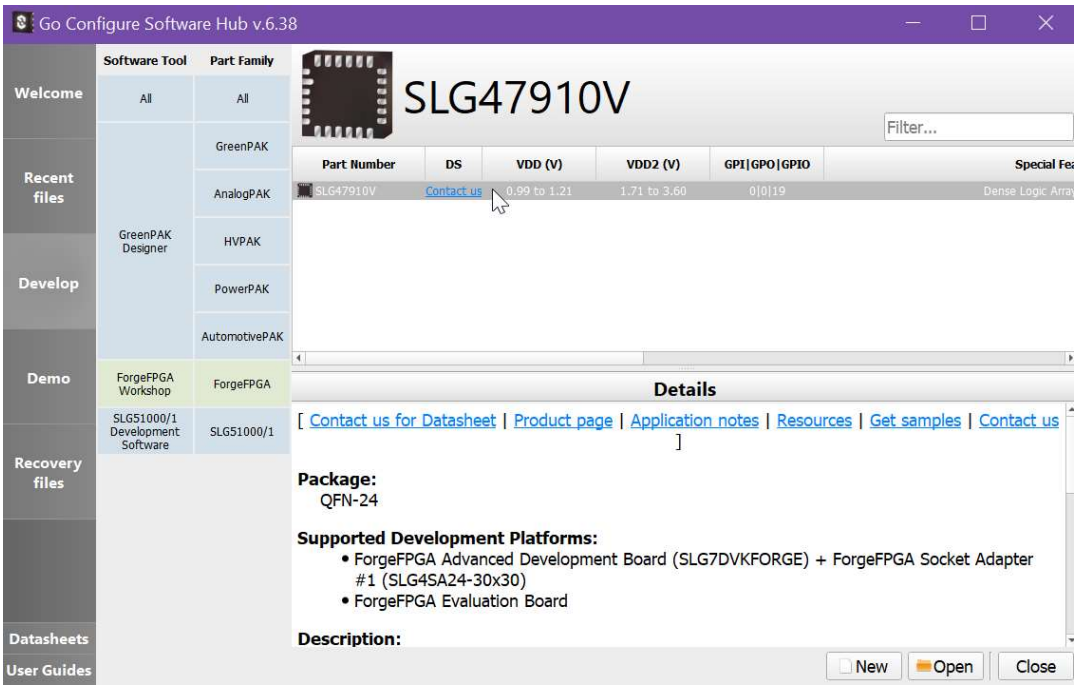

Figure 2. Select SLG47910

d. Click on the FPGA Editor in the middle of the toolbar at top or user can also double-click on the FPGA Core structure in the middle of the window.

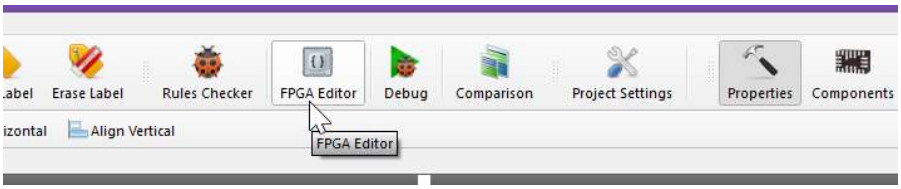

Figure 3. FPGA Editor Button

e. A new window opens called the Forge Workshop. In the menu toolbar on the top, click on Options → Settings. In the Settings dialog box, go to Tools under User Settings tab. Unselect the Use "system environment box" for both Icarus Verilog and GTKWave. Add the path to Iverilog and GTKWave saved in your system into the space given (see Figure 4).

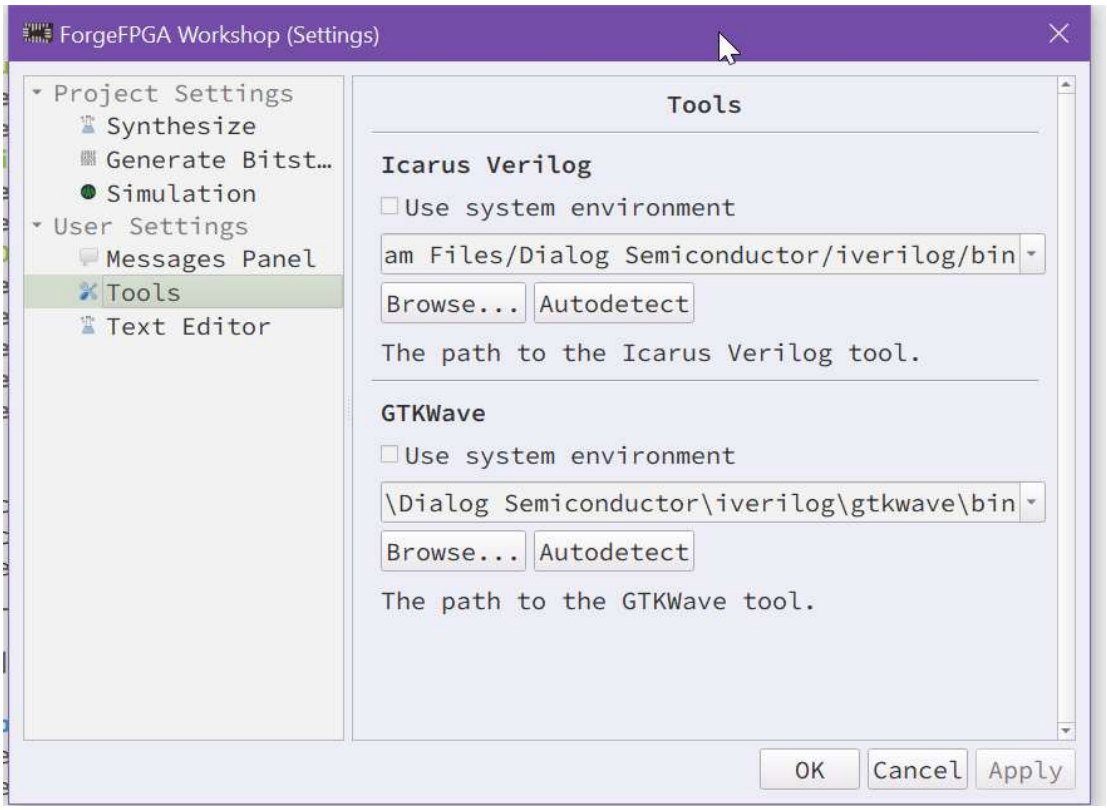

Figure 4. Configure Tools

You are all set down to simulate a testbench and the above steps ensure that the GTKWave launches automatically when simulating a testbench on Go Configure software.

2. Testbench

The most crucial step in successfully implementing any system is to verify the design and its functionality. Verifying a complex system after implementing the hardware is not a wise choice. It is ineffective in terms of money, time, and resources. Hence, in the case of FPGA, a testbench is used to test the Verilog source code.

Suppose we have an input which is of 11 bits, and we want to test the device for all the possible input combination values i.e. (2^{11}). As this is a very large number of combinations, it is impossible to test it manually. In such cases, testbenches are very useful as you can test the design automatically for all the possible values and hence, confirm the reliability of the test design. Verilog Testbenches are used to simulate and analyze designs without the need for any physical hardware device.

A design under test, abbreviated as DUT, is a synthesizable module of the functionality we want to test. In other words, it is the circuit design that we would like to test. We can describe our DUT using one of the three modeling styles in Verilog – Gate-level, Dataflow, or Behavioral.

A testbench is not synthesizable, hence it is used for simulation purposes only. This allows the user to use a full range of Verilog constructs e.g., keywords such as “for”, “$display” and “$monitor” etc. for writing testbenches. A simple testbench will instantiate the Unit Under Test (UUT) or Device Under Test (DUT) and drive inputs.

2.1 Understanding a Testbench

```
22 //timescale definition
23 `timescale 1ns / 1ps
24 // module definition
25 module ALU_tb;
26   //Parameter
27   parameter WIDTH = 8;
28   //Inputs
29   reg clk = 0;
30   reg nreset = 0;
31   reg [15:0] sel = 0;
32   reg [WIDTH-1:0] a = 0;
33   reg [WIDTH-1:0] b = 0;
34   //Outputs
35   wire [WIDTH-1:0] out;
36   wire carryout;
37
38   integer i;
39 // DUT instantiation
40   ALU #(
41     WIDTH
42   ) dut (
43     .clk (clk),
44     .nreset (nreset),
45     .sel (sel),
46     .a (a),
47     .b (b),
48     .out (out),
49     .carryout (carryout)
50   );
51 //clock
52   always #1 clk <= ~clk;
53
54   initial begin
55     $dumpfile ("ALU_tb.vcd"); // GTKWave simulation file
56     $dumpvars (0, ALU_tb);    // dump variable file
57 //stimulus defined
58     clk = 0;
59     nreset = 0;
60     sel = 'h01;
61     a = 'h2C;
62     b = 'h14;
63     #10 nreset = 1;
64     #10;
65     for (i=1; i<16; i=i+1) begin
66       sel = sel << 1;
67       #10;
68     end  // for loop
69     a = 'hAF;
70     b = 'h71;
71     #10 $finish;
72   end  // end of initial block
73 endmodule
```

Time Scale Definition

Module Definition
- Inputs
- Outputs

DUT Instantiation

Clock Definition

Dump files

Stimulus Definition

Figure 5. Parts of a Testbench

2.2 Timescale Definition in Testbench

When simulating, the software needs to know how the time has been defined. The delay unit is specified using the `timescale directive, which specifies the time unit and the precision for the modules that follow it. The `timescale helps in determining what #1 means in terms of time. # is used to define the delay to be introduced in the system in accordance with time unit specified in timescale. So, #1 means 1 ns of delay if the time_unit is in ns.

Syntax :

```
`timescale <time_unit>/<time_precision>
```

time_unit is the amount of time a delay of #1 represents. The *time_precision* base represents how many decimal points of precision to use relative to the time units. (See line 23 in Figure 5)

We can use the timescale constructs to use different time units in the same design. The user needs to remember that delay specifications are not synthesizable and cannot be converted to hardware logic. The delay functions are entirely for simulation purposes. $time and $realtime system functions return the current time and the default reporting format can be changed with another system task $timeformat.

Example:

```
`timescale 10us/100ns
`timescale 1ns/1ps
#10 reset = 1;  // delays the signal by 10 ns
#0.49  $display( "T = %0t at Time #0.49", $realtime) ;
```

The delay specified is #0.49 which is less than the half a unit time. However, the time precision is specified to be 1ps and hence the simulator cannot go smaller than 1ns which makes it to round the given delay statement and yield 0ns. So, this statement fails to provide any delay.

Simulation Log:

```
T = 1 at Time #0.49
```

2.3 Module Declaration

Module declaration in any testbench is unlike the main Verilog code. In a testbench, the module is declared without any terminal ports along with it. (See line 25 in Figure 5)

Syntax:

```
module <modulename_tb>;
```

The module declaration is followed by defining the input and output signals defined earlier in the main design file. We use two signal types for driving and monitoring signals during the simulation. The reg datatype will hold the value until a new value is assigned to it. This datatype can be assigned a value only in always or initial block.

The wire datatype is like that of a physical connection. It will hold the value that is driven by a port, assign statement, or reg. This data type cannot be used in initial or always block. Any parameter and integer declaration are also done in this section.

Example:

```
Reg a,b;    // the input in the HDL code is defined as reg in testbench
Wire y;     // output signal in HDL is defined as wire in testbench
```

2.4 DUT Instantiation

The purpose of a testbench is to verify whether our DUT module is functioning. Hence, we need to instantiate our design module to test module.

Syntax:

<dut_module><instance_name> (. <signal1> (signal1), . signal2> (signal2));

Example:

```
ALU d0 (.a(a), // signal "a" in ALU should be connected to "a" in ALU_tb module
        .b(b),  // signal "b" in ALU should be connected to "b" in ALU_tb module
        .c(c)) ;// signal "c" in ALU should be connected to "c" in ALU_tb module
```

We have instantiated the DUT module ALU to the test module. The instance name (d0) is the user's choice. The signals with a period "`.`" in front of them are the names for the signals inside the ALU module, while the wire or reg they connect to in the test bench is next to the signal in parenthesis (). It is recommended to code each port connection in a separate line so that any compilation error message will correctly point to the line number where the error occurred. Because these connections are made by name, the order in which they appear is irrelevant.

DUT instantiation can also be made for the modules where the testbench module has different signal names. The correct mapping of the signals is what is important when instantiating.

Example :

```
ALU d0 (.a(A),              // signal "a" in ALU should be connected to "A" in ALU_tb module
        .clk(clock),        // signal "clk" in ALU should be connected to "clock" ALU_tb module
        .out(OUT)) ;      // signal "out" in ALU should be connected to "OUT" in ALU_tb module
```

2.5 Always & Initial Block in a Testbench

There are two sequential blocks in Verilog, initial and always. It is in these blocks that we apply the stimulus.

2.5.1 Initial block

The initial block which is executed only once and terminated when the last line of the block is executed. The stimulus is written into the initial block. (See line 54-72 in Figure 5)

Syntax:

```
..
initial begin
$dumpfile();
$dumpvars();
..(enter stimulus)
end
```

the initial block begins its execution at the start of the simulation at time t = 0. Starting with the first line between the begin and end, each line executes from top to bottom until a delay is reached. When the delay is reached, the execution of this block waits until the delay time (10-time units) has passed and then picks up execution again. User can define stimuli using loops (for, while, if-else) as well inside this initial block instead of entering all the combinations manually.

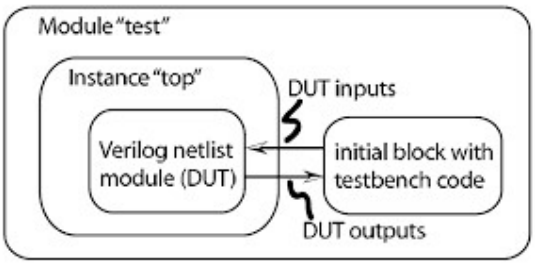

Figure 6. DUT and initial block relation

Example :

```
Initial Begin
A = 0; b = 0;            // start execution
#10 a = 0; b = 1;        // execution is at t = 10-unit time
#10 a = 1; b = 0;        // execution is at t = 20-unit time
end
```

2.5.2 Dump Files

Another thing to keep in mind is the declaration of $dumpfiles and $dumpvars inside the initial block (see line 55-56 in Figure 5). The $dumpfile is used to dump the changes in the values of nets and registers in a file that is named as its argument.

For example:

```
$dumpfile("alu_tb.vcd");
```

will dump the changes in a file named alu_tb.vcd. The changes are recorded in a file called VCD file that stands for value change dump. A VCD (value change dump) stores all the information about value changes. We cannot have more than one $dumpfile statements in Verilog simulation.

The $dumpvars is used to specify which variables are to be dumped (in the file mentioned by $dumpfile). The simplest way to use it is without any argument. The general syntax of the `$dumpvars` is

```
$dumpvars(<levels> <, <module_or_variable>>);
```

We basically can specify which modules, and which variables in modules will be dumped. The simplest way to use this is to set the level to 0 and module name as the top module (typically the top testbench module).

```
$dumpvars(0, alu_tb);
```

When level is set to 0, and only the module name is specified, it dumps ALL the variables of that module and all the variables in ALL lower-level modules instantiated by this top module. If any module in not instantiated by this top module, then its variable will not be covered. One more thing, the declaration of $dumpfile must come before the $dumpvars or any other system tasks that specifies dump. These dumpfiles must be declared before the stimulus inputs else, no value will be saved in these dumpfiles.

2.5.3 Always Block

Contrary to the initial statements, an always block repeatedly executes, although the execution starts at time t = 0. For example, the clock signal is essential for the operation of sequential circuits like Flip-flops. It needs to be supplied continuously. Hence, we can write the code for operation of the clock in a testbench as (see line 52 in Figure 5):

```
always
#10 clk = ~clk;
endmodule
```

The above statement gets executed after 10 ns starting from t = 0. The value of the clk will get inverted after 10 ns from the previous value. Thus, generating a clock signal of 20 ns pulse width. Therefore, this statement generates a signal of frequency 50 MHz. It is important to note that, the initialization of the signal is done before the always block. If we do not do the initialization part, the clk signal will be x from t - 0, and after 10 ns, it will be inverted to another x.

2.6 Self-Checking Testbench

A self-checking testbench includes a statement to check the current state.

- $display system task are mainly used to display debug messages to track the flow of simulation

```
initial begin
A = 0 ; b = 0  ; c = 0;   #10;       // apply input, wait
if( y ! == 1)  begin
     $display( "000 failed") ;       //check
     c = 1; #10 ;                    //apply input, wait
     end
else if ( y ! == 0) begin
     $display("001 failed")          // check
     b = 1; c = 0; #10 ;  end
else if(y!==0)
     $display (" 010 failed");       //check
     end
endmodule
```

$display is used for displaying values of variables, strings, or expressions. From the above example, whenever any of the if-else loop is satisfied, then the simulator log will display its respective $display statement. There is a newline by default at the end of the strings.

```
$display ("time = %t , A = %b, B = %b, C = % b", $time, A,B,C);
```

The characters mentioned in the quotes will be printed as they are. The letter along with % denotes the string format. We use %b to represent binary data. We can use %d, %h, %o for representing decimal, hexadecimal, and octal, respectively. The %g is used for expressing real numbers. These will be replaced with the values outside the quote in the order mentioned. For example, the above statement will be displayed in the simulation log as:

time = 20, A = 0, B =1, C = 0

Table 1. Verilog Table Formats

Argument	Description
%h, %H	Display in Hexadecimal format
%d, %D	Display in decimal forma
%b, %B	Display in binary format
%m, %M	Display hierarchical name
%s, %S	Display as string
%t, %T	Display in time format
%f, %F	Display 'real' in decimal format
%e, %E	Display 'real' in an exponential format

$display mainly prints the data or variable as it is at that instant of that time like the *printf in C*. We must mention $display for whatever text we have to view in the simulation log.

- $time

$time is a system task that will return the current time of the simulation.

- $monitor

$monitor will monitor the data or variable for which it is written and whenever the variable changes, it will print the changed value. It achieves a similar effect of calling $display after every time any of its arguments get updated. $monitor is like a task that is spawned to run in the background of the main thread which monitors and displays value changes of its argument variables. $monitor has the same syntax as $display.

```
$monitor(" time = %t, A = %b, B = %b, C = % b", $time, A,B,C);
```

```
//stimulus defined
    clk = 0;
    nreset = 0;
    sel = 'h01;
    a = 'h2C;
    b = 'h14;
    #10 nreset = 1;
    #10;
$display( "<display 1> time=%t, a = %h, B = %h, sel = %d, out = %h", $time, a, b,sel,out);
$monitor( "<monitor 1> time=%t, a = %h, B = %h, sel = %d, out = %h", $time, a, b,sel,out);
    for (i=1; i<16; i=i+1) begin
      sel = sel << 1;
      #10;
$display( "<display 2> time=%t, a = %h, B = %h, sel = %d, out = %h", $time, a, b,sel,out);
$monitor( "<monitor 2> time=%t, a = %h, B = %h, sel = %d, out = %h", $time, a, b,sel,out);
    end  // for loop
$monitor( "<monitor 3> time=%t, a = %h, B = %h, sel = %d, out = %h", $time, a, b,sel,out);
$display( "<display 3> time=%t, a = %h, B = %h, sel = %d, out = %h", $time, a, b,sel,out);


    a = 'hAF;
    b = 'h71;
    #10 $finish;
  end  // end of initial block
endmodule
```

Figure 7: Self-checking testbench example statements

From Figure 7 you can observe that new lines of codes have been added to self-evaluate the testbench. The placement of the $display and $monitor statements in different sections of the testbench will yield different results (see Figure 8). $time mentioned in these statements prints the time at which the value is being printed

for. At the same time unit say 170000, we can see how there is a difference in the value for A and B due to the $display and $monitor statements.

```
Logger    Issues
VCD info: dumpfile ALU_tb.vcd opened for output.
<display 1> time=          20000, a = 2c, B = 14, sel =     1, out = 40
<monitor 1> time=          20000, a = 2c, B = 14, sel =     2, out = 40
<monitor 1> time=          21000, a = 2c, B = 14, sel =     2, out = 18
<display 2> time=          30000, a = 2c, B = 14, sel =     2, out = 18
<monitor 2> time=          30000, a = 2c, B = 14, sel =     4, out = 18
<monitor 2> time=          31000, a = 2c, B = 14, sel =     4, out = 70
<display 2> time=          40000, a = 2c, B = 14, sel =     4, out = 70
<monitor 2> time=          40000, a = 2c, B = 14, sel =     8, out = 70
<monitor 2> time=          41000, a = 2c, B = 14, sel =     8, out = d3
<display 2> time=          50000, a = 2c, B = 14, sel =     8, out = d3
<monitor 2> time=          50000, a = 2c, B = 14, sel =    16, out = d3
<monitor 2> time=          51000, a = 2c, B = 14, sel =    16, out = 58
<display 2> time=          60000, a = 2c, B = 14, sel =    16, out = 58
<monitor 2> time=          60000, a = 2c, B = 14, sel =    32, out = 58
<monitor 2> time=          61000, a = 2c, B = 14, sel =    32, out = 16
<display 2> time=          70000, a = 2c, B = 14, sel =    32, out = 16
<monitor 2> time=          70000, a = 2c, B = 14, sel =    64, out = 16
<monitor 2> time=          71000, a = 2c, B = 14, sel =    64, out = 58
<display 2> time=          80000, a = 2c, B = 14, sel =    64, out = 58
<monitor 2> time=          80000, a = 2c, B = 14, sel =   128, out = 58
<monitor 2> time=          81000, a = 2c, B = 14, sel =   128, out = 16
<display 2> time=          90000, a = 2c, B = 14, sel =   128, out = 16
<monitor 2> time=          90000, a = 2c, B = 14, sel =   256, out = 16
<monitor 2> time=          91000, a = 2c, B = 14, sel =   256, out = 04
<display 2> time=         100000, a = 2c, B = 14, sel =   256, out = 04
<monitor 2> time=         100000, a = 2c, B = 14, sel =   512, out = 04
<monitor 2> time=         101000, a = 2c, B = 14, sel =   512, out = 3c
<display 2> time=         110000, a = 2c, B = 14, sel =   512, out = 3c
<monitor 2> time=         110000, a = 2c, B = 14, sel =  1024, out = 3c
<monitor 2> time=         111000, a = 2c, B = 14, sel =  1024, out = 38
<display 2> time=         120000, a = 2c, B = 14, sel =  1024, out = 38
<monitor 2> time=         120000, a = 2c, B = 14, sel =  2048, out = 38
<monitor 2> time=         121000, a = 2c, B = 14, sel =  2048, out = fb
<display 2> time=         130000, a = 2c, B = 14, sel =  2048, out = fb
<monitor 2> time=         130000, a = 2c, B = 14, sel =  4096, out = fb
<monitor 2> time=         131000, a = 2c, B = 14, sel =  4096, out = c3
<display 2> time=         140000, a = 2c, B = 14, sel =  4096, out = c3
<monitor 2> time=         140000, a = 2c, B = 14, sel =  8192, out = c3
<monitor 2> time=         141000, a = 2c, B = 14, sel =  8192, out = c7
<display 2> time=         150000, a = 2c, B = 14, sel =  8192, out = c7
<monitor 2> time=         150000, a = 2c, B = 14, sel = 16384, out = c7
<monitor 2> time=         151000, a = 2c, B = 14, sel = 16384, out = 01
<display 2> time=         160000, a = 2c, B = 14, sel = 16384, out = 01
<monitor 2> time=         160000, a = 2c, B = 14, sel = 32768, out = 01
<monitor 2> time=         161000, a = 2c, B = 14, sel = 32768, out = 00
<display 2> time=         170000, a = 2c, B = 14, sel = 32768, out = 00
<display 3> time=         170000, a = 2c, B = 14, sel = 32768, out = 00
<monitor 3> time=         170000, a = af, B = 71, sel = 32768, out = 00
```

Figure 8: Simulation Log of printed values

3. GTKWave Software

GTKWave is a fully featured GTK+ wave viewer for Unix, Win32, and Mac OSX which reads LXT, LXT2, VZT, FST, and GHW files as well as standard VCD/EVCD files and allows their viewing. Its official website is at http://gtkwave.sourceforge.net/ . GTKWave is the recommended viewer by Icarus Verilog simulation tool.

Once the user is successfully created a testbench to test the functionality of the design, the user can now use the GTKWave software to view the waveforms.

To launch the GTKWave software to view the waveforms, the user needs to click on Simulate Testbench button on the top of the toolbar or from the main menu *Tools → Simulation → Simulate Testbench.* If there are no syntax errors then depending on the design, the GTKWave should be launched automatically or the results of the stimuli in the testbench will be displayed in the Logger section of the window.

The GTKWave software opens the .vcd format dumpfile automatically. The GTKWave window does not display the waveform when its opens. This gives the user an opportunity to select which signals it wants to view and observe. To choose the signal, the user needs to display, the user needs click on the name of their module/instance on the left side of the window under the SST tab. By clicking the **+** of every instance, you can see the signals that are related wit that instance in the bottom section. Then you can drag & drop the desired signal or double-click them to be displayed in the Signals window. You can also select all (CTRL + A) and insert them to the signals window (see Figure 9).

The signals are now added to the signal window but its yet to be simulated. After adding the desired signals to the signal window, click on to fit the signals to the current width of the window and then reload the signals from the reload symbol present on the toolbar. You can now see the signals with their respective values.

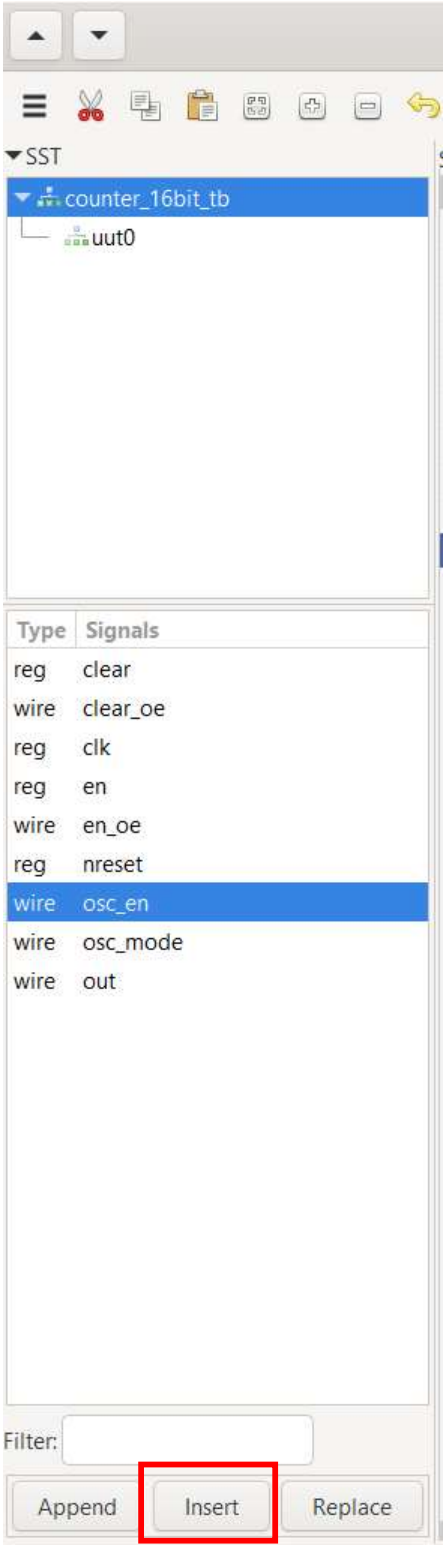

Figure 9: Signal selection

3.1 Signal Values

By default, the values of the signals are in hexadecimal format and all the waves are colored green (if correctly running).

User can change the properties of these signal by right-clicking on the signal and choosing Data Format or Color Format. User can also insert a blank signal to make sections between group of signals. When you have the desired optical result, you can save your configurations by going File → Write Save File.

3.2 GTKWave Toolbar

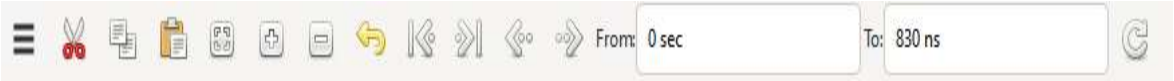

Figure 10: GTKWave Toolbar

The toolbar (see Figure 10) allows the user to perform basic functions for the signal. Let us discuss each option on the toolbar from left to right.

1. **Menu Options:** Under this option we can view all the various features of the software that can be used to play around with the software. The details under this menu option are covered under Section 8 of this user guide.
2. **Cut Traces:** It is used to delete/cut the select signal from the signal window
3. **Copy Traces:** It is used to copy the selected signal from the signal window
4. **Paste Traces:** The copied/cut trace can be pasted at a different location in the signal window
5. **Zoom Fit:** It is used to fit the signals according to the size of the window the user chooses to display
6. **Zoom In:** It is used to zoom in the signal window
7. **Zoom Out:** It is used to zoom out the signal window
8. **Zoom Undo:** it is used to undo the zoom in/out on the signal window
9. **Zoom to Start:** this will zoom the signal window, displaying the start time of the signals.
10. **Zoom to End:** this will zoom the signal window displaying the end time of the signals
11. **Find previous edge:** This shifts the marker to the left side indicating the previous edge
12. **Find next edge:** This shifts the marker to the right indicating the next edge
13. **Scroll lower/upper bond:** using this we can set the time frame in which the user wants to display. For example, we can set the time frame to 0 sec to 500 ns, it will display the signals under that duration only.
14. **Reload:** The reload is pressed whenever there is a change to the displayed signal. It will reload and display the signal according to the new parameters. For example, after changing the time frame of the signal, we need to reload the signal to display the signal in the new set time frame.

3.3 Menu Options

From the left top corner of the GTKWave software, user can access the menu options by clicking the three vertical lines (see Figure 11). The user can find the following options under the Menu options:

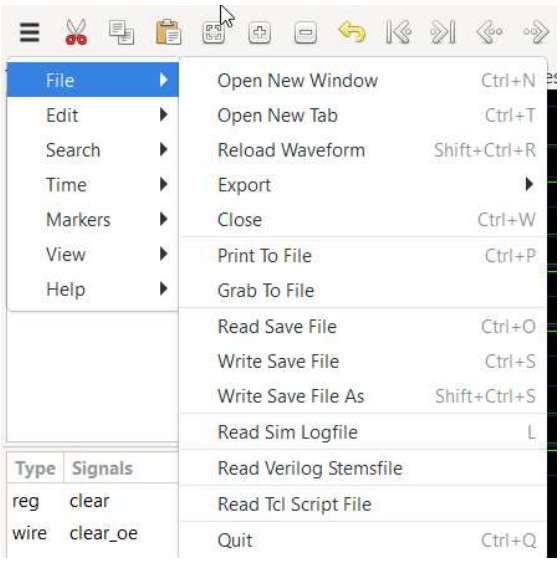

Figure 11: Menu Options

3.3.1 File

The File submenu contains various items related to accessing files, importing-exporting VCD files, printing, and reading/writing files and exiting.

3.3.2 Edit

The Edit submenu is used to perform various utility functions such as changing the data representation of values in the wave subwindow. Using the options under the Edit submenu, user can change the data format of the signals, rearrange them, shift them, trim it, highlight it, group signals, comment on signals, change the color of the signals, etc.

3.3.3 Search

The Search submenu is used to perform searches on net names and values. It helps to perform functions on different hierarchy levels of the signals and instances in the VCD file.

3.3.4 Time

The time submenu contains a superset of the functions performed by the Navigations and the Status Panel buttons. It enables simple, time related, functions like zooming, moving to a particular time point, shifting the signal in a certain direction, etc.

3.3.5 Marker

The marker submenu is used to perform various manipulations on the marker as well as control scrolling offscreen. It enables the functionality of adding numerous markers on the signal window. A maximum of 26 names markers are allowed and the times for all must be different.

a. **To add Markers in the signal window**

   Left click at the required point where you want the Marker to be placed and press ALT + N. This will place a named marker (A,B,C, etc.) at the required point. User can continue to do this for 26 different time locations.

   To compare the time value at all the places markers, **Menu → Markers → Show Change Marker Data**. This will open a window with the time value at each Marker. The user can manually note the time value at each marker placed and subtract them to calculate the time difference between 2 markers.

b. **To remove Marker in the signal window**

User can go to **Menu → Markers → Collect Named Marker**. This will remove the last-named Marker placed in the signal window. User can remove all the named Markers by going to **Menu → Markers → Collect All Named Marker** (Figure 12).

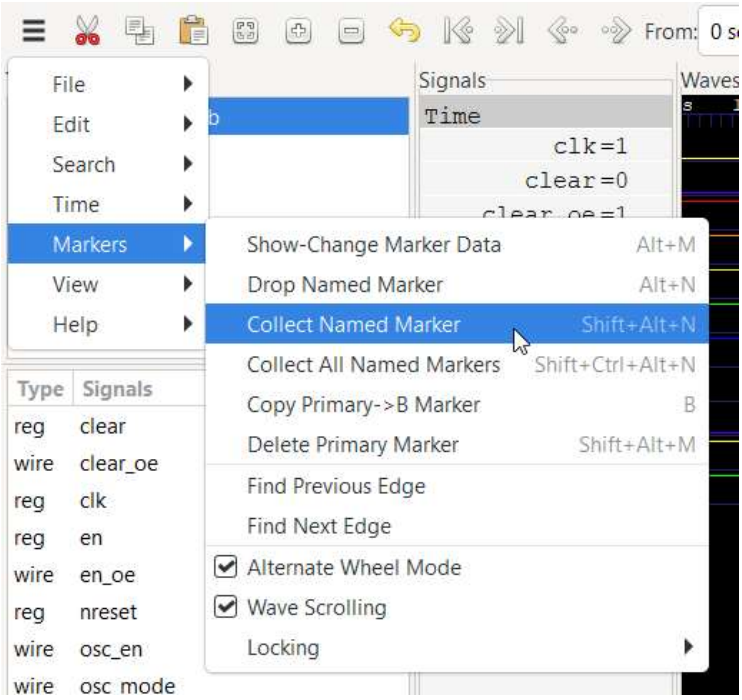

Figure 12: Marker Options

In Figure 13, we can see how the signal colors have been changed. You can observe a Blank Signal added to the signal window as well with a comment - Blank Signal.

Also note the presence of 6 Named Markers (A - E) and the comparation of the time value between these Markers in ps.

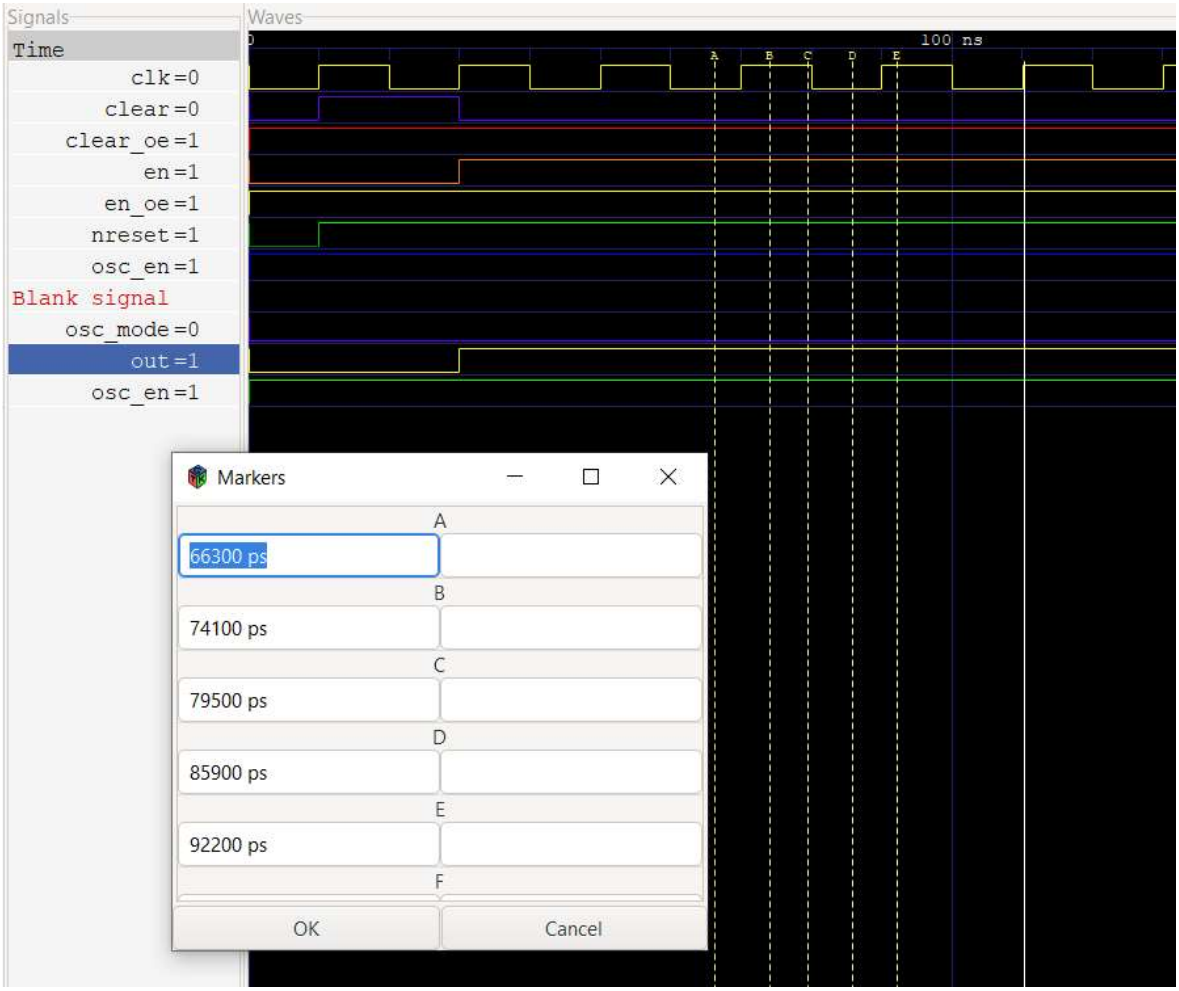

Figure 13: Collecting data from Markers

3.3.6 View

The View submenu is used to control various attributes dealing with the graphical rendering of status items as well as values in the signal sub window. From this menu, you can convert the signal window to Black & White or colored as well. The View submenu also enables you to change the time Dimension ranging from seconds (secs) to ficoseconds (fs). The user can find this option **View → Scale to Time Dimension → fs**.

3.3.7 Help

The help submenu contains options for enabling on-line help as well as displaying program version information.

4. Conclusion

This document was created to assist the user in successfully simulating their design and verifying the functionality by correcting drafting the needed testbench and using Icarus Verilog along with GTKWave to display the waveforms and observe the results.

5. Revision History

Revision	Date	Description
1.00	May 20, 2024	Initial release.