

Using the ModelSim-Intel FPGA Simulator

for Verilog Code

This tutorial introduces the simulation of Verilog code using the *ModelSim-Intel FPGA* simulator. We assume that you are using *ModelSim-Intel FPGA Starter Edition version 18.1*. This software can be downloaded and installed from the *Download Center for Intel FPGAs*. In this download center, you can select release *18.1* of the *Quartus Prime Lite Edition*, and then on the `Individual Files` tab choose to download and install the *ModelSim-Intel FPGA Starter Edition* software.

Getting Started

To introduce the *ModelSim* software, we will first open an existing simulation project. The project is a multibit adder named *Addern*, and is included as part of the *design files* provided along with this tutorial. Copy the *Addern* project to a folder on your computer, such as `C:\ModelSim_Tutorial\Addern`. In the *Addern* folder there is a Verilog source-code file called *Addern.v* and a subfolder named *ModelSim*. The *Addern.v* file, shown in Figure 1, is the Verilog code that we will simulate in this part of the tutorial. We will use a Verilog *testbench* to specify signal values for the adder's inputs, *Cin*, *X*, and *Y*, and then *ModelSim* will generate corresponding values for the outputs, *Sum* and *Cout*.

```
// A multi-bit adder
module Addern (Cin, X, Y, Sum, Cout);

    parameter n = 16;
    input Cin;
    input [n-1:0] X, Y;
    output [n-1:0] Sum;
    output Cout;

    assign {Cout, Sum} = X + Y + Cin;

endmodule
```

Figure 1: Verilog code for the multibit adder.

We will use three files, included in the *ModelSim* subfolder, to control the *ModelSim* simulator. The files are named *testbench.v*, *testbench.tcl*, and *wave.do*.

The *testbench.v* file is a type of Verilog file known as a *testbench*. It is used to *instantiate* the multibit adder module, and to specify values for its inputs. The first statement in this Verilog testbench, illustrated in Figure 2, is called a *timescale compiler directive*. Its first argument sets the *units* of simulation time to 1 nanosecond, and the second argument sets the *resolution* of the simulation to 1 picosecond. We will use these values for all of our simulations in this tutorial.

Line 3 is the start of the testbench module, which has no inputs or outputs. In lines 5 to 8 we declare *reg* type signals for the adder inputs *Cin*, *X* and *Y*, and we declare *wire* type signals for the adder outputs *Sum* and *Cout*. The *reg* and *wire* types are chosen for these signals based on how they are used later in the testbench code.

Lines 10 to 17 provide an *initial block*, which is used to specify the values of the adder inputs. First, in Line 11 *X*, *Y*, and *Cin* are initialized to 0. Line 12 specifies that after 20 simulation time *units* the value of the input *Y*

changes to 10. Since the unit of simulation time is set to 1 ns by the timescale directive in Line 1, this means that *Y* changes to the value 10 at 20 ns in simulation time. Line 13 specifies that after another 20 ns, meaning at 40 ns in simulation time, input *X* changes to 10. The rest of the initial block specifies various values for the adder inputs at 20 ns time increments.

```

1  `timescale 1ns / 1ps
2
3  module testbench ( );
4
5      reg Cin;
6      reg [15:0] X, Y;
7      wire [15:0] Sum;
8      wire Cout;
9
10     initial begin
11         X <= 0; Y <= 0; Cin <= 0;
12         #20 X <= 0; Y <= 10; Cin <= 0;
13         #20 X <= 10; Y <= 10; Cin <= 0;
14         #20 X <= 10; Y <= 10; Cin <= 1;
15         #20 X <= 16'hFFF0; Y <= 16'hF; Cin <= 0;
16         #20 X <= 16'hFFF0; Y <= 16'hF; Cin <= 1;
17     end // initial
18
19     Addern U1 (Cin, X, Y, Sum, Cout);
20
21 endmodule

```

Figure 2: The Verilog testbench code.

In Line 19 the testbench module instantiates the *Addern* module. Its inputs are driven by the testbench signal values specified in the initial block. Verilog syntax requires the type *reg* to be used for any signal that is assigned a value inside an initial block. This is why *Cin*, *X*, and *Y* are declared as type *reg* in Lines 5 and 6. Verilog syntax specifies that the type *wire* has to be used for the testbench signals *Sum* and *Cout* that are attached to the outputs of the instantiated *Addern* module.

Open the *ModelSim* software to reach the window shown in Figure 3. Click on the *Transcript* window at the bottom of the figure and then use the `cd` command to navigate to the *ModelSim* project for the multibit adder. For example, in our case we would type `cd C:/ModelSim_Tutorial/Addern/ModelSim`. Note that *ModelSim* uses the `/` symbol to navigate between filesystem folders, even though the Windows operating system uses the `\` symbol for this purpose. Next, we wish to run a series of simulator commands that are included in the script file *testbench.tcl*. To run the script, in the *Transcript* window type the command `do testbench.tcl`. *ModelSim* will execute the commands in this script and then update its graphical user interface to display the simulation results.

Figure 4 shows the contents of the script *testbench.tcl*. First, the `quit` command is invoked to ensure that no simulation is already running. Then, in Line 5 the `vlib` command is executed to create a working design library for the current project. The Verilog compiler is invoked in Line 8 to compile the source code for the project, which is in the *parent* folder (`..`), and in Line 10 to compile *testbench.v* in the current folder.

The `vsim` command in Line 12 starts the simulation. It includes some simulation libraries for Intel FPGAs that may be needed, depending on what library elements are used in the project. If the included libraries aren't needed for the current project, then they will be ignored during the simulation. Line 14 in Figure 4 executes the command `do wave.do`, which configures the *ModelSim* waveform-display window. The final command in Figure 6 advances the simulation by the specified amount of time, which in this case is 120 ns. The updated *ModelSim* window after running the *testbench.tcl* script is illustrated in Figure 5.

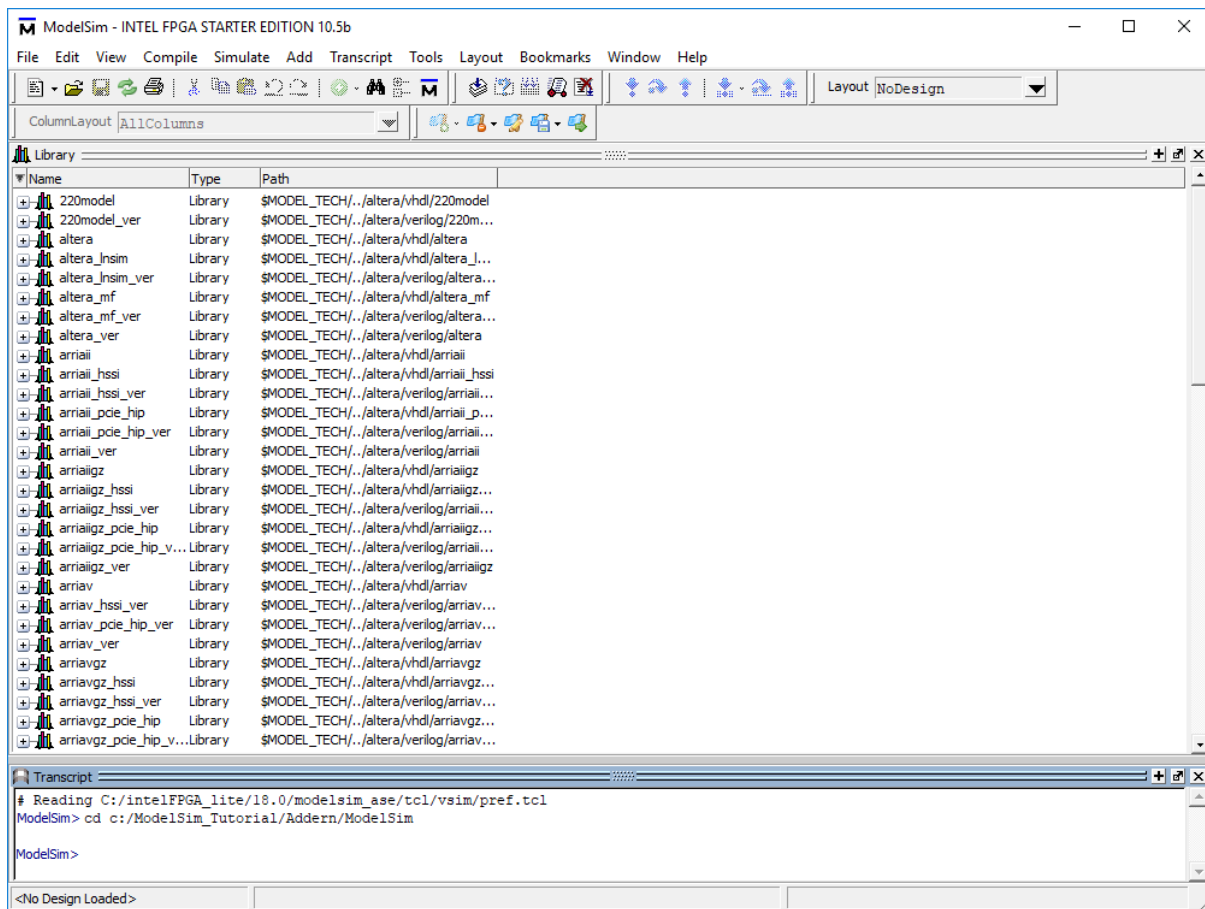


Figure 3: The *ModelSim* window.

```

1  # stop any simulation that is currently running
2  quit -sim
3
4  # create the default "work" library
5  vlib work;
6
7  # compile the Verilog source code in the parent folder
8  vlog ../*.v
9  # compile the Verilog code of the testbench
10 vlog *.v
11 # start the Simulator, including some libraries
12 vsim work.testbench -Lf 220model -Lf altera_mf_ver -Lf verilog
13 # show waveforms specified in wave.do
14 do wave.do
15 # advance the simulation the desired amount of time
16 run 120 ns

```

Figure 4: The *testbench.tcl* file.

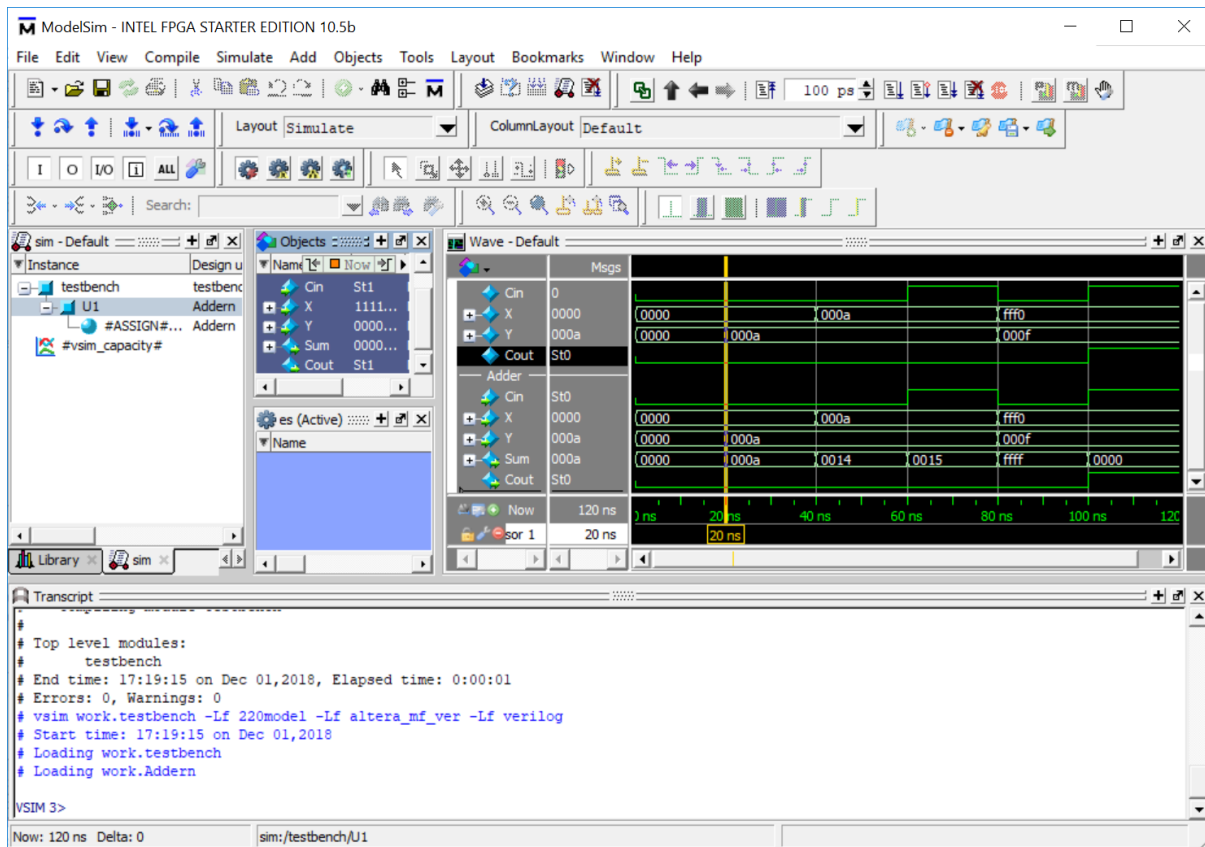


Figure 5: The updated *ModelSim* window.

The *wave.do* file used for this project appears in Figure 6. It specifies, in Lines 3 to 12, which signal waveforms should be displayed in the simulation results, and also includes a number of settings related to the display. To add or delete waveforms in the display, you can manually edit the *wave.do* file, using any text editor, or you can modify which waveforms are displayed within the ModelSim graphical user interface. Referring to Figure 5, changes to the displayed waveforms can be selected by right-clicking in the waveform window. Waveforms can be added to the display by selecting a signal in the *Objects* window and then *dragging-and-dropping* that signal name into the *Wave* window. A more detailed discussion about commands available in the graphical user interface is provided in Appendix A.

Quit the ModelSim software to complete this part of the tutorial. To quit the program you can either select the *File > Quit* command, or just click on the X in the upper-right corner of the ModelSim window.

```

1 onerror {resume}
2 quietly WaveActivateNextPane {} 0
3 add wave -noupdate -label Cin /testbench/Cin
4 add wave -noupdate -label X -radix hexadecimal /testbench/X
5 add wave -noupdate -label Y -radix hexadecimal /testbench/Y
6 add wave -noupdate -label Cout /testbench/Cout
7 add wave -noupdate -divider Adder
8 add wave -noupdate -label Cin /testbench/U1/Cin
9 add wave -noupdate -label X -radix hexadecimal /testbench/U1/X
10 add wave -noupdate -label Y -radix hexadecimal /testbench/U1/Y
11 add wave -noupdate -label Sum -radix hexadecimal /testbench/U1/Sum
12 add wave -noupdate -label Cout /testbench/U1/Cout
13 TreeUpdate [SetDefaultTree]
14 WaveRestoreCursors {{Cursor 1} {20000 ps} 0}
15 quietly wave cursor active 1
16 configure wave -namecolwidth 73
17 configure wave -valuecolwidth 64
18 configure wave -justifyvalue left
19 configure wave -signalnamewidth 0
20 configure wave -snapdistance 10
21 configure wave -datasetprefix 0
22 configure wave -rowmargin 4
23 configure wave -childrowmargin 2
24 configure wave -gridoffset 0
25 configure wave -gridperiod 1
26 configure wave -griddelta 40
27 configure wave -timeline 0
28 configure wave -timelineunits ns
29 update
30 WaveRestoreZoom {0 ps} {120 ns}

```

Figure 6: The *wave.do* file.

Simulating a Sequential Circuit

Another ModelSim project, called *Accumulate*, is included as part of the *design files* provided along with this tutorial. Copy the *Accumulate* project to a folder on your computer, such as *C:\ModelSim_Tutorial\Accumulate*. In the *Accumulate* folder there is a Verilog source-code file called *Accumulate.v* and a subfolder named *ModelSim*. The *Accumulate.v* Verilog code that we will simulate in this part of the tutorial, shown in Figure 7, represents the logic circuit illustrated in Figure 8. This circuit functions as an *accumulator* that adds the contents of the register named *Sum* to itself for each clock cycle until the down-counter reaches 0.

The *Accumulate* module in Figure 7 has ports *KEY*, *CLOCK_50*, *SW*, and *LEDR* because the module is intended to be implemented on a DE-series board. After simulating the design to verify its correct operation, you may wish to compile it using the Quartus Prime CAD tools and then download and test the resulting circuit on a board.

A *testbench.v* file for the accumulator design is given in Figure 9. Lines 5 to 8 declare signals for connecting to the inputs and output of the *Accumulate* module. The testbench code creates a periodic clock signal that is used to simulate our sequential circuit. Line 10 defines a parameter to represent the clock period, which is 20 ns. Lines 12 to 14 initialize the clock signal for the circuit to 0. Then, in Lines 15 to 18 an *always* block is used to specify that after each half clock period the clock signal should be inverted. Since the clock period is defined as 20 simulation time units, this always block generates a 50 MHz periodic clock signal. Lines 20 to 24 set up the *KEY* and *SW* inputs for the simulation, using an initial block.

Reopen the *ModelSim* software to get to the window in Figure 3. Click on the *Transcript* window at the bottom of the figure and then use the `cd` command to navigate to the *ModelSim* project for the accumulator. For example, in our case we would type `cd C:/ModelSim_Tutorial/Accumulate/ModelSim`. Then, in the *Transcript* window type the command `do testbench.tcl`, as you did previously to run the simulation for the *Addern* example. The *testbench.tcl* script for this example is identical to the one shown in Figure 4, except that the last line specifies `run 300 ns`.

The simulation results for our sequential circuit, which displays the waveforms selected in its *wave.do* file, appears in Figure 10.

```

module Accumulate (KEY, CLOCK_50, SW, LEDR);

    input [0:0] KEY;
    input CLOCK_50;
    input [9:0] SW;
    output [9:0] LEDR;

    reg [4:0] Count;
    reg [9:0] Sum;
    wire Clock, Resetn, z;
    wire [4:0] X, Y;

    assign Clock = CLOCK_50;
    assign Resetn = KEY[0];
    assign X = SW[4:0];
    assign Y = SW[9:5];

    // the accumulator
    always @(posedge Clock)
        if (Resetn == 1'b0) // synchronous clear
            Sum <= 0;
        else if (z == 1'b1)
            Sum <= Sum + X;

    // the counter
    always @(posedge Clock)
        if (Resetn == 1'b0) // synchronous load
            Count <= Y;
        else if (z == 1'b1)
            Count <= Count - 1'b1;

    assign z = | Count;
    assign LEDR = Sum;
endmodule

```

Figure 7: Verilog code for the accumulator.

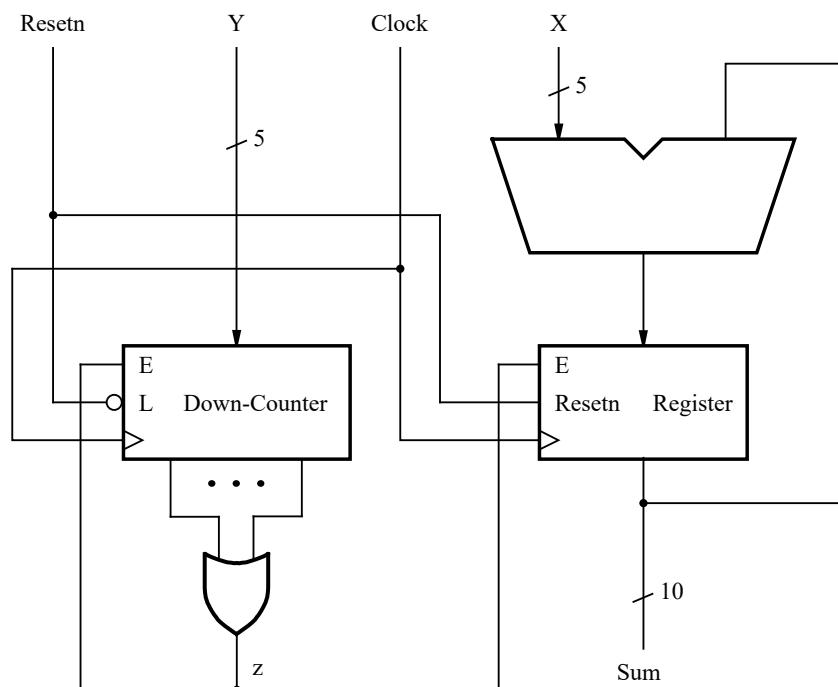


Figure 8: The accumulator circuit.

```

1  `timescale 1ns / 1ps
2
3  module testbench ( );
4
5      reg [0:0] KEY;
6      reg CLOCK_50;
7      reg [9:0] SW;
8      wire [9:0] LEDR;
9
10     parameter CLOCK_PERIOD = 20;
11
12     initial begin
13         CLOCK_50 <= 1'b0;
14     end // initial
15     always @ (*)
16     begin : Clock_Generator
17         #((CLOCK_PERIOD) / 2) CLOCK_50 <= ~CLOCK_50;
18     end
19
20     initial begin
21         KEY[0] <= 1'b0; SW <= 10'h0;
22         #20 SW[4:0] <= 5'b11110; SW[9:5] <= 5'b01010;
23         #40 KEY[0] <= 1'b1;
24     end // initial
25
26     Accumulate U1 (KEY, CLOCK_50, SW, LEDR);
27
28 endmodule

```

Figure 9: The Verilog testbench code for the sequential circuit.

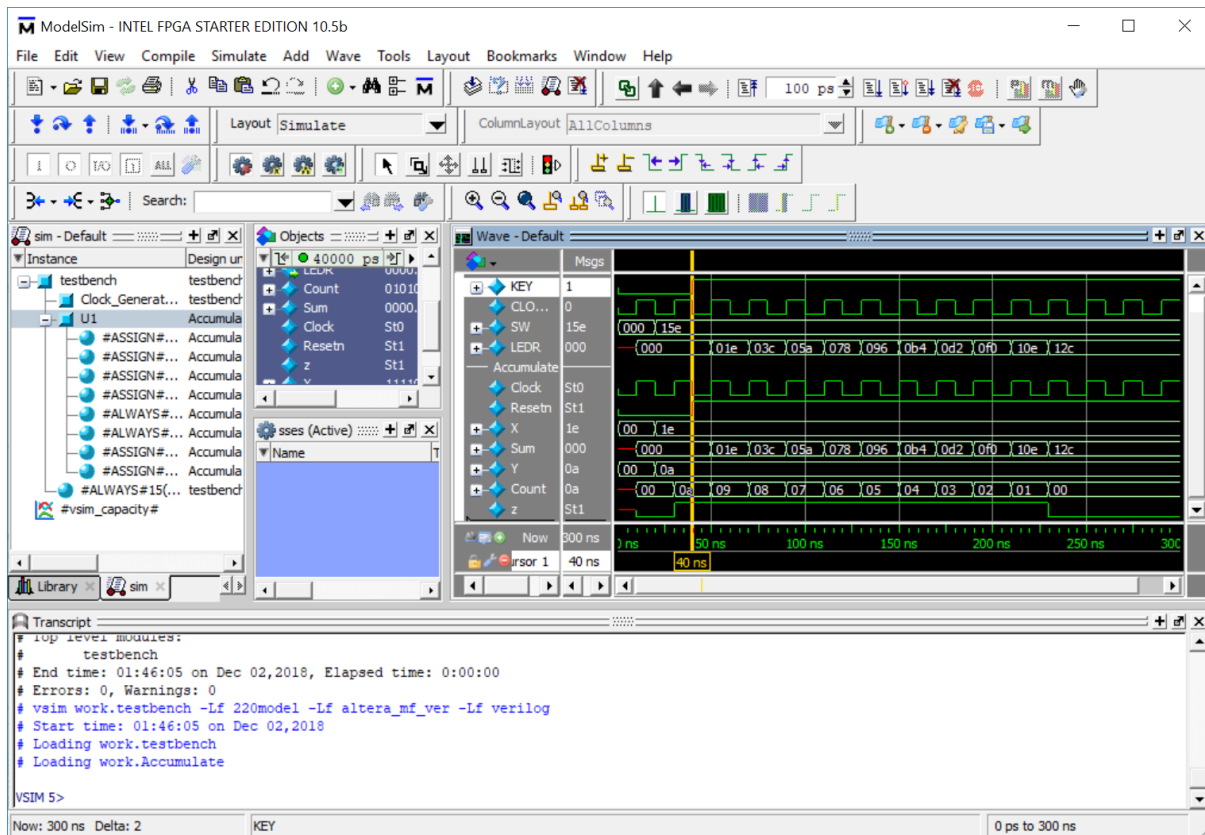


Figure 10: The simulation results for our sequential circuit.

Creating a New ModelSim Project

A easy way to set up your own ModelSim project is to use the files provided here as a starting point, as follows. Within the folder where your Verilog source-code files are stored make a subfolder named *ModelSim*. Copy into this subfolder the files *testbench.v*, *testbench.tcl*, and *wave.do* from one of the examples presented above. Then, modify *testbench.v* to create whatever waveforms you need, and to instantiate your top-level Verilog module. The *testbench.tcl* script can be used for the new project without any changes, except that you might want to specify a different amount of simulation time in the `run` command on the last line of the script. Finally, modify the *wave.do* file to choose the waveforms that should be displayed. You can change the *wave.do* file manually by editing it with a text editor, or you can make use of the commands available in the ModelSim graphical user interface, as discussed in Appendix A.

Appendix A: Using the ModelSim Graphical User Interface

This appendix illustrates some of the features available in the ModelSim graphical user interface for displaying waveforms. We will show how to add waveforms to the ModelSim window, and how to change the properties of a waveform, such as its displayed name and number radix.

As an example we will show how waveforms can be added to the ModelSim display for the *accumulator* circuit from the previous section. Figure 11 shows the ModelSim waveform display window for this circuit before any signals have been selected.

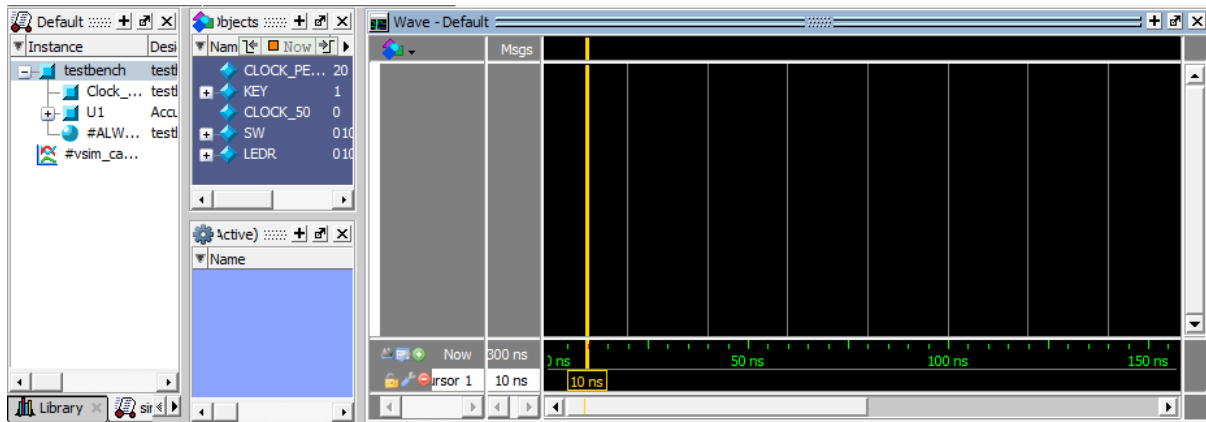


Figure 11: The *ModelSim* waveform display.

In Figure 12 we have selected a waveform, as follows. First, we clicked on the *testbench* module in the upper left part of the display. As a result of this action the signals that exist in the selected module are listed in the *Objects* pane (the area with the dark blue background). In this list we then used the left mouse button to *drag-and-drop* the *KEY* signal name from the *Objects* list into the *Wave* window. Then, as illustrated in the figure, we right-clicked on the name of the signal in the *Wave* display, which is */testbench/KEY*, and then clicked on *Properties* to open the window in Figure 13.

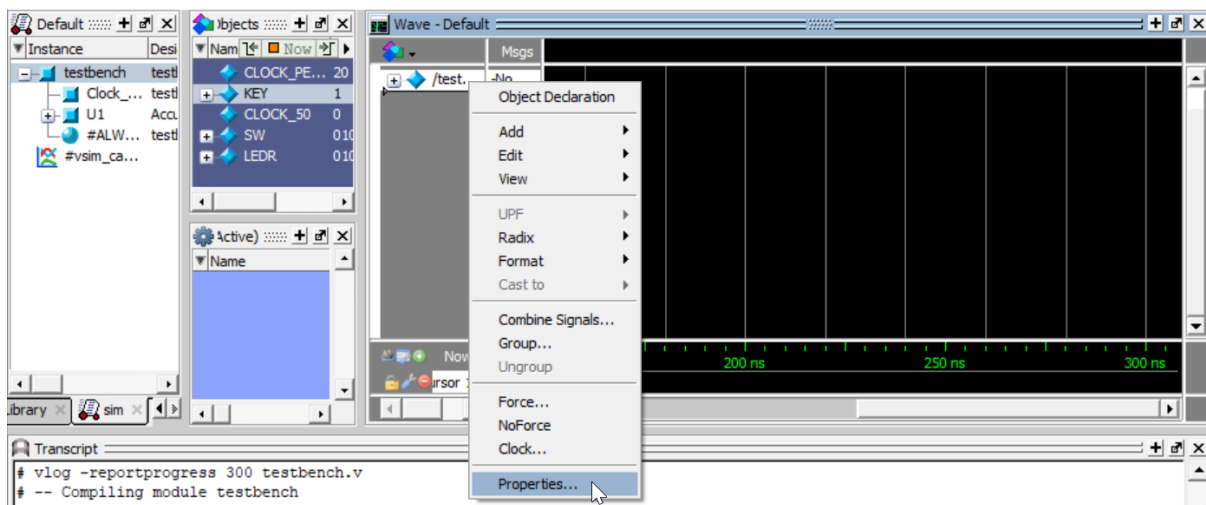


Figure 12: Adding a waveform from the *testbench* module.

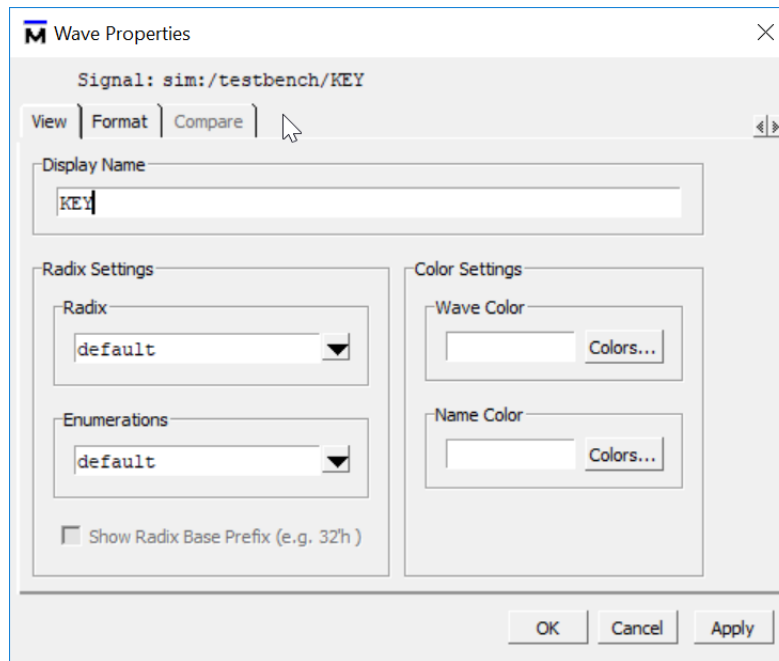


Figure 13: Specifying a display name for a waveform.

In Figure 13 we assigned the name KEY to the waveform, clicked `Apply` and then closed this dialogue. We then used the same drag-and-drop mechanism to add the signals `CLOCK_50`, `SW`, and `LEDR` from the `testbench` module to the Wave window, and set convenient display names for these waveforms. The updated Wave window is shown in Figure 14.

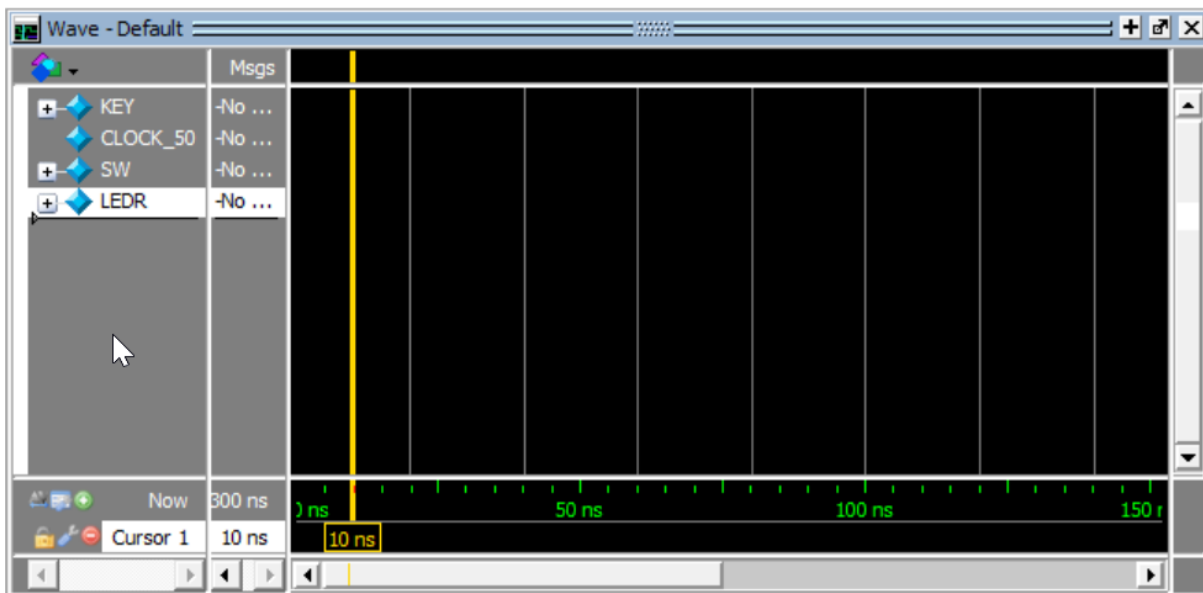


Figure 14: The waveform display after adding more *testbench* signals.

Next, we wish to add signals from the `Accumulate` module to the Wave window. But first we can add a *divider*, as a visual aid that separates the testbench signals and the `Accumulate` module signals. A divider can be added by right-clicking on the Wave window, as indicated in the Figure 15, clicking on Add in the pop-up menu, and then selecting New Divider to open the window in Figure 16.

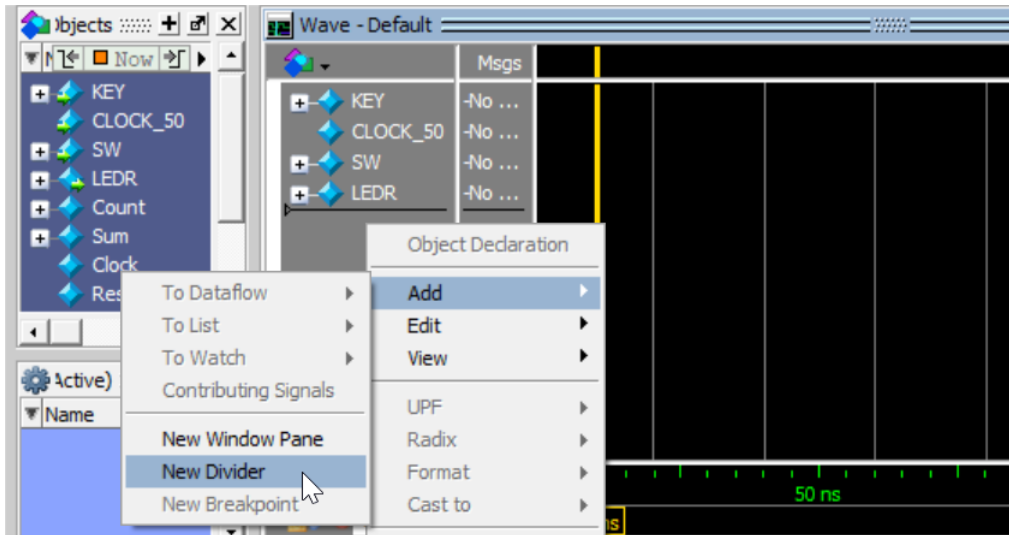


Figure 15: Adding a divider to the waveform display.

We assigned `Accumulate` as the divider name in Figure 16, and then closed this dialogue. The Wave window now appears as shown in Figure 17.

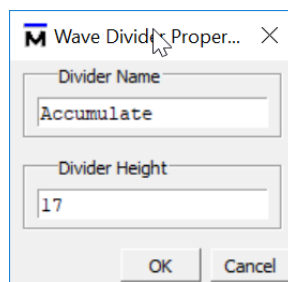


Figure 16: Assigning a name to the divider.

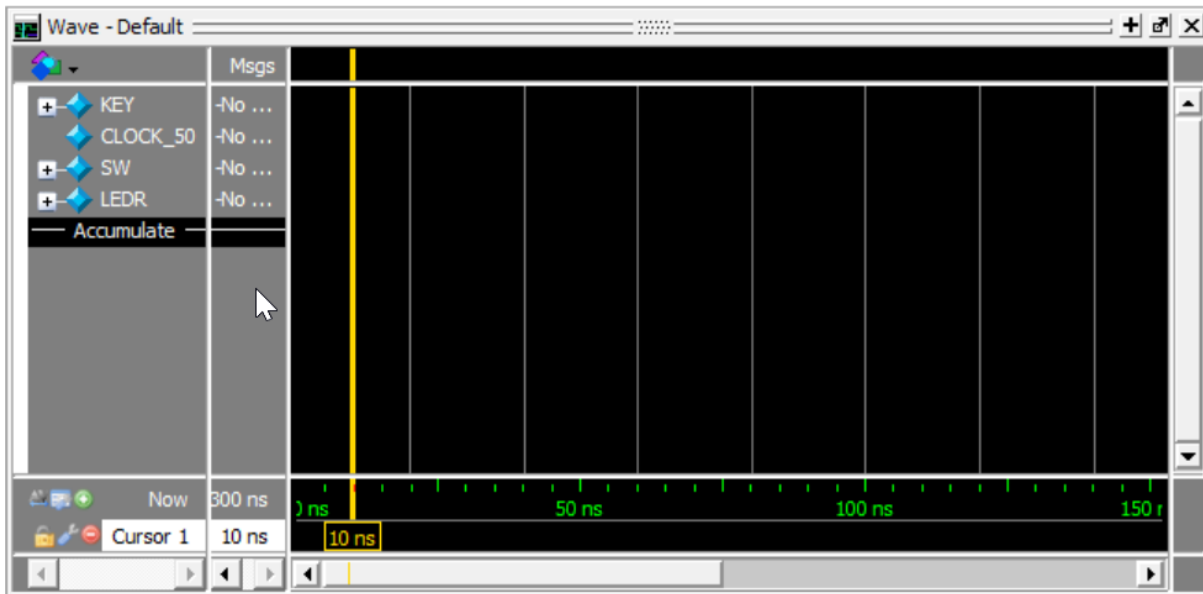


Figure 17: The waveform display after adding the Accumulate divider.

To add signals from the Accumulate module to the Wave window we need to click on the U1 instance name of the Accumulate model, as indicated on the left-hand side of Figure 18. The signals available in this module are then listed in the Objects pane. To obtain the display in the figure, we used the drag-and-drop mechanism, and the Wave Properties dialogue, described previously, to add the Clock, Resetn, X, Sum, Y, Count, and z signals to the Wave window.

We now wish to perform a simulation of our testbench so that waveforms will be generated and shown for our selected signals. However, it is *critical* to first *save* the selections that have been made in the Wave display to the *wave.do* file. If you run a simulation *without* first performing a save to the *wave.do* file, then all changes made to the Wave window will be discarded and lost!

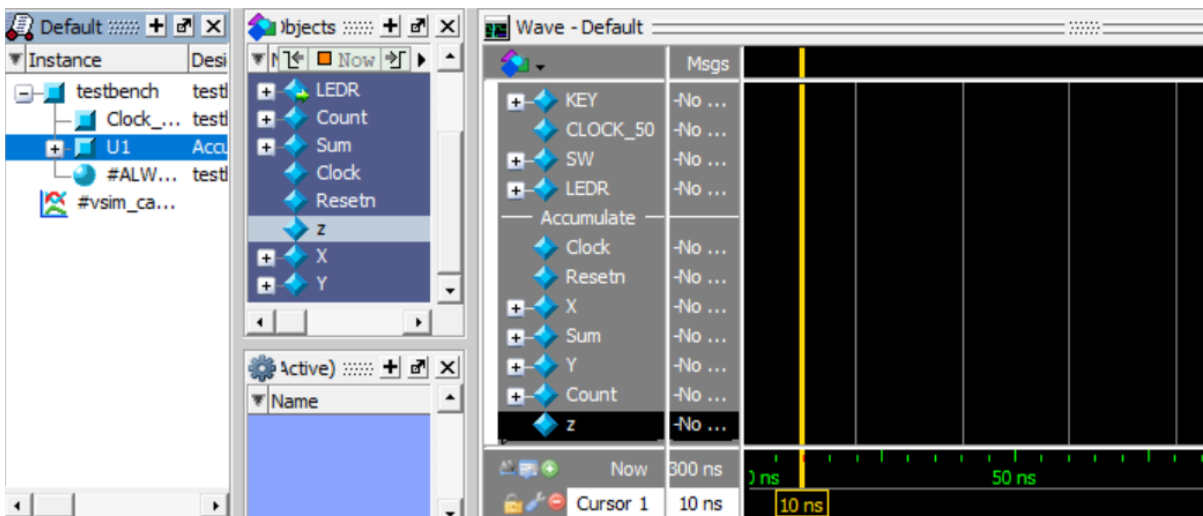


Figure 18: The waveform display after adding signals from the accumulate module.

The command `File > Save Format` opens the dialogue shown in Figure 19. After clicking OK and then over-writing the *wave.do* file, the testbench simulation can be executed by typing the command `do testbench.tcl`. The resulting waveform display is illustrated in Figure 20. In this figure we right-clicked on the Wave window and selected `Zoom Range` to open the dialogue in Figure 21. As indicated in the figure, we select a time range from 0 to 300 ns for the Wave display.

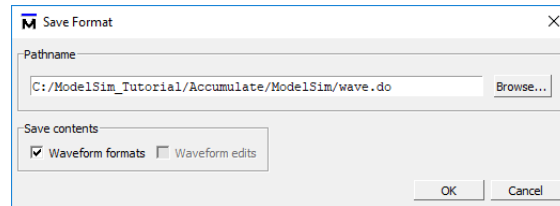


Figure 19: The Save Format dialogue.

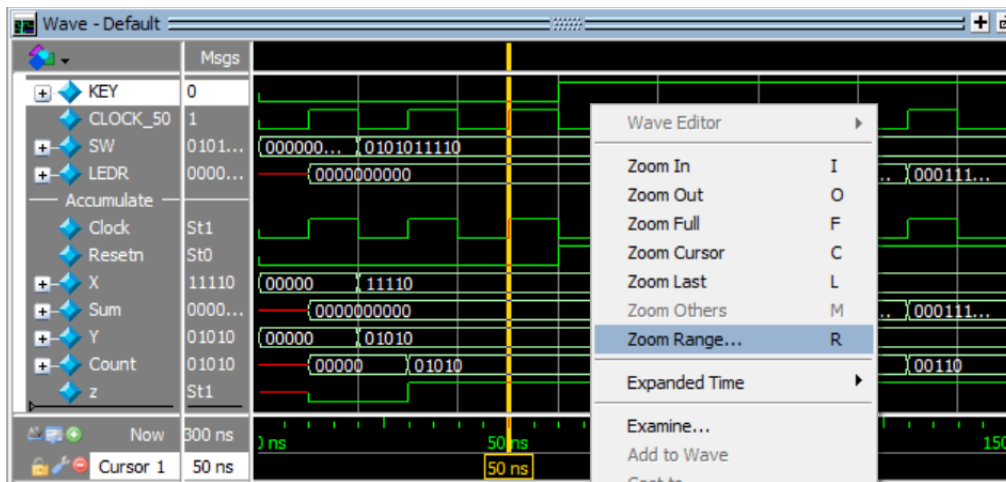


Figure 20: The display after running the simulation; changing the zoom range.

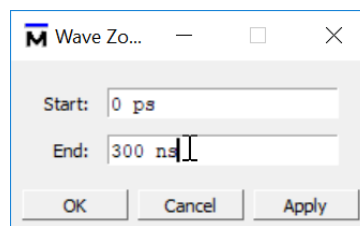


Figure 21: Inputting the zoom range.

To make it easier to see the values of signals in the Wave window, you can select radices other than binary, which is the default. For example, in Figure 22 we right-clicked on the SW signal, clicked on Radix, and then selected Hexadecimal. After setting the radix to hexadecimal for several additional signals, the final Wave display appears as illustrated in Figure 23.

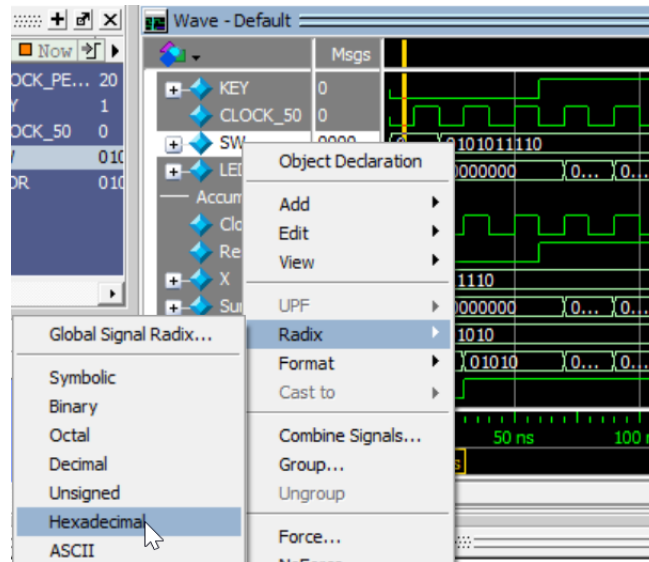


Figure 22: Setting the radix for a waveform.

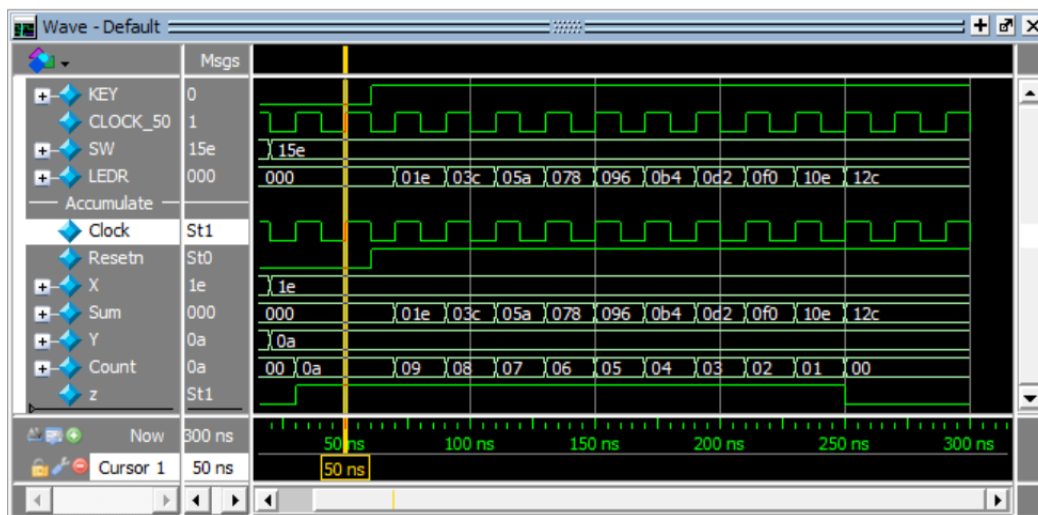


Figure 23: The final waveform display.

This tutorial has described only a subset of the commands that are provided in the ModelSim graphical user interface. Although a discussion of other available commands is beyond the scope of this tutorial, a number of more detailed ModelSim tutorials can be found by searching for them on the Internet.

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Avalon, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.