



## Timing Simulation of Verilog Designs

For Quartus II 12.0

### 1 Introduction

An introduction to functional simulation using Altera's University Program Simulation Tool, called *Qsim*, is given in the tutorial *Introduction to Simulation of Verilog Designs Using Altera's University Program Simulator*, which is available on Altera's University Program web site. A knowledge of the material in that tutorial is essential for understanding the material that follows. The present tutorial shows how the simulator can be used to perform timing simulation of a circuit specified in Verilog HDL. A basic understanding of Verilog is needed for this purpose.

#### Contents:

- Design Project
- Creating Waveforms for Simulation
- Simulation
- Concluding Remarks

The Qsim simulation tool works in conjunction with Altera's Quartus II software. It can be used only on a computer that has Quartus II software installed. The simulation tool allows the user to apply inputs to the designed circuit, usually referred to as *test vectors*, and to observe the outputs generated in response. The Qsim tools include a graphical interface for creating the input waveforms.

In this tutorial, the reader will learn about:

- Test vectors needed to test the designed circuit
- Using Qsim to draw test vectors
- Timing simulation, which is used to verify the timing of signals in a synthesized circuit

This tutorial is aimed at the reader who wishes to simulate circuits defined by using the Verilog hardware description language. An equivalent tutorial is available for the user who prefers the VHDL language.

## 2 Design Project

To provide an example of timing simulation, we will use a an  $n$ -bit adder circuit that incorporates registers on both the input and output sides. The circuit is shown in Figure 1. Inputs to the circuit are two  $n$ -bit numbers,  $X$  and  $Y$ , which are provided on the input pins of the FPGA device. The  $n$ -bit sum,  $Z = X + Y$ , is available on the output pins. A clock signal, *Clock*, is used to load the data into the registers. We will assume that the positive (0-to-1) edge of the clock signal is used to load the data.

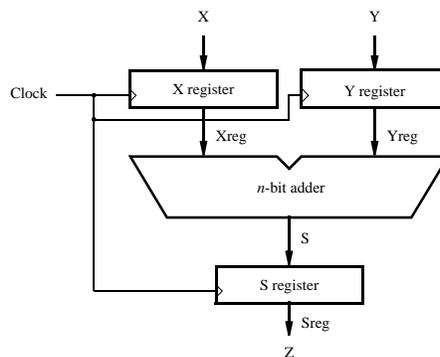


Figure 1. An example circuit.

Figure 2 gives Verilog code for the circuit in Figure 1. Observe that the length of the operands is 32 bits, as specified in the parameter statement. Note also the “synthesis keep” comment included in the statement that specifies the output,  $S$ , of the adder. Inclusion of this comment informs the compiler to keep the signal name  $S$  when implementing the circuit. Otherwise, the compiler will keep the original names of only those signals that appear on the pins or on outputs of various registers, and it may change the names of other signals. We wish to observe the  $n$ -bit signal  $S$ ,

because we are interested in determining the delay that occurs in generating the sum  $S$ , which indicates the speed of the  $n$ -bit adder combinational circuit.

```
module adder (X, Y, Z, Clock);
  parameter n = 32;
  input [n-1:0] X, Y;
  input Clock;
  output [n-1:0] Z;
  reg [n-1:0] Xreg, Yreg, Sreg;
  reg [n-1:0] S /* synthesis keep */;

  assign Z = Sreg;
  always @(X, Y)
    S = Xreg + Yreg;
  always @(posedge Clock)
  begin
    Xreg <= X;
    Yreg <= Y;
    Sreg <= S;
  end
endmodule
```

Figure 2. Verilog code for the circuit in Figure 1

The desired circuit has to be first implemented in a Quartus II project. To do so, create a new directory (folder) for the Quartus II project, and for consistence with the description in this tutorial call it *simulator\_timing*. Enter the code in Figure 2 into a file called *adder.v*, and save this file in the *simulator\_timing* directory. Create a Quartus II project called *adder* and include *adder.v* as part of the project. Compile your project.

Open the Qsim tool, which will display the window in Figure 3. In this window select File > Open Project, which leads to a pop-up window in which you should choose the *adder* Quartus II project that you created.

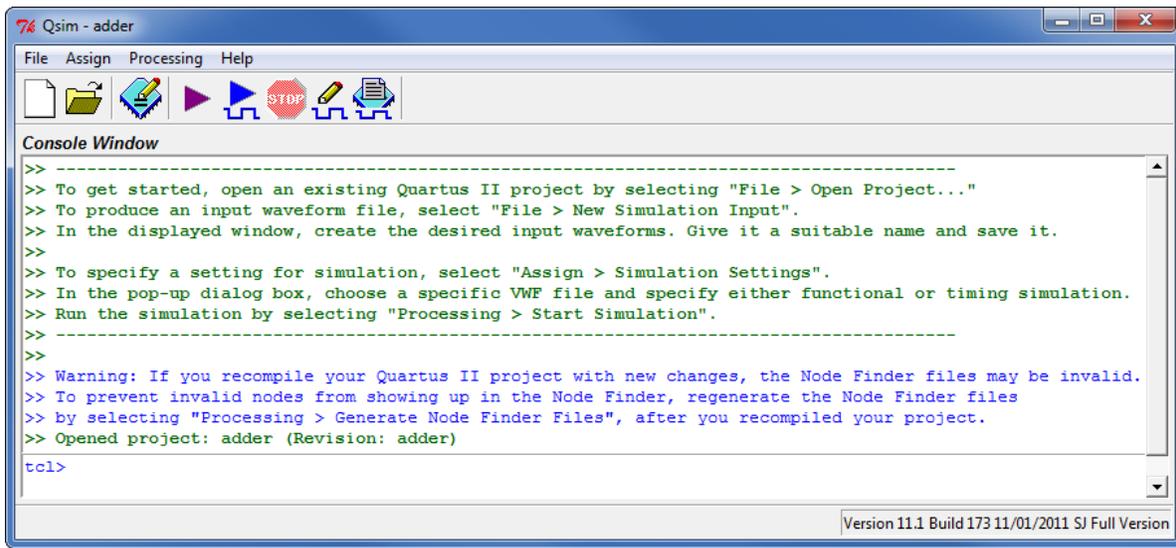


Figure 3. The Qsim window.

### 3 Creating Waveforms for Simulation

To create test vectors for your design, select the Qsim command File > New Simulation Input File. This command opens the *Waveform Editor* tool, which allows you to specify the desired input waveforms. Figure 4 displays the Waveform Editor window.

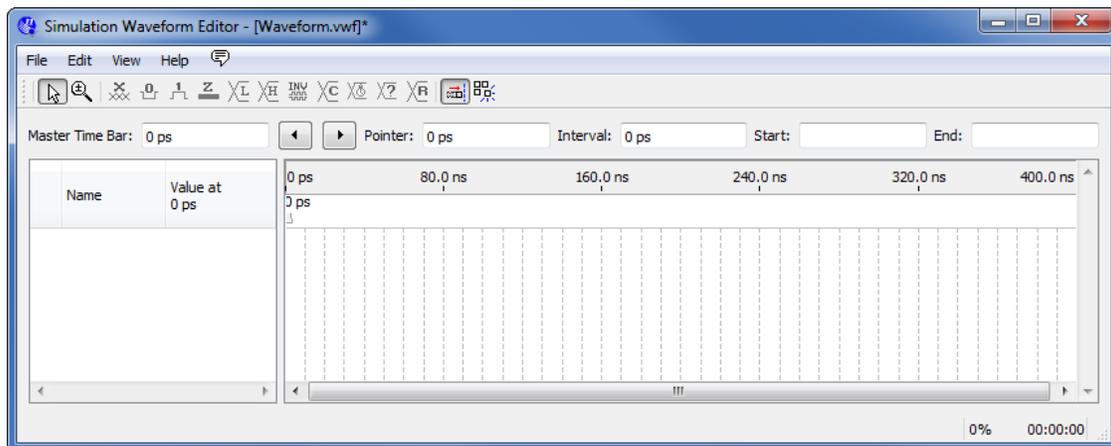


Figure 4. The Waveform Editor window.

We will run the simulation for 400 ns; so, select Edit > Set End Time and in the pop-up window that will appear specify the time of 400 ns and click OK. This will adjust the time scale in the window in Figure 4.

Before drawing the input waveforms, it is necessary to locate the desired signals, referred to as *nodes*, in the implemented circuit. This is done by using a utility program called the Node Finder. In the Waveform Editor window, select Edit > Insert > Insert Node or Bus. In the pop-up window that appears, which is shown in Figure 5, click on Node Finder.

The Node Finder window is presented in Figure 6. A filter is used to identify the nodes of interest. In our circuit, we are interested not only in the nodes that appear on the pins (i.e. external connections) of the FPGA chip, but also in some internal nodes that are important in assessing the various signal propagation delays in the designed circuit. For example, it is of interest to determine the propagation delay through the adder combinational circuit, namely the time it takes for the sum bits, *S*, to be generated after the signals *Xreg* and *Yreg* have been applied.

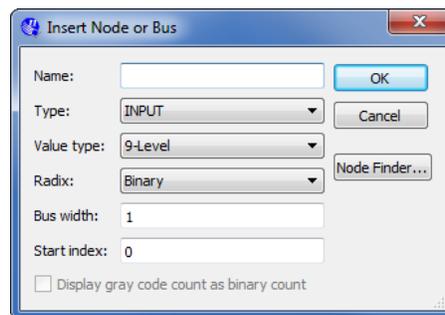


Figure 5. The Insert Node or Bus dialog.

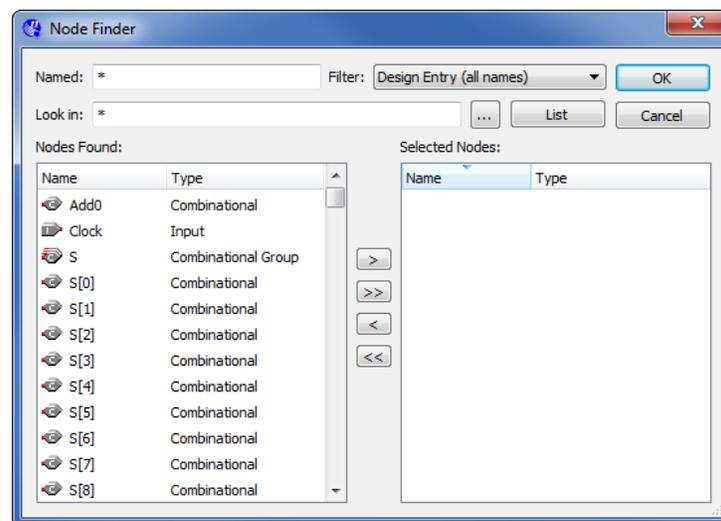


Figure 6. The Node Finder dialog.

Set the filter to Design Entry (all names). Click on the List button, which will display the nodes as indicated in Figure 6. In a large circuit there could be many nodes displayed. We need to select the nodes that we wish to observe

in the simulation. This is done by highlighting the desired nodes and clicking on the > button. Select the nodes labeled *Clock*, *X*, *Y*, *Xreg*, *Yreg*, *S*, *Sreg*, and *Z*, which will lead to the image in Figure 7. Click OK in this window and also upon return to the window in Figure 5. This returns to the Waveform Editor window, with the selected signals included as presented in Figure 8.

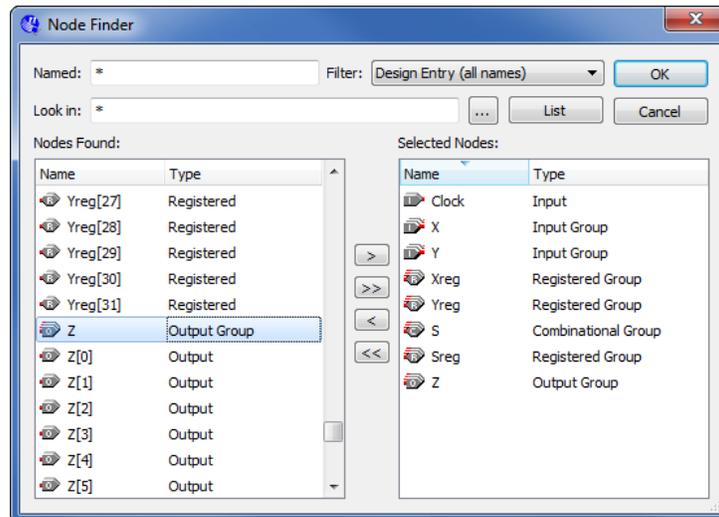


Figure 7. The selected signals.

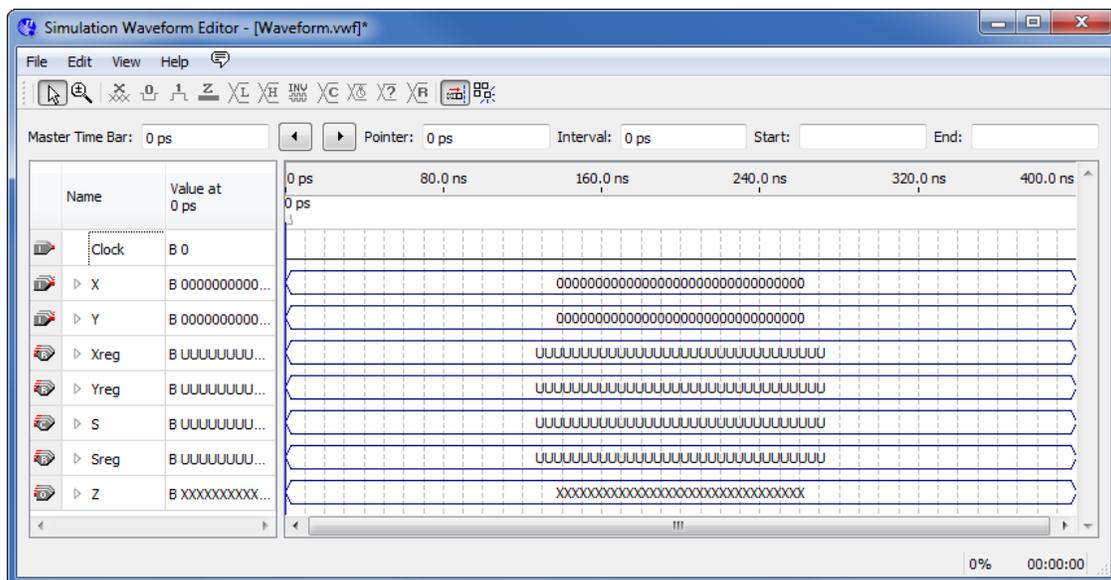


Figure 8. Signals in the Waveform Editor.

Observe that in Figure 8 all input signals have a value 0, while the values of the rest of the signals are undefined. All signals except *Clock* are represented as 32-bit vectors, referred to as *groups* in the Node Finder window. It is awkward to display these vectors as binary values. Since we are dealing with an adder circuit, it is most convenient to use the signed decimal representation. To use this notation, right-click on the name of the *X* signal, and in the drop-down box choose **Radix > Signed Decimal** as indicated in Figure 9. This will replace the 32-bit vector with the number 0. Do the same for all other vectors, which should produce the image in Figure 10.

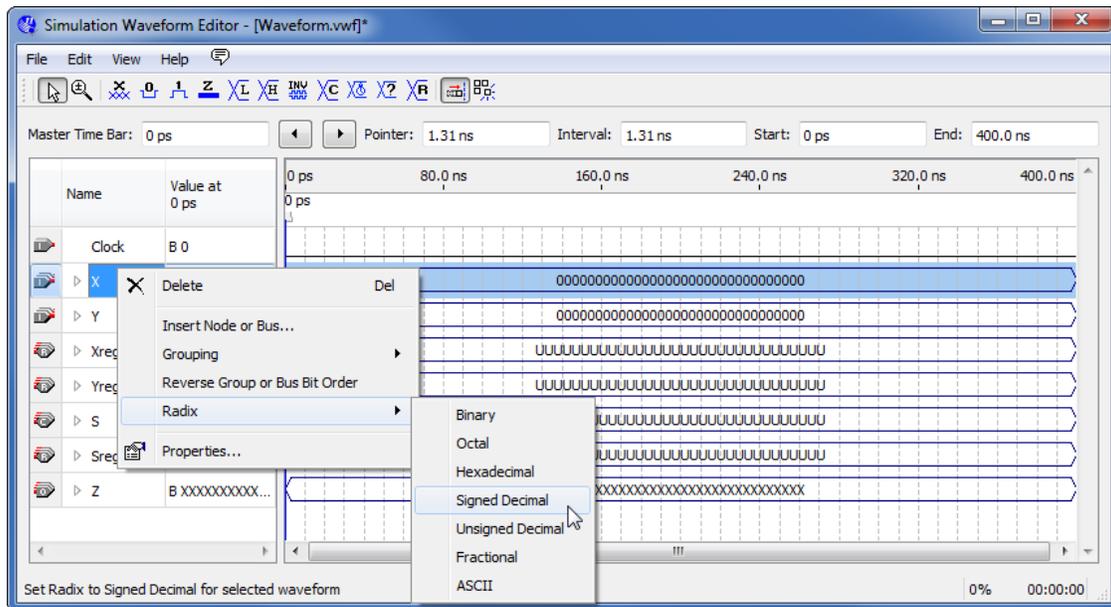


Figure 9. Choosing the number representation.

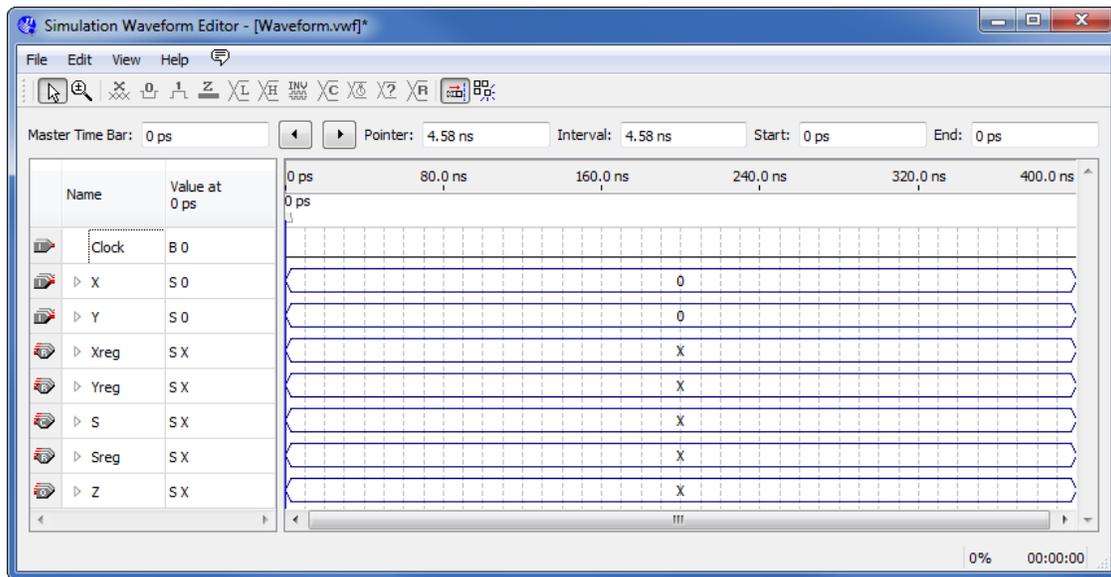


Figure 10. The modified Waveform Editor window.

Now, we can specify the input waveforms. First, specify the clock signal. This signal has to alternate between logic values 0 and 1 at periods defined by the clock frequency used in the designed circuit. We will assume that the clock period is 80 ns. Click on the *Clock* input, which selects the entire 400-ns interval. Then, click on the **Overwrite Clock** icon , as indicated in Figure 11. This leads to the pop-up window in Figure 12. Specify the clock period of 80 ns and the duty cycle of 50%, and click OK. The result is depicted in Figure 13.

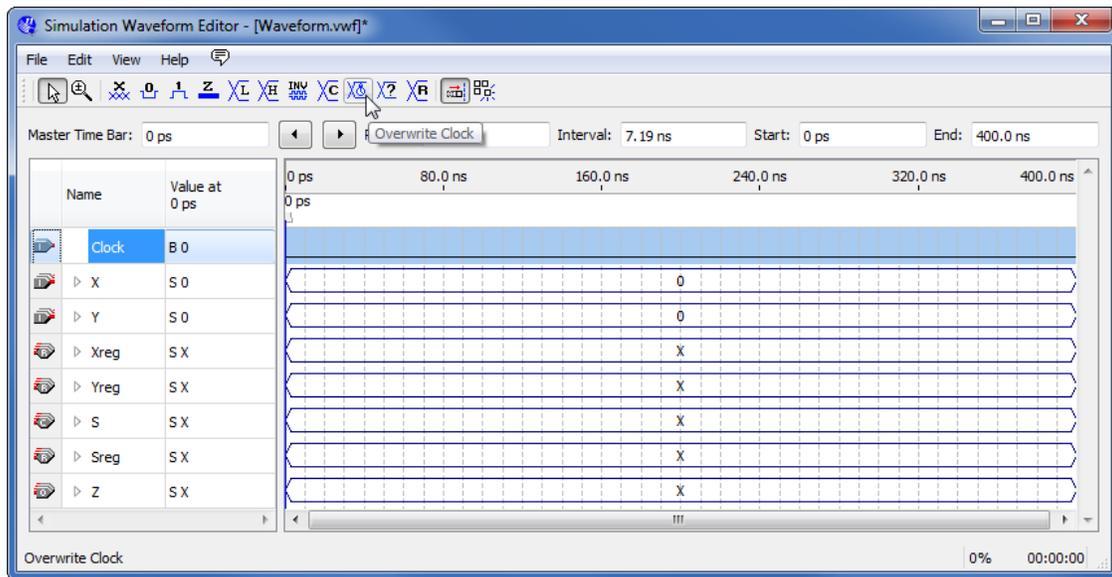


Figure 11. Selecting the Clock signal.

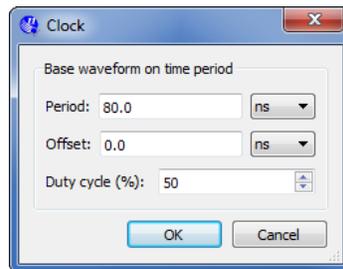


Figure 12. Setting the clock parameters.

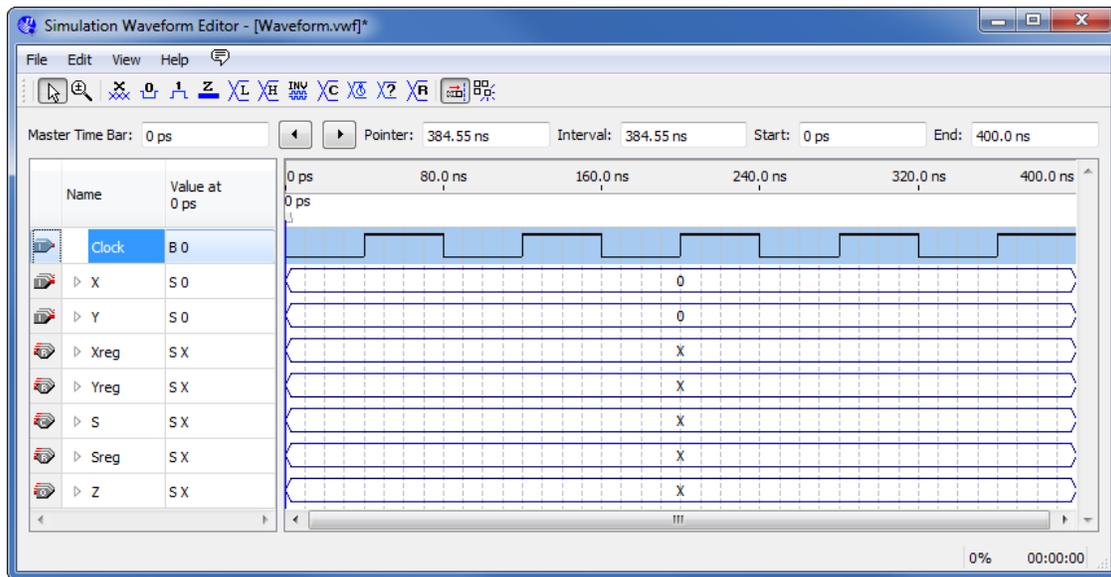


Figure 13. Waveform for the Clock signal.

Next, we have to specify the test values for the input vectors  $X$  and  $Y$ . We will choose several different values to see the effect of the propagation delays through the adder circuit. Initially, both  $X$  and  $Y$  are equal to 0. Let us start by setting  $X = 2$  and  $Y = 1$  at the time  $t = 20$  ns, and maintaining these values until  $t = 100$  ns. To do this, click the mouse on the  $X$  waveform at the 20-ns point and then drag the mouse to the 100-ns point. The selected time interval will be highlighted, as indicated in Figure 14. Then, click on the Arbitrary Value icon , as shown in the figure, which leads to the pop-up window in Figure 15. Here, make sure that the signed-decimal radix is selected, enter the value 2, and click OK. The result is depicted in Figure 16.

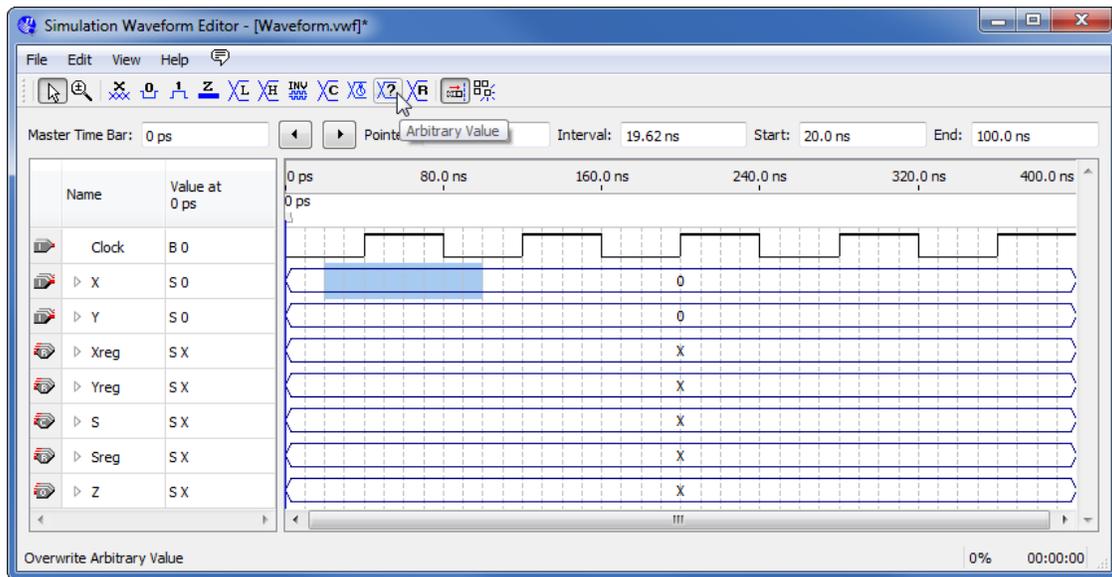


Figure 14. Selecting a range for signal X.

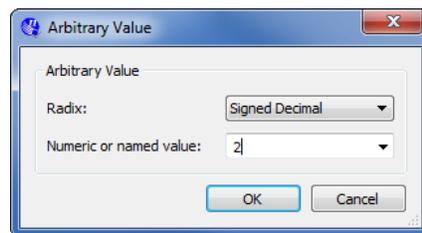


Figure 15. Setting the value of signal X.

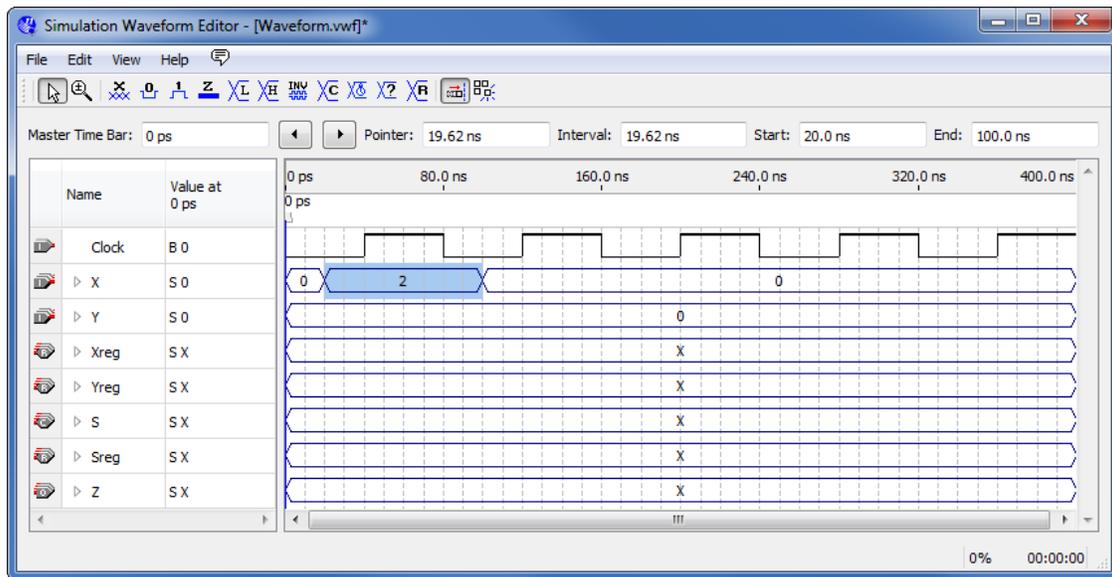


Figure 16. The value of signal *X* is set to 2.

Next, use the same procedure to set the values of *X* to 60007, 3, and 8000015, in the intervals 100-180 ns, 180-260 ns, and 260-400 ns, respectively. Similarly, set the values for *Y* to 1, -1, and -15, in the intervals 20-180 ns, 180-260 ns, and 260-400 ns, respectively. This should produce the image in Figure 17. Save the waveform file using a suitable name; we chose the name *adder.vwf*. Note that the suffix *vwf* stands for *vector waveform file*.

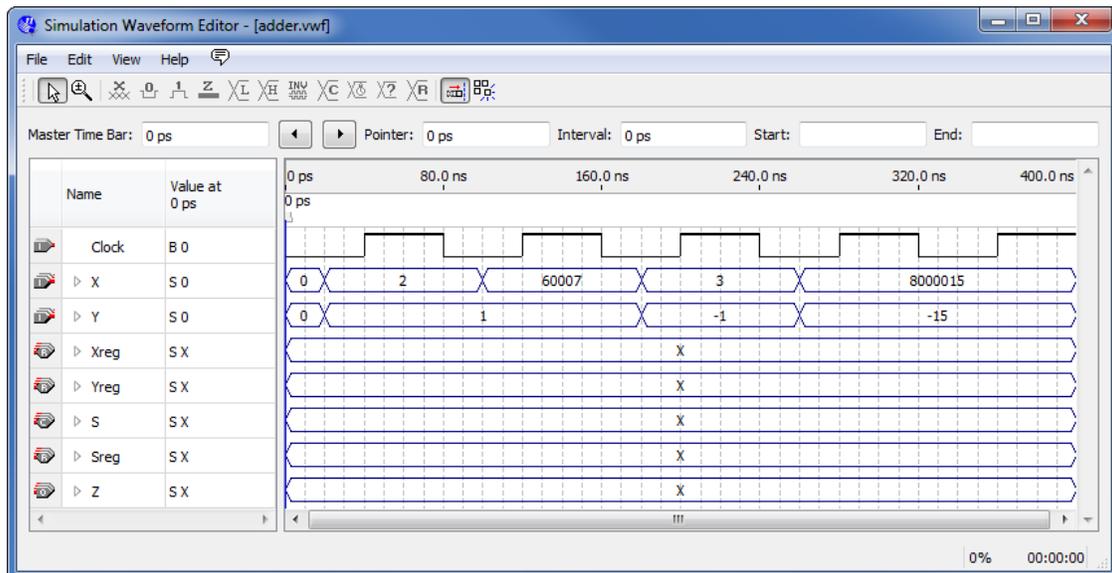


Figure 17. Setting the values of signals *X* and *Y*.

## 4 Simulation

There are two possibilities for simulating a circuit: functional and timing simulation. To see the difference between these possibilities, we will first perform the functional simulation.

### 4.1 Functional Simulation

We have to select the type of simulation to be performed. Return to the Qsim window (in Figure 3). Select **Assign > Simulation Settings**, which displays the Simulation Settings window in Figure 18. Here, you have to select the .vwf file that contains the input waveforms. Browse to find the file *adder.vwf*, choose **Functional** as the simulation type, and click **OK**.

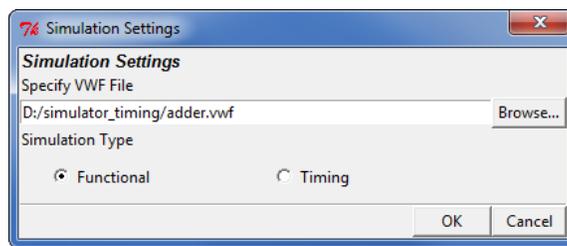


Figure 18. Choosing the type of simulation.

To enable the functional simulation to be performed, it is necessary to generate a *functional netlist* of the circuit. This netlist specifies the logic elements and the connections needed to implement the circuit. In the Qsim window, select **Processing > Generate Simulation Netlist**, or click on the icon .

Now, we can simulate the circuit. Select **Processing > Start Simulation**, or click on the icon . A pop-up window will indicate that “simulator was successful”. Click **OK**. Another pop-up window will state that “the file is read-only and cannot be edited”. This states that the output of the simulation is a file that you cannot alter. Any changes in simulation have to be done by modifying the *adder.vwf* file and resimulating the circuit. Click **OK**. Qsim will now display the waveforms produced in the simulation process, which are depicted in Figure 19.

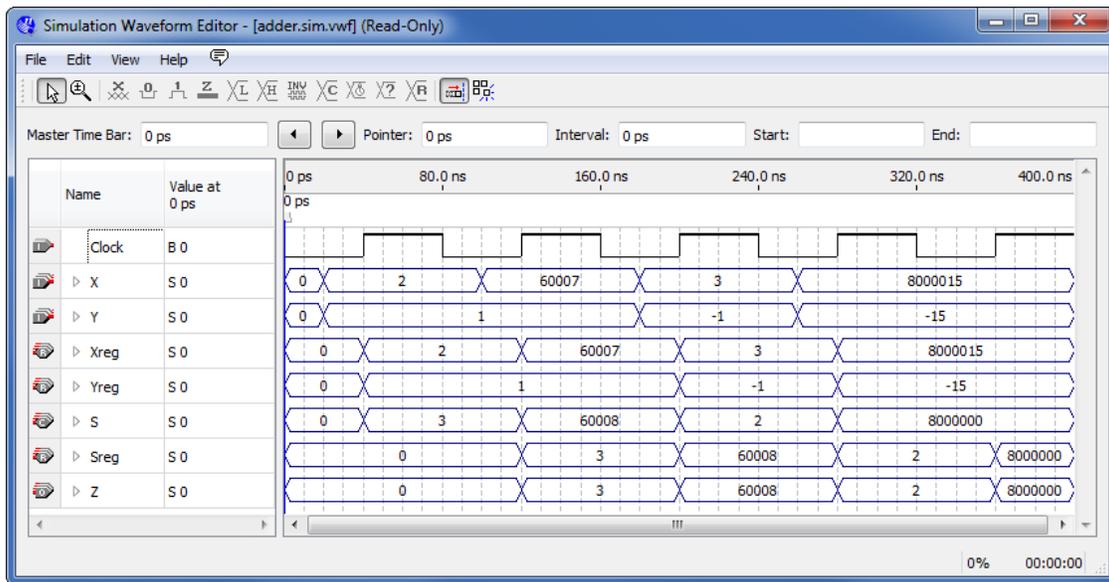


Figure 19. Result of the functional simulation.

Observe that the changes in signals that propagate through our circuit occur exactly at the positive edges of the clock. The propagation delay through the combinational adder, namely from *Xreg* and *Yreg* to *S*, is shown to be zero because we are using functional simulation.

## 4.2 Timing Simulation

To observe the actual propagation delays in our circuit, we have to perform a timing simulation. In the window of Figure 18, choose Timing to be the simulation type. To enable the timing simulation to be performed, it is necessary to generate a *timing netlist* of the circuit. This netlist specifies the logic elements and the connections needed to implement the circuit. In the Qsim window, select Processing > Generate Simulation Netlist, or click on the icon . Then, simulate the circuit again by clicking on the icon . The result is shown in Figure 20.

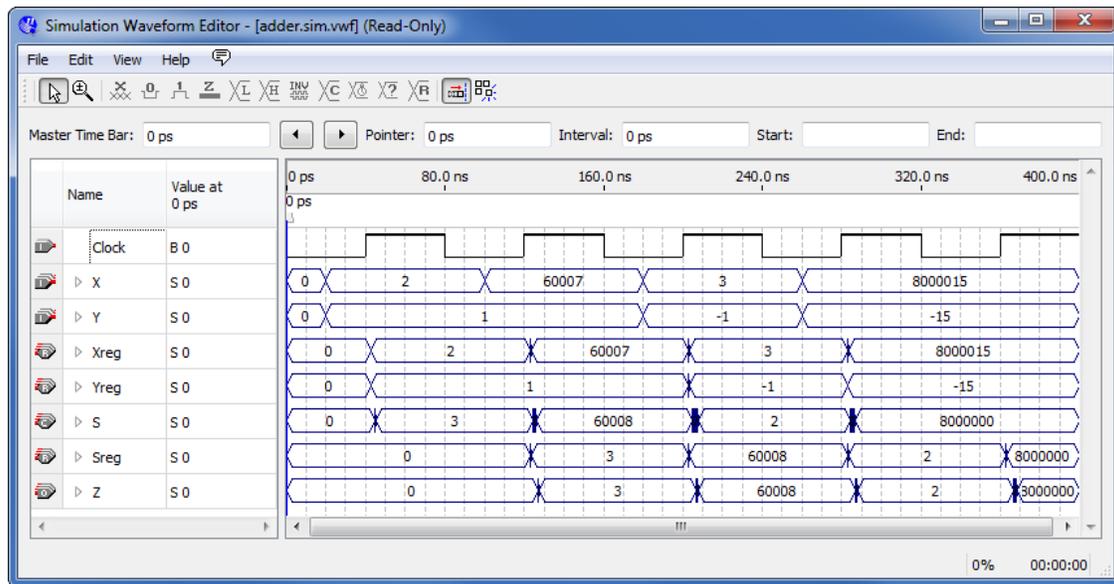


Figure 20. Result of the timing simulation.

The timing simulation shows that there are delays when signals change from one value to another. Some delays are longer than others. The longest delay seems to occur following the clock edge at the 200-ns point. To see this delay more clearly, zoom in on the waveforms by clicking the Zoom Tool icon  and enlarging the image by clicking the mouse button. Figure 21 depicts the enlarged image. Observe the value of the S signal, which is indicative of the propagation delay through the adder when the inputs X and Y are such that a carry signal has to propagate through most of the adder stages before the value of S becomes valid and stable. The displayed image indicates that the maximum delay is about 7 ns.

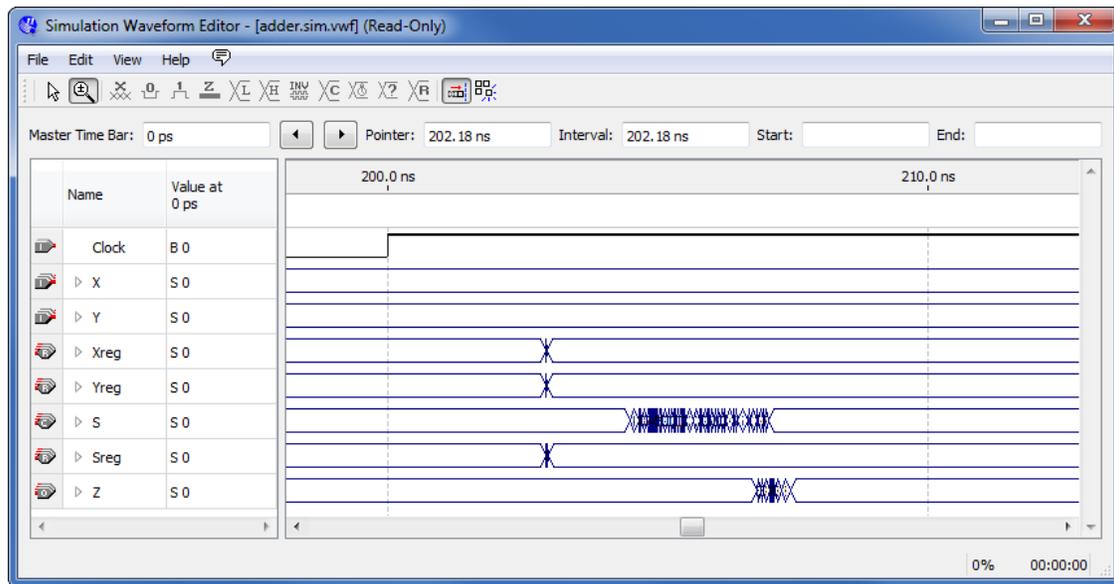


Figure 21. Enlarged image around the 200-ns point in time.

To see the value of this delay more accurately, we can use a *reference line* provided in the Waveform Editor. First click on the Selection Tool icon , which will turn off the Zoom Tool. Then, double-click at the 200-ns clock edge, in the space between the time line and the *Clock* waveform. A vertical blue line will appear at the 200-ns point. Now, we wish to set another reference line at the point where the *S* signal becomes stable. Double-click near this point to display a second reference line. You can position this reference line more precisely by clicking on its square tip and dragging it to the desired location. To do this, you should first turn off the Snap to Grid feature, which is selected by the icon . Note that another transition of interest is when the input signals *X* and *Y* are loaded into the respective registers, as depicted by the change in the *Xreg* and *Yreg* values. Display a third reference line at that point. This should produce the image in Figure 22. The simulation result indicates that it takes almost 3 ns to load the new values of *X* and *Y* into the respective registers, and another 4 ns to produce a correct sum *S*.

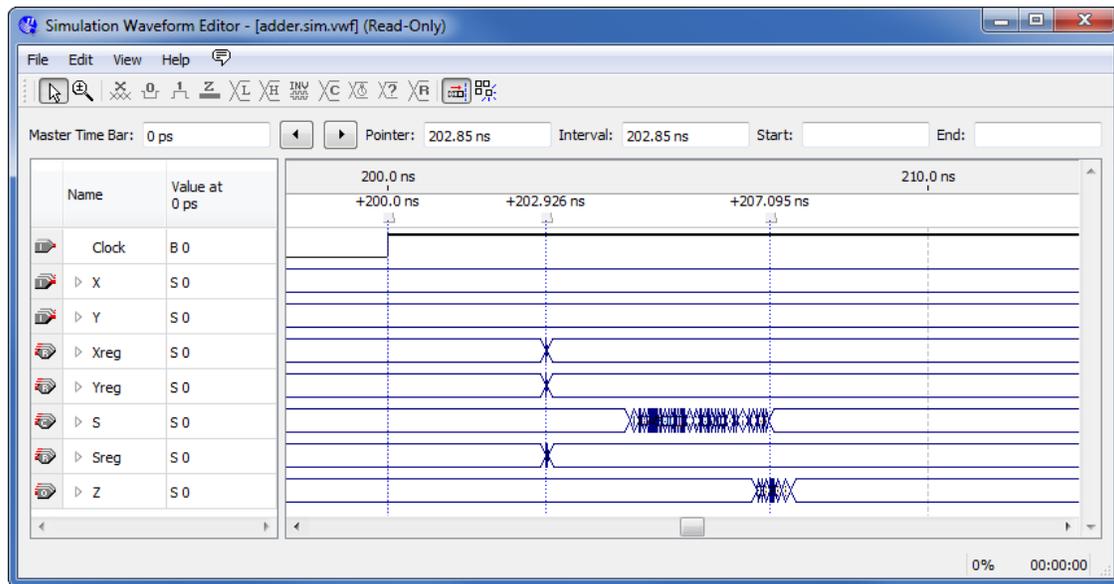


Figure 22. Time reference lines in the waveform image.

## 5 Concluding Remarks

The purpose of this tutorial is to show how timing simulation can be used to observe the propagation delays in a designed circuit.

Copyright ©2011 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera 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 Altera Corporation. Altera 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.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.