Quartus II Simulation with Verilog Designs

This tutorial introduces the basic features of the Quartus® II Simulator. It shows how the Simulator can be used to assess the correctness and performance of a designed circuit.

Contents:

Quartus® II software includes a simulator which can be used to simulate the behavior and performance of circuits designed for implementation in Altera's programmable logic devices. The simulator allows the user to apply test vectors as inputs to the designed circuit and to observe the outputs generated in response. In addition to being able to observe the simulated values on the I/O pins of the circuit, it is also possible to probe the internal nodes in the circuit. The simulator makes use of the Waveform Editor, which makes it easy to represent the desired signals as waveforms.

Doing this tutorial, the reader will learn about:

- Test vectors needed to test the designed circuit
- Using the Quartus II Waveform Editor to draw the test vectors
- Functional simulation, which is used to verify the functional correctness of a synthesized circuit
- Timing simulation, which takes into account propagation delays due to logic elements and interconnecting wiring

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.

PREREQUISITES
The reader is expected to have access to a computer that has Quartus II software installed. The detailed examples in the tutorial were obtained using the Quartus II version 5.0, but other versions of the software can also be used.

1 Example Circuit

As an example, we will use the adder/subtractor circuit shown in Figure 1. The circuit can add, subtract, and accumulate n-bit numbers using the 2's complement number representation. The two primary inputs are numbers $A = a_{n-1}a_{n-2} \cdots a_0$ and $B = b_{n-1}b_{n-2} \cdots b_0$, and the primary output is $Z = z_{n-1}z_{n-2} \cdots z_0$. Another input is the *AddSub* control signal which causes $Z = A + B$ to be performed when *AddSub* = 0 and $Z = A - B$ when *AddSub* = 1. A second control input, *Sel*, is used to select the accumulator mode of operation. If *Sel* = 0, the operation $Z = A \pm B$ is performed, but if *Sel* = 1, then B is added to or subtracted from the current value of Z. If the addition or subtraction operations result in arithmetic overflow, an output signal, *Overflow*, is asserted.

To make it easier to deal with asynchronous input signals, they are loaded into flip-flops on a positive edge of the clock. Thus, inputs A and B will be loaded into registers *Areg* and *Breg*, while *Sel* and *AddSub* will be loaded into flip-flops *SelR* and *AddSubR*, respectively. The adder/subtractor circuit places the result into register *Zreg*.

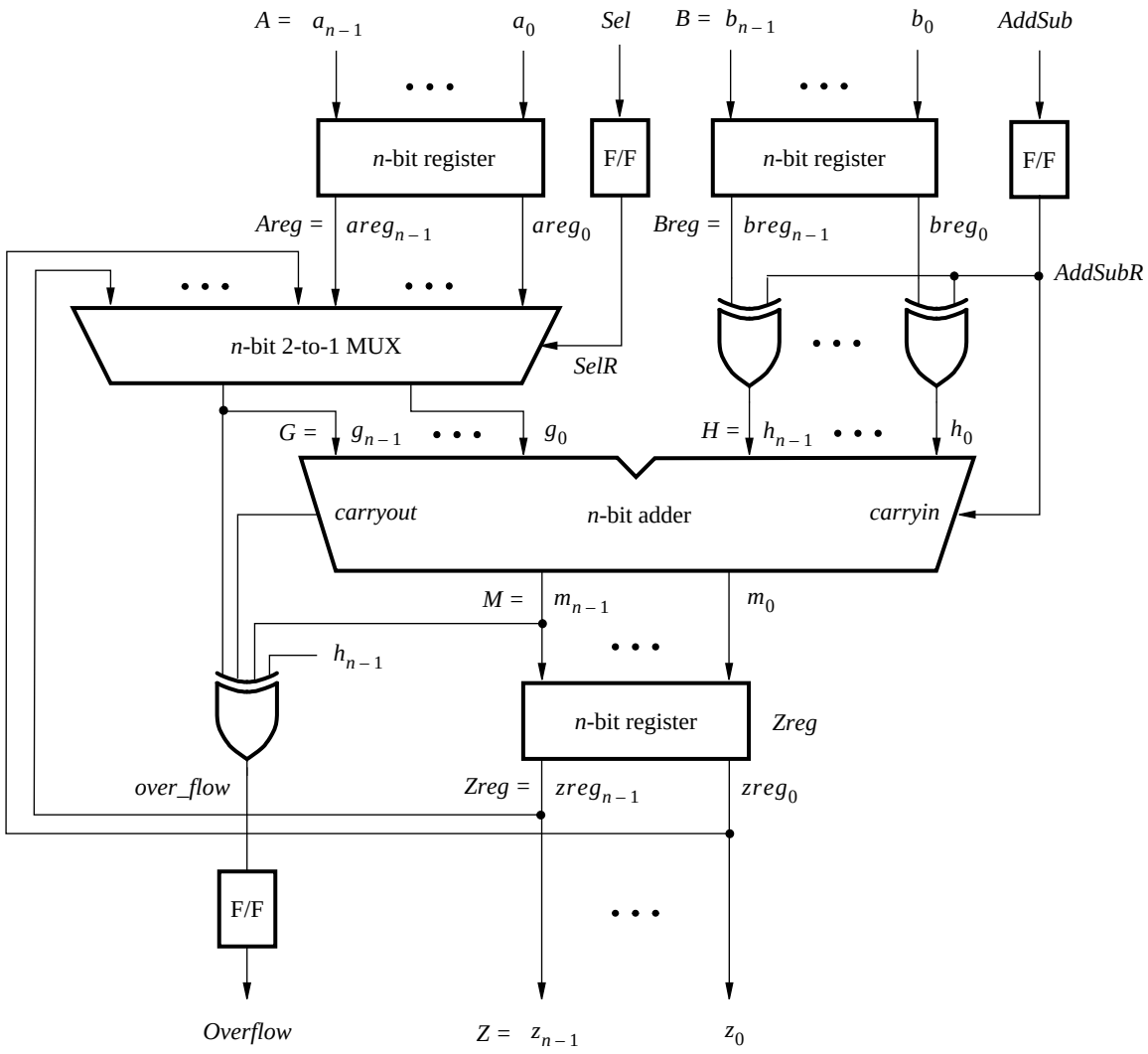

Figure 1. The adder/subtractor circuit.

The required circuit is described by the Verilog code in Figure 2. For our example, we use a 16-bit circuit as specified by $n = 16$. Implement this circuit as follows:

- Create a project *addersubtractor*.
- Include a file *addersubtractor.v*, which corresponds to Figure 2, in the project. For convenience, this file is provided in the directory *DE2_tutorials\design_files*, which is included on the CD-ROM that accompanies the DE2 board and can also be found on Altera's DE2 web pages.
- Choose the Cyclone II EP2C35F672C6 device, which is the FPGA chip on Altera's DE2 board.
- Compile the design.

```
// Top-level module
module addersubtractor (A, B, Clock, Reset, Sel, AddSub, Z, Overflow);
    parameter n = 16;
    input [n−1:0] A, B;
    input Clock, Reset, Sel, AddSub;
    output [n−1:0] Z;
    output Overflow;
    reg SelR, AddSubR, Overflow;
    reg [n−1:0] Areg, Breg, Zreg;
    wire [n−1:0] G, H, M, Z;
    wire carryout, over_flow;

// Define combinational logic circuit
    assign H = Breg ^ {n{AddSubR}};
    mux2to1 multiplexer (Areg, Z, SelR, G);
        defparam multiplexer.k = n;
    adderk nbit_adder (AddSubR, G, H, M, carryout);
        defparam nbit_adder.k = n;
    assign over_flow = carryout ^ G[n−1] ^ H[n−1] ^ M[n−1];
    assign Z = Zreg;

// Define flip-flops and registers
    always @(posedge Reset or posedge Clock)
        if (Reset == 1)
        begin
            Areg <= 0; Breg <= 0; Zreg <= 0;
            SelR <= 0; AddSubR <= 0; Overflow <= 0;
        end
        else
        begin
            Areg <= A; Breg <= B; Zreg <= M;
            SelR <= Sel; AddSubR <= AddSub; Overflow <= over_flow;
        end
endmodule

// k-bit 2-to-1 multiplexer
module mux2to1 (V, W, Selm, F);
    parameter k = 8;
    input [k−1:0] V, W;
    input Selm;
    output [k−1:0] F;
    reg [k−1:0] F;

    always @(V or W or Selm)
        if (Selm == 0) F = V;
        else F = W;
endmodule
```

... continued in Part *b*

Figure 2. Verilog code for the circuit in Figure 1 (Part *a*).

```
// k-bit adder
module   adderk (carryin, X, Y, S, carryout);
    parameter k = 8;
    input  [k−1:0] X, Y;
    input  carryin;
    output  [k−1:0] S;
    output  carryout;
    reg  [k−1:0] S;
    reg  carryout;

    always @(X or Y or carryin)
        {carryout, S} = X + Y + carryin;
endmodule
```

Figure 2. Verilog code for the circuit in Figure 1 (Part *b*).

2 Using the Waveform Editor

Quartus II software includes a simulation tool that can be used to simulate the behavior of a designed circuit. Before the circuit can be simulated, it is necessary to create the desired waveforms, called *test vectors*, to represent the input signals. It is also necessary to specify the outputs, as well as possible internal points in the circuit, which the designer wishes to observe. The simulator applies the test vectors to the model of the implemented circuit and determines the expected response. We will use the Quartus II Waveform Editor to draw the test vectors, as follows:

1. Open the Waveform Editor window by selecting File > New, which gives the window shown in Figure 3. Click on the Other Files tab to reach the window displayed in Figure 4. Choose Vector Waveform File and click OK.

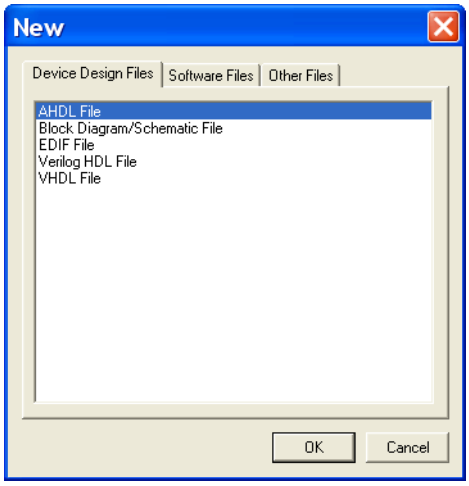

Figure 3. Need to prepare a new file.

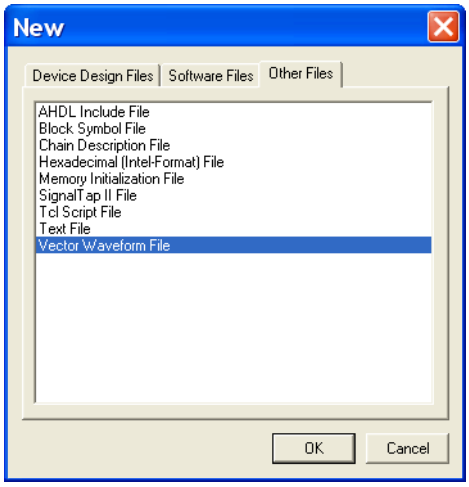

Figure 4. Choose to prepare a test-vector file.

2. The Waveform Editor window is depicted in Figure 5. Save the file under the name *addersubtractor.vwf*; note that this changes the name in the displayed window. In this figure, we have set the desired simulation to run from 0 to 180 ns by selecting Edit $>$ End Time and entering 180 ns in the dialog box that pops up. Selecting View $>$ Fit in Window displays the entire simulation range of 0 to 180 ns in the window, as shown. Resize the window to its maximum size.

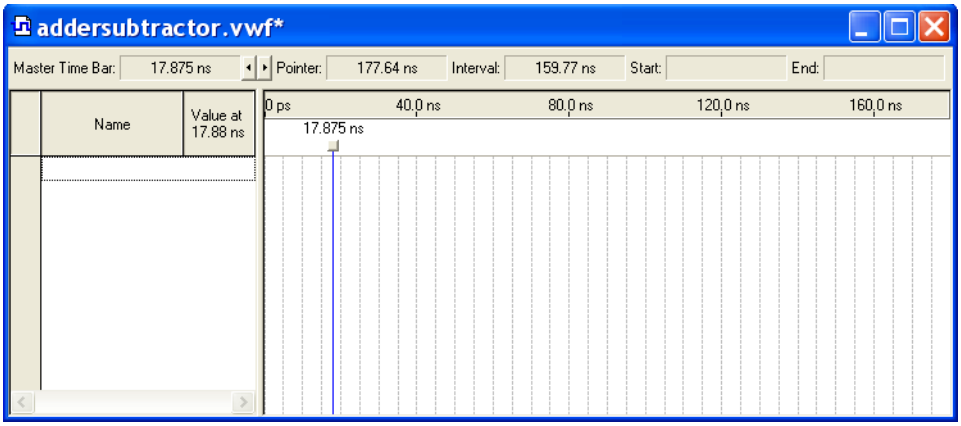

Figure 5. The Waveform Editor window.

3. Next, we want to include the input and output nodes of the circuit to be simulated. Click Edit $>$ Insert Node or Bus to open the window in Figure 6. It is possible to type the full hierarchical name of a signal (pin) into the Name box, but it is easier to click on the button labeled Node Finder to open the window in Figure 7. The Node Finder utility has a filter used to indicate what type nodes are to be found. Since we are interested in input and output pins, set the filter to Pins: all. Click the List button to find the pin names as indicated on the left side of the figure. Observe that the input and output signals A, B, and Z can be selected either as individual nodes (denoted by bracketed subscripts) or as 16-bit vectors, which is a more convenient form.

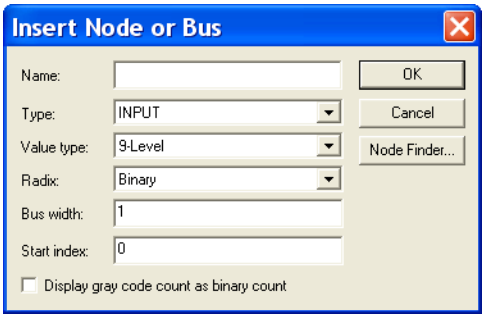

Figure 6. The Insert Node or Bus dialogue.

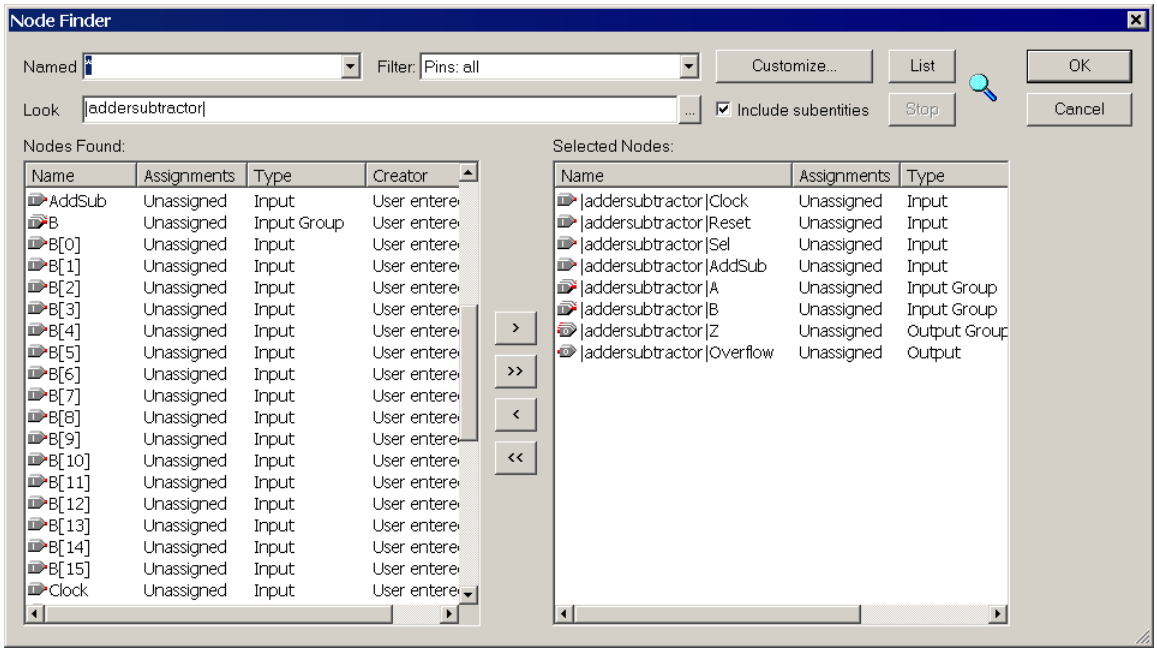

Figure 7. Selecting nodes to insert into the Waveform Editor.

Use the scroll bar inside the Nodes Found box in Figure 7 to find the *Clock* signal. Click on this signal and then click the $>$ sign in the middle of the window to add it to the Selected Nodes box on the right side of the figure. Do the same for *Reset*, *Sel*, and *AddSub*. Then choose vectors A, B and Z, as well as the output *Overflow*, in the same way. (Several nodes can be selected simultaneously in a standard Windows manner.) Click OK to close the Node Finder window, and then click OK in the window of Figure 6. This leaves a fully displayed Waveform Editor window, as shown in Figure 8. If you did not select the nodes in the same order as displayed in Figure 8, it is possible to rearrange them. To move a waveform up or down in the Waveform Editor window, click on the node name (in the Name column) and release the mouse button. The waveform is now highlighted to show the selection. Click again on the waveform and drag it up or down in the Waveform Editor.

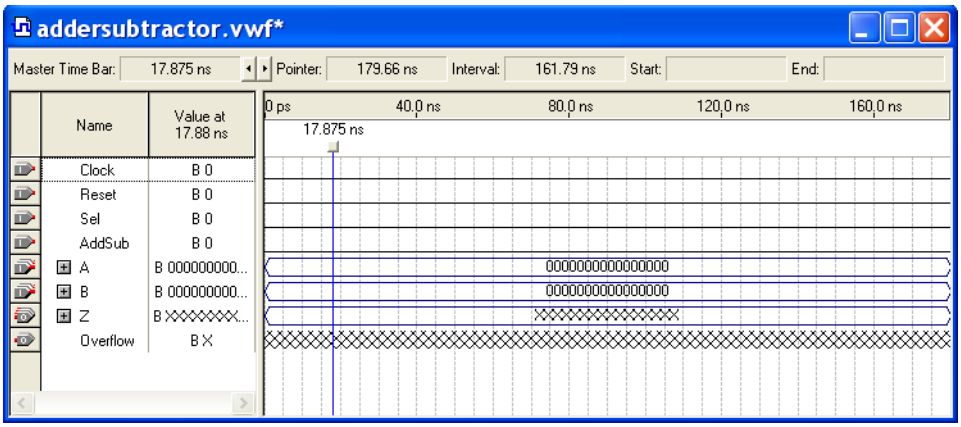

Figure 8. The nodes needed for simulation.

4. We will now specify the logic values to be used for the input signals during simulation. The logic values at the outputs *Z* and *Overflow* will be generated automatically by the simulator. To make it easy to draw the desired waveforms, the Waveform Editor displays (by default) vertical guidelines and provides a drawing feature that snaps on these lines (which can otherwise be invoked by choosing View > Snap to Grid). Observe also a solid vertical line, which can be moved by pointing to its top and dragging it horizontally. This *reference line* is used in analyzing the timing of a circuit, as described later; move it to the *time* = 0 position. The waveforms can be drawn using the Selection Tool, which is activated by selecting the icon in the toolbar, or the Waveform Editing Tool, which is activated by the icon . In the instructions below, we will use the Selection Tool.

   To simulate the behavior of a large circuit, it is necessary to apply a sufficient number of input valuations and observe the expected values of the outputs. The number of possible input valuations may be huge, so in practice we choose a relatively small (but representative) sample of these input valuations. We will choose a very small set of input test vectors, which is not sufficient to simulate the circuit properly but is adequate for tutorial purposes. We will use eight 20-ns time intervals to apply the test vectors as shown in Figure 9. The values of signals *Reset*, *Sel*, *AddSub*, *A* and *B* are applied at the input pins as indicated in the figure. The value of Z at time t_i is a function of the inputs at time t_{i-1}. When *Sel* = 1, the accumulator feedback loop is activated so that the current value of Z (rather than A) is used to compute the new value of Z.

Time	$Reset$	Sel	$AddSub$	A	B	Z
t_0	1	0	0	0	0	0
t_1	0	0	0	54	1850	0
t_2	0	0	1	132	63	1904
t_3	0	0	0	0	0	69
t_4	0	0	1	750	120	0
t_5	0	1	0	0	7000	630
t_6	0	1	0	0	30000	7630
t_7	0	1	0	0	0	37630

Figure 9. The required testing behavior.

The effect of the test vectors in Figure 9 is to perform the following computation:

$$
\begin{aligned}
&t_0 : \text{Reset} \\
&t_1 : Z(t_1) = 0 \\
&t_2 : Z(t_2) = A(t_1) + B(t_1) = 54 + 1850 = 1904 \\
&t_3 : Z(t_3) = A(t_2) - B(t_2) = 132 - 63 = 69 \\
&t_4 : Z(t_4) = A(t_3) + B(t_3) = 0 + 0 = 0 \\
&t_5 : Z(t_5) = A(t_4) - B(t_4) = 750 - 120 = 630 \\
&t_6 : Z(t_6) = Z(t_5) + B(t_5) = 630 + 7000 = 7630 \\
&t_7 : Z(t_7) = Z(t_6) + B(t_6) = 7630 + 30000 = 37630 \text{ (overflow)}
\end{aligned}
$$

Initially, the circuit is reset asynchronously. Then for two clock cycles the output Z is first the sum and then the difference of the values of A and B at that time. This is followed by setting both A and B to zero to clear the contents of register Z. Then, the accumulator feedback path is tested in the next three clock cycles by performing the computation

$$Z = A(t_4) - B(t_4) + B(t_5) + B(t_6)$$

using the values of A and B shown above.

We can generate the desired input waveforms as follows. Click on the waveform name for the *Clock* node. Once a waveform is selected, the editing commands in the Waveform Editor can be used to draw the desired waveforms. Commands are available for defining the clock, or setting the selected signal to 0, 1, unknown (X), high impedance (Z), don't care (DC), and inverting its existing value (INV). Each command can be activated by using the Edit > Value command, or via the toolbar for the Waveform Editor. The Edit menu can also be opened by right-clicking on a waveform name.

With the *Clock* signal highlighted, click on the Overwrite Clock icon in the toolbar. This leads to the pop-up window in Figure 10. Enter the clock period value of 20 ns, make sure that the phase is 0 and the duty cycle is 50 percent, and click OK. The desired clock signal is now displayed in the Waveform window.

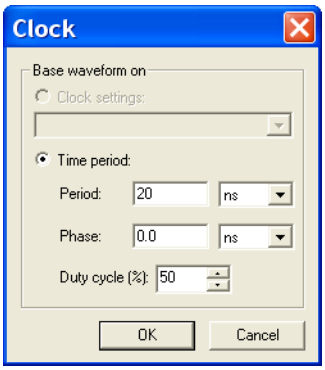

Figure 10. Definition of the clock period, phase and duty cycle.

We will assume, for simplicity of timing, that the input signals change coincident with the negative edges of the clock. To reset the circuit, set *Reset* = 1 in the time interval 0 to 20 ns. Do this by pressing the mouse at the start of the interval and dragging it to its end, which highlights the selected interval, and choosing the logic value 1 in the toolbar. Make *Sel* = 1 from 100 to 160 ns, and *AddSub* = 1 in periods 40 to 60 ns and 80 to 100 ns. This should produce the image in Figure 11.

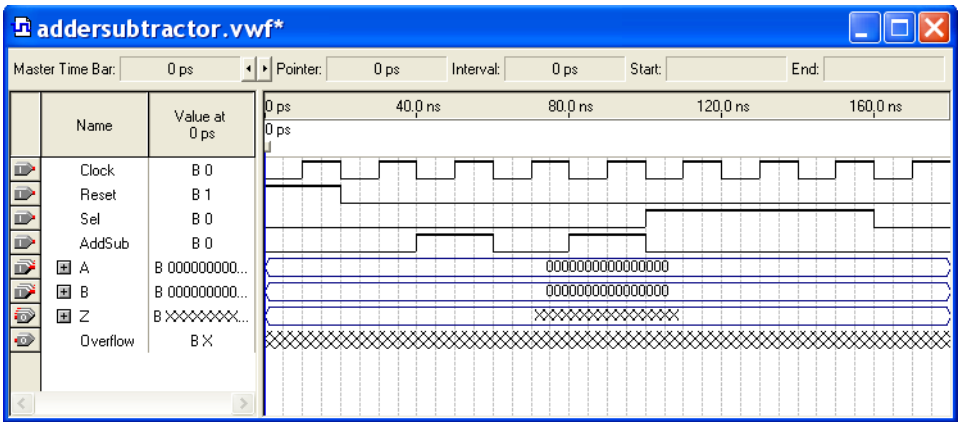

Figure 11. Setting of test values for the control signals.

5. Vectors can be treated as either octal, hexadecimal, signed decimal, or unsigned decimal numbers. The vectors A, B, and Z are initially treated as binary numbers. For our purpose it is convenient to treat them as signed decimal numbers, so right-click on A and select Properties in the pop-up box to get to the window displayed in Figure 12. Choose signed decimal as the radix, make sure that the bus width is 16 bits, and click OK. In the same manner, declare that B and Z should be treated as signed decimal numbers.

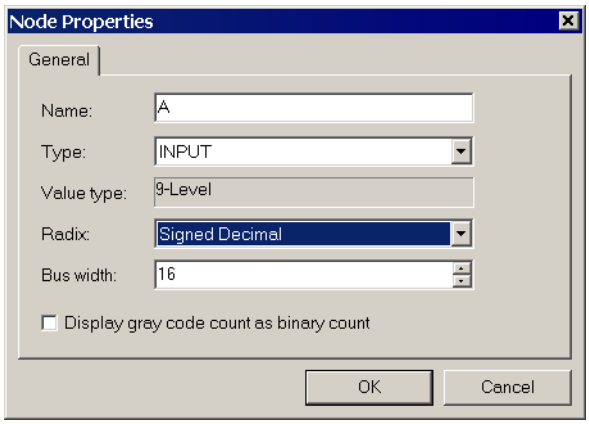

Figure 12. Definition of node properties.

The default value of A is 0. To assign specific values in various intervals proceed as follows. Select (highlight) the interval from 20 to 40 ns and press the Arbitrary Value icon in the toolbar, to bring up the pop-up window in Figure 13. Enter the value 54 and click OK. Similarly, for the subsequent 20-ns intervals set A to the values 132, 0, 750, and then 0 to the end. Set the corresponding values of B to 1850, 63, 0, 120, 7000, 30000, and 0, to generate the waveforms depicted in Figure 14. Observe that the outputs Z and *Overflow* are displayed as having unknown values at this time, which is indicated by a hashed pattern; their values will be determined during simulation. Save the file.

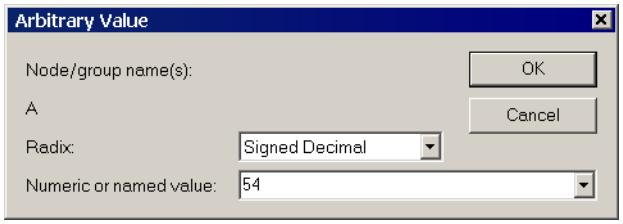

Figure 13. Specifying a value for a multibit signal.

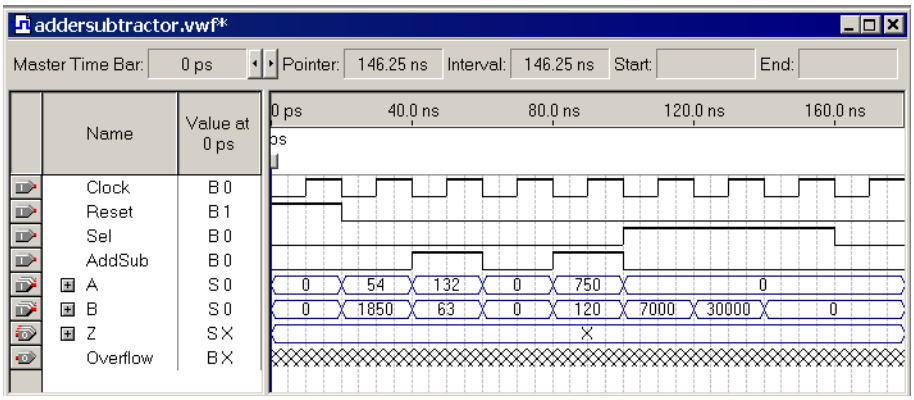

Figure 14. The specified input test vectors.

Another convenient mechanism for changing the input waveforms is provided by the Waveform Editing tool, which is activated by the icon . When the mouse is dragged over some time interval in which the waveform is 0 (1), the waveform will be changed to 1 (0). Experiment with this feature on signal *AddSub*.

3 Performing the Simulation

A designed circuit can be simulated in two ways. The simplest way is to assume that logic elements and interconnection wires are perfect, thus causing no delay in propagation of signals through the circuit. This is called *functional simulation*. A more complex alternative is to take all propagation delays into account, which leads to *timing simulation*. Typically, functional simulation is used to verify the functional correctness of a circuit as it is being designed. This takes much less time, because the simulation can be performed simply by using the logic expressions that define the circuit.

3.1 Functional Simulation

To perform the functional simulation, select Assignments > Settings to open the Settings window shown in Figure 15. On the left side of this window click on Simulator to display the window in Figure 16, choose Functional as the simulation mode, and click OK. The Quartus II simulator takes the inputs and generates the outputs defined in the *addersubtractor.vwf* file. Before running the functional simulation it is necessary to create the required netlist, which is done by selecting Processing > Generate Functional Simulation Netlist.

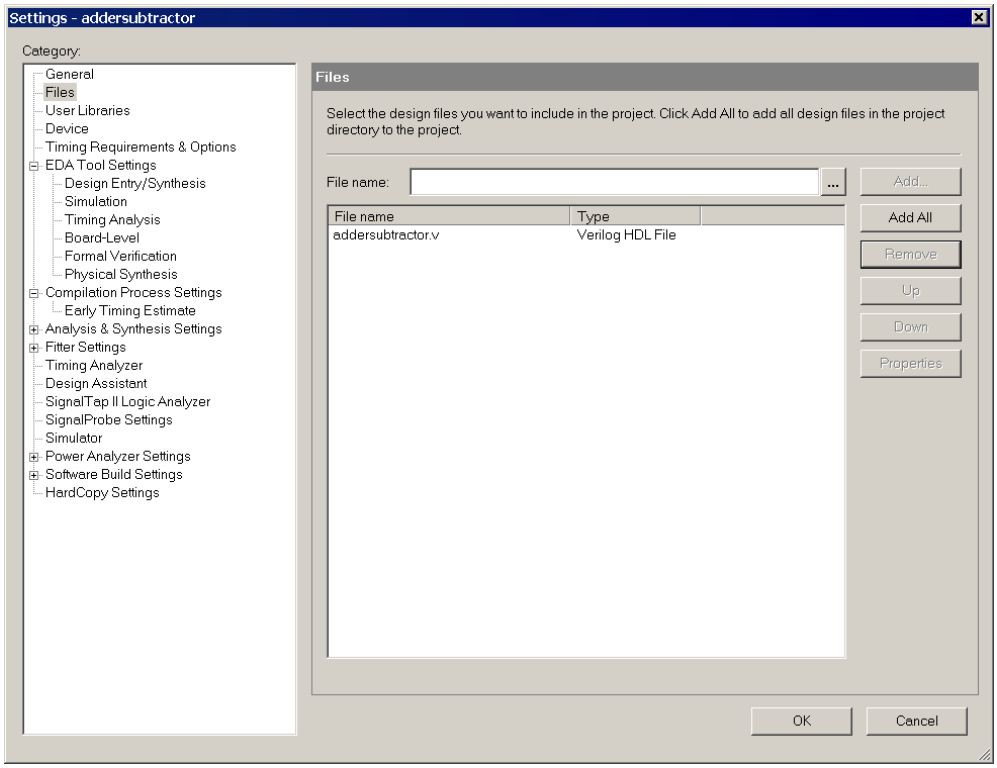

Figure 15. Settings window.

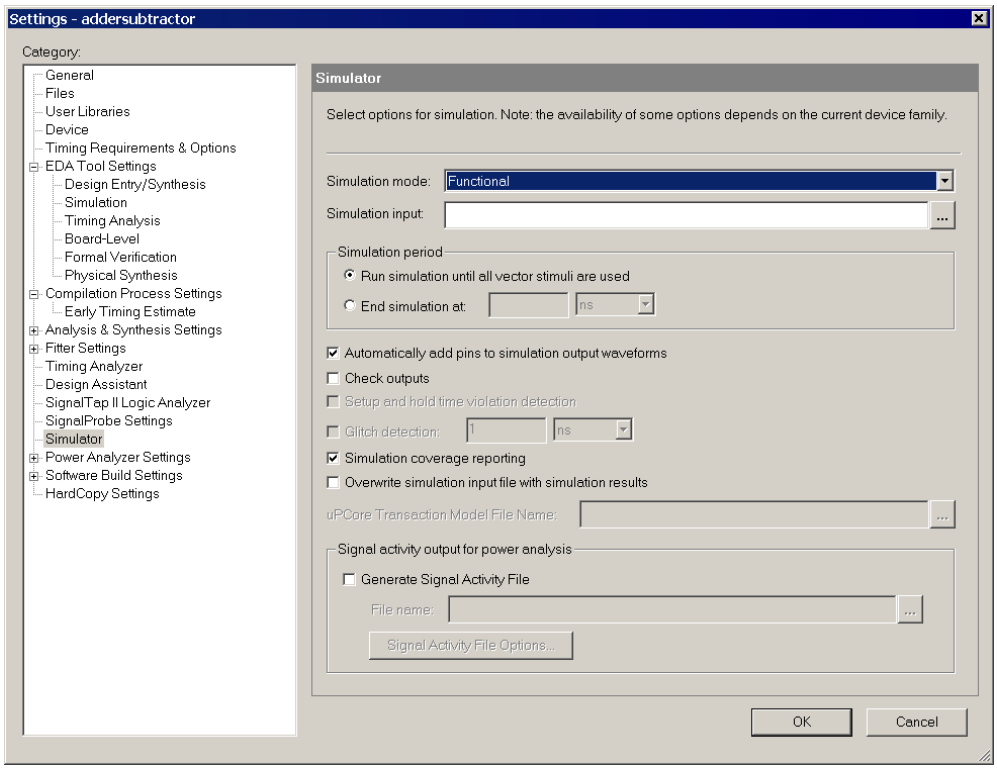

Figure 16. Specifying the simulation mode.

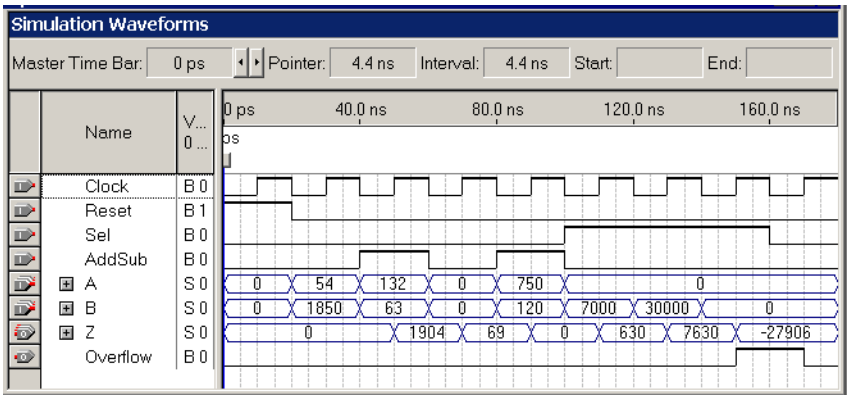

Figure 17. The result of functional simulation.

A simulation run is started by Processing > Start Simulation, or by using the icon . At the end of the simulation, Quartus II software indicates its successful completion and displays a Simulation Report illustrated in Figure 17. As seen in the figure, the Simulator creates waveforms for the outputs *Z* and *Overflow*. As expected, the values of *Z* indicate the correct sum or difference of the applied inputs one clock cycle later because of the registers in the circuit. Note that the last value of *Z* is incorrect because the expected sum of 37630 is too big to be represented as a signed number in 16 bits, which is indicated by the *Overflow* signal being set to 1.

In this simulation, we considered only the input and output signals, which appear on the pins of the FPGA chip. It is also possible to look at the behavior of internal signals. For example, let us consider the registered signals *SelR*, *AddSubR*, *Areg*, *Breg*, and *Zreg*. Open the *addersubtractor.vwf* file and activate the Node Finder window, as done for Figure 6. The filter in Figure 6 specified Pins: all. There are several other choices. To find the registered signals, set the filter to Registers: post-fitting and press List. Figure 18 shows the result. Select the signals *SelR*, *AddSubR*, *Areg*, *Breg*, and *Zreg* for inclusion in the *addersubtractor.vwf* file, and specify that *Areg*, *Breg*, and *Zreg* have to be displayed as signed decimal numbers, thus obtaining the display in Figure 19. Save the file and simulate the circuit using these waveforms, which should produce the result shown in Figure 20.

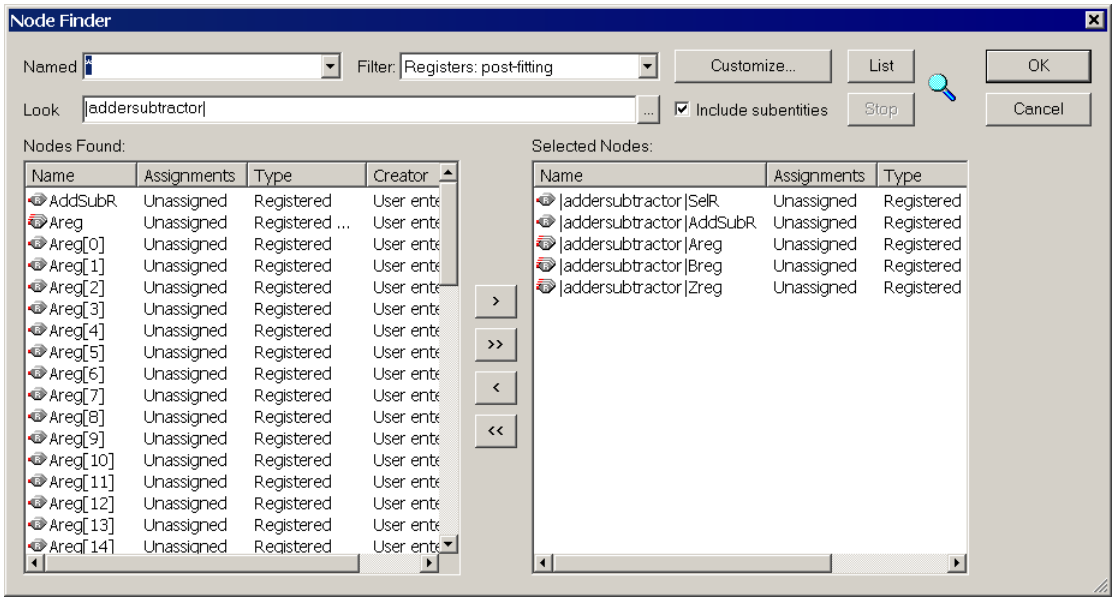

Figure 18. Finding the registered signals.

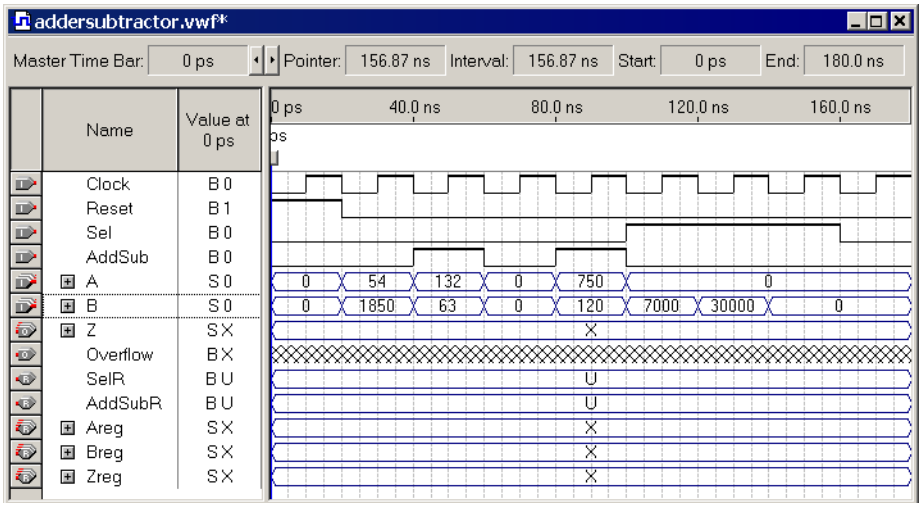

Figure 19. Inclusion of registered signals in the test.

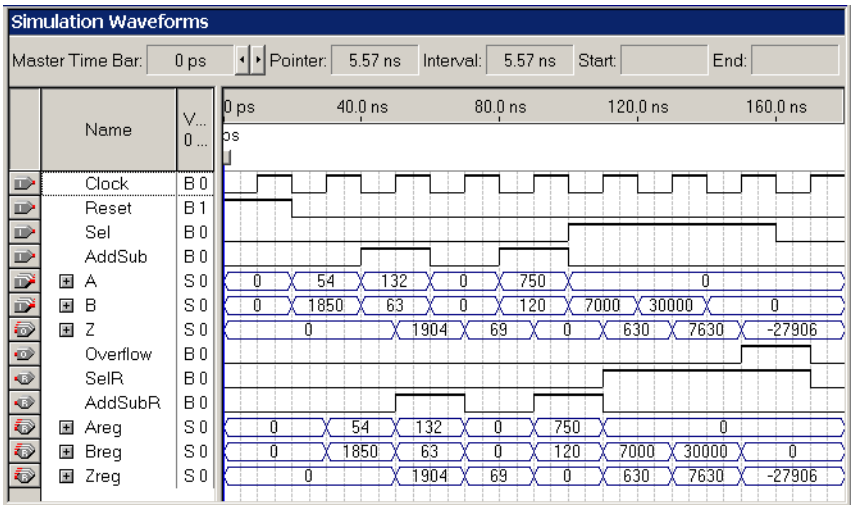

Figure 20. The result of new simulation.

3.2 Timing Simulation

Having ascertained that the designed circuit is functionally correct, we should now perform the timing simulation to see how well it performs in terms of speed. Select Assignments > Settings > Simulator to get to the window in Figure 16, choose Timing as the simulation mode, and click OK. Run the simulator, which should produce the waveforms in Figure 21. Observe that there are delays in loading the various registers as well as longer delays in producing valid signals on the output pins.

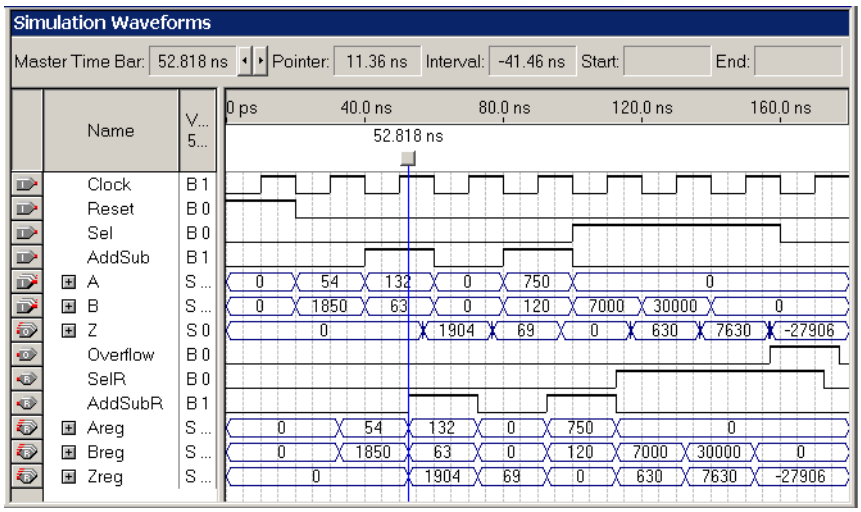

Figure 21. The result of timing simulation.

As an aid in seeing the actual values of the delays, we can use the reference line. Point to the small square handle at the top of the reference line and drag it to the rising edge of the first *AddSubR* pulse, at which time the registers are also loaded, as indicated in the figure. (To make it possible to move the reference line to any point in the waveform display, you may have to turn off the feature View > Snap on Grid.) This operation places the reference line at about the 52.8 ns point, which indicates that it takes 2.8 ns to load the registers after the rising edge of the clock (which occurs at 50 ns). The output Z attains its correct value some time after this value has been loaded into *Zreg*. To determine the propagation delay to the output pins, drag the reference line to the point where Z becomes valid. This can be done more accurately by enlarging the displayed simulation waveforms by using the Zoom Tool. Left-click on the display to enlarge it and right-click to reduce it. Enlarge the display so that it looks like the image in Figure 22. (After enlarging the image, click on the Selection Tool icon . Position the reference line where Z changes to 1904, which occurs at about 57.2 ns. The display indicates that the propagation delay from register $Zreg$ to the output pins Z is $57.2 - 52.8 = 4.4$ ns. It is useful to note that even before we performed this simulation, the Quartus II timing analyzer evaluated various delays in the implemented circuit and reported them in the Compilation Report. From the Compilation Report we can see that the worst case *tco* (Clock to Output Delay) for the Z output (pin z_3) was estimated as 7.18 ns; this delay can be found by zooming into the simulation results at the point where Z changes to the value 7630.

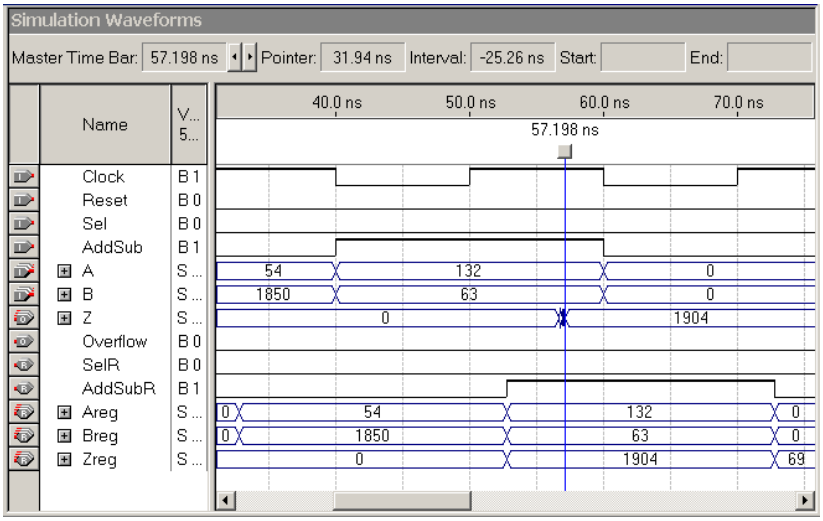

Figure 22. An enlarged image of the simulated waveforms.

In this discussion, we have used the numbers obtained during our simulation run. The user is likely to obtain somewhat different numbers, depending on the version of Quartus II software that is used.