Simulating the Logical Sub-Block

The ModelSim Simulator

Having created a design unit which has a clearly defined behavior, we need to verify that we have correctly specified that behavior in the VHDL code. We will do this using **FPGA Advantage's** simulation tool, **ModelSim**.

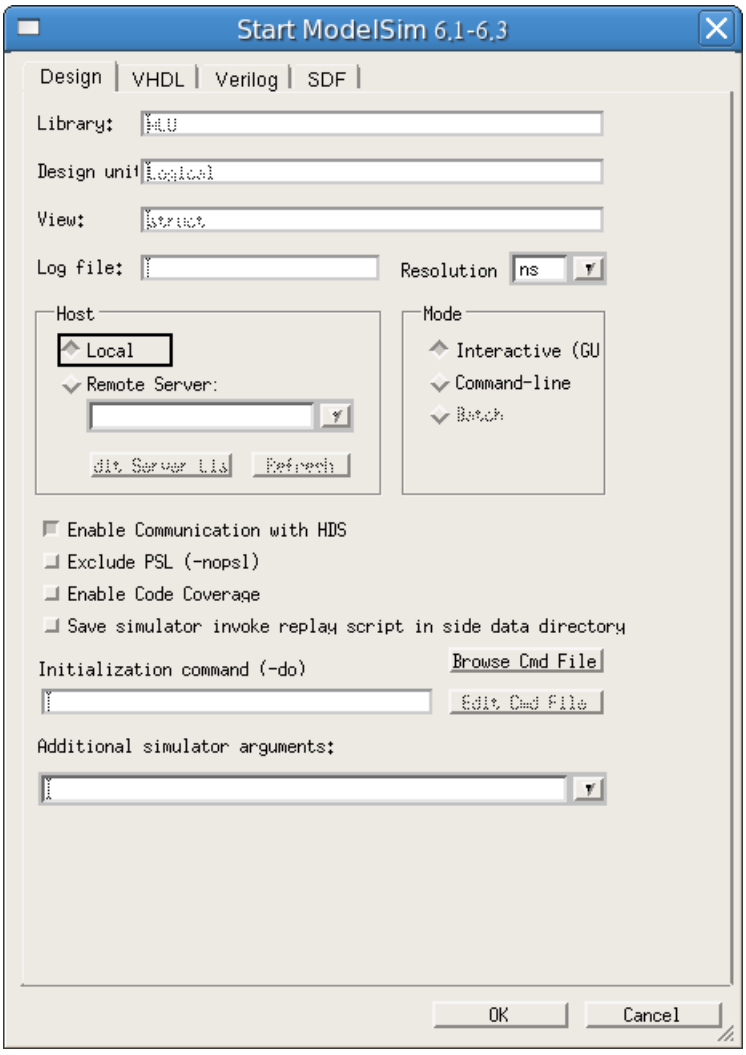
ModelSim is a very powerful and versatile HDL simulation tool which has been tightly integrated with **FPGA Advantage**. As a result, there are several methods by which we could go about verifying our design ranging from loading the bare design unit into the simulator and watching the outputs as we force the input signals into different states to creating a VHDL *Test Bench* which will automate this process for us. For our introduction to **ModelSim**, we will be using the first approach to get a feel for some of what the simulator can do.

The **ModelSim** simulator cannot directly load and simulate your design unit source files in **FPGA Advantage**. In order to prepare a design for simulation, two steps must be taken: generation and compilation.

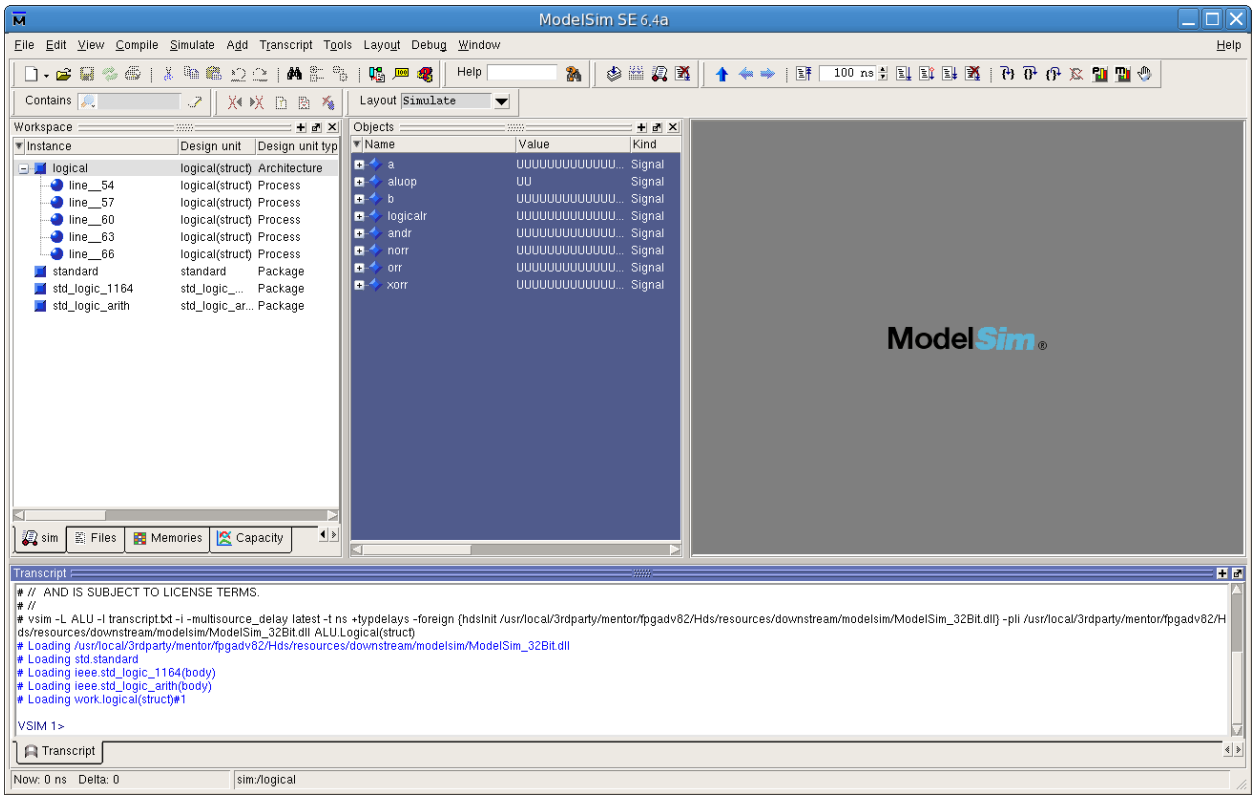
Once the VHDL for the *Logical* sub-block has been generated, it needs to be compiled into a **ModelSim** simulation file. This can be done in one step by highlighting the Logical block in the Design Manager and using the ModelSim design flow button:  from the following available buttons:

If there is a problem, check the design unit block diagram for errors and/or call the instructors.

Once the design has been compiled, the simulator should start automatically. This will bring up the options window seen in the figure below. Leave all of the default options as they are and click the **OK** button.



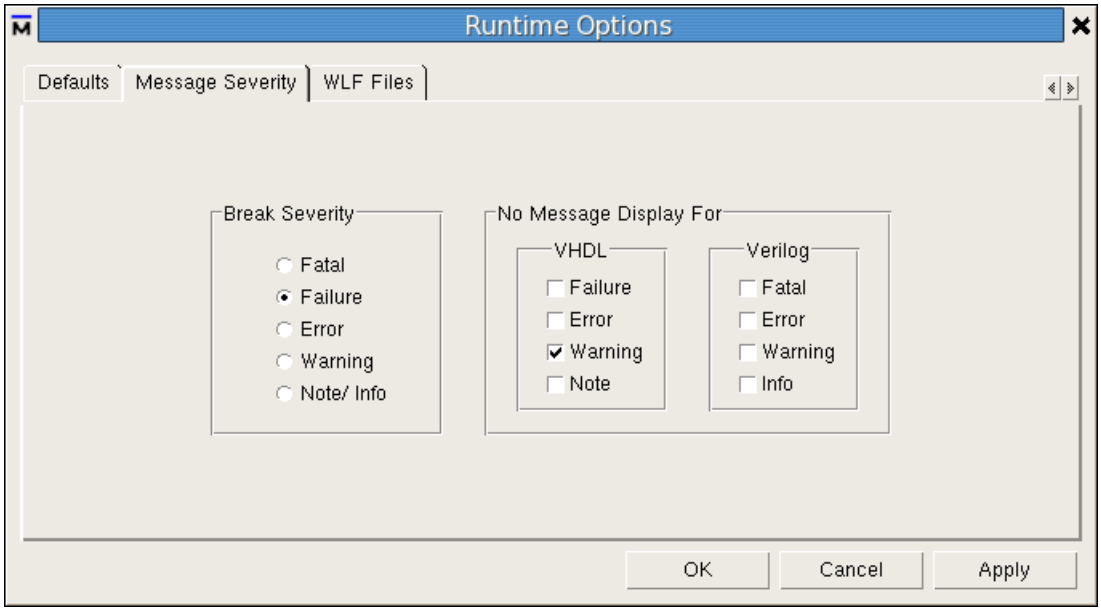
After a few seconds, the **ModelSim** main window pictured in the figure below will appear.



You will also notice that the bottom of the **FPGA Advantage** design window with the *Logical* sub-block diagram has a new simulation toolbar pictured below:

In addition to the **ModelSim** main window, where all text commands to the simulator are entered, there are several other graphical interface windows available. In this section of the tutorial we will be using the *Objects* and *Wave* windows.

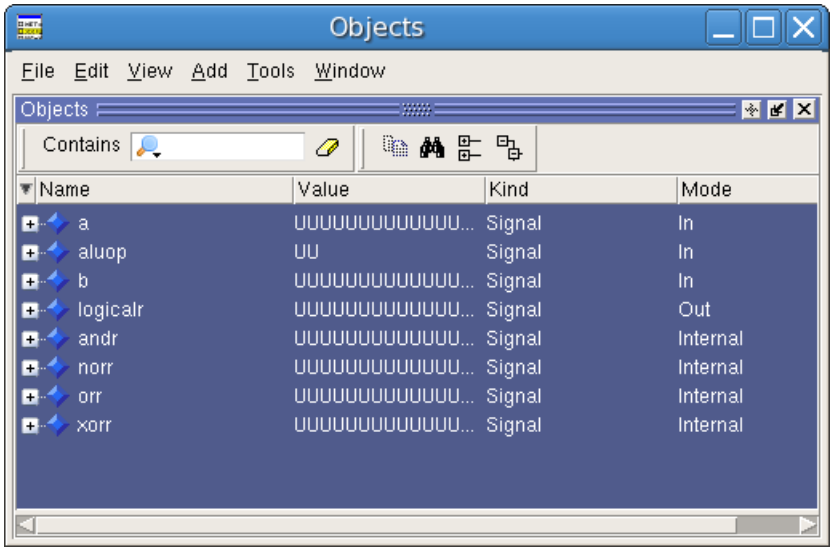
When we first attempt to "power up" the Logical unit in **ModelSim**, it must first be placed into a starting state. In a real circuit this state will be completely random; **ModelSim** will treat everything as unknown. To prevent **ModelSim** from throwing warnings about every unknown output, go to the **Simulate** menu in **ModelSim**, select **Runtime Options**, click the **Message Severity** tab and place a check next to **Warning** in the **No Messesage Display For***:* **VHDL** list.



To display the **Signals** window, go to the **View** menu in the main **ModelSim** window and select **Objects** from the list (note: this window may already be displayed by default).

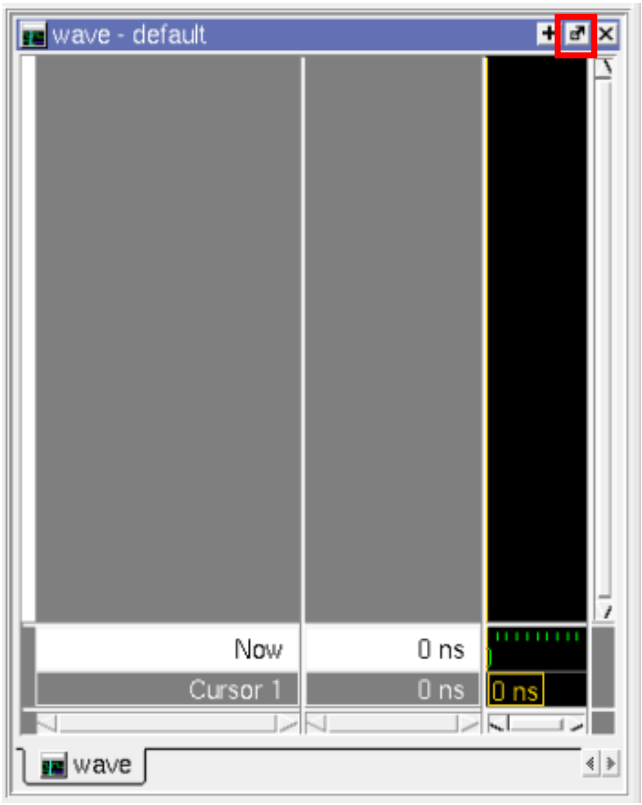
You will find that most current digital simulators consist of graphical shells which issue text commands to the actual simulator. This is a remnant of the fact that until recently most serious digital simulation was done on high-end UNIX workstations where a text interface is the norm.

Looking at the **Objects** window, you can see that all of the ports and internal signals for the *Logical* sub-block are present. Since the simulator has not yet run forward in time, they are all currently in an undefined state.

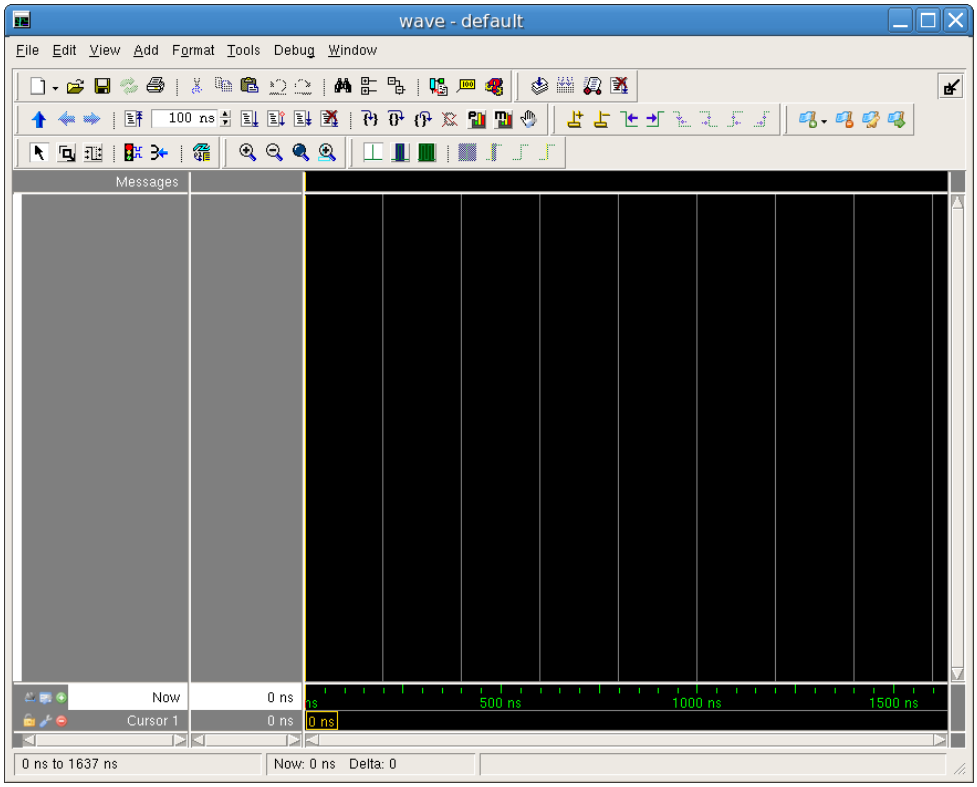


Next open the **Wave** window from the **View > Wave** menu. The window in figure below will

appear. Click the "undock" button  to undock the wave window from the main Modelsim window.

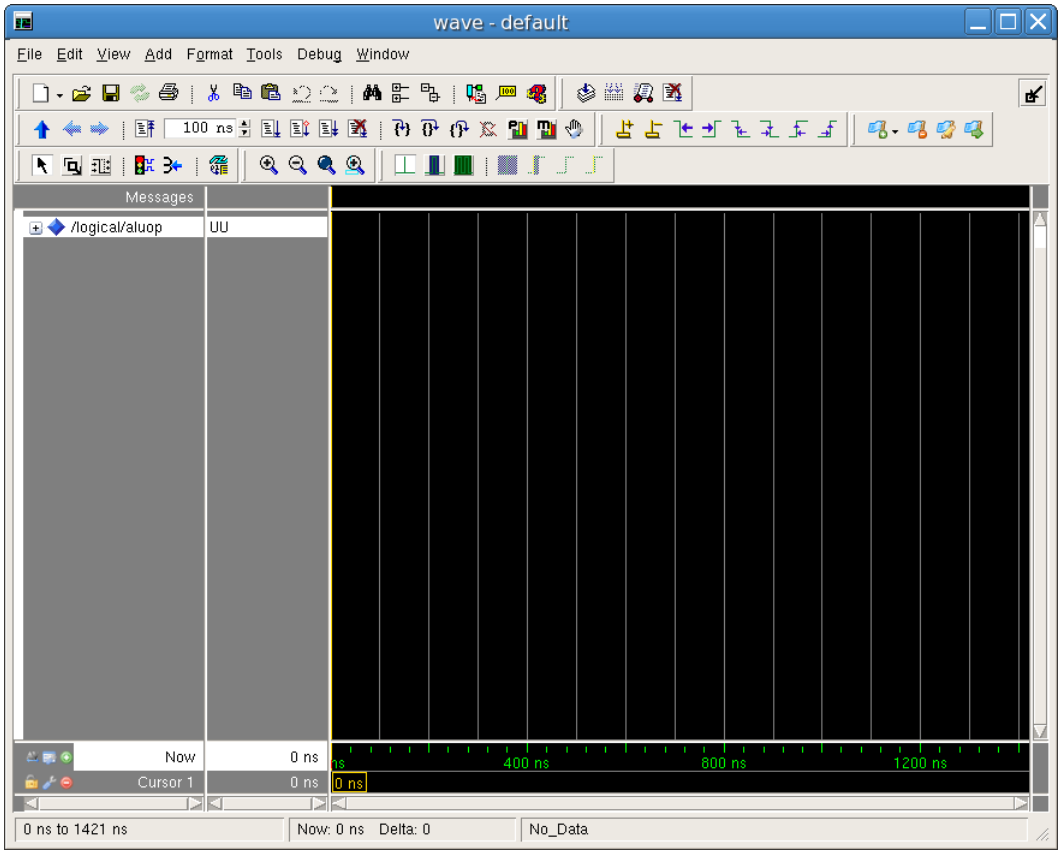


This is the waveform viewer for **ModelSim**. By default, it appears without any signals in the viewing area, they must be added manually.

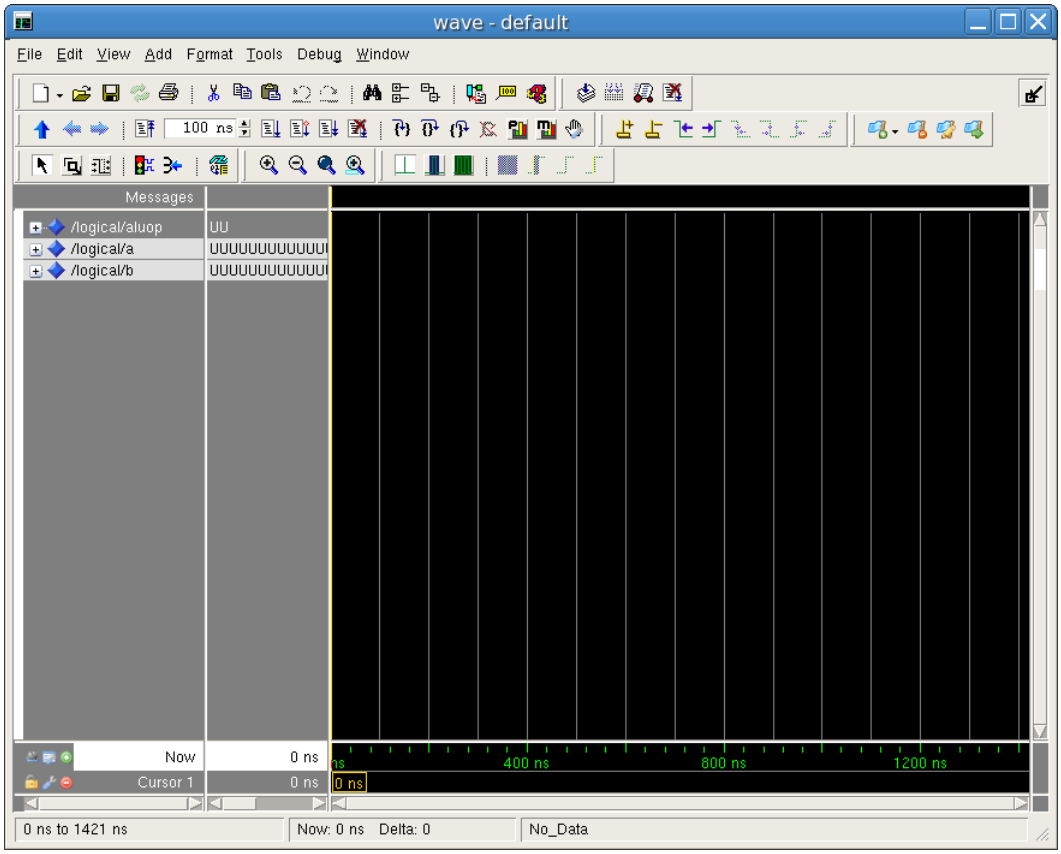


Now that we have the Objects and Wave windows open, we will start adding signals to the waveform viewer. There are several different ways to do this. We will do it by dragging them from the Objects window and dropping them into the left hand frame of the Wave window.

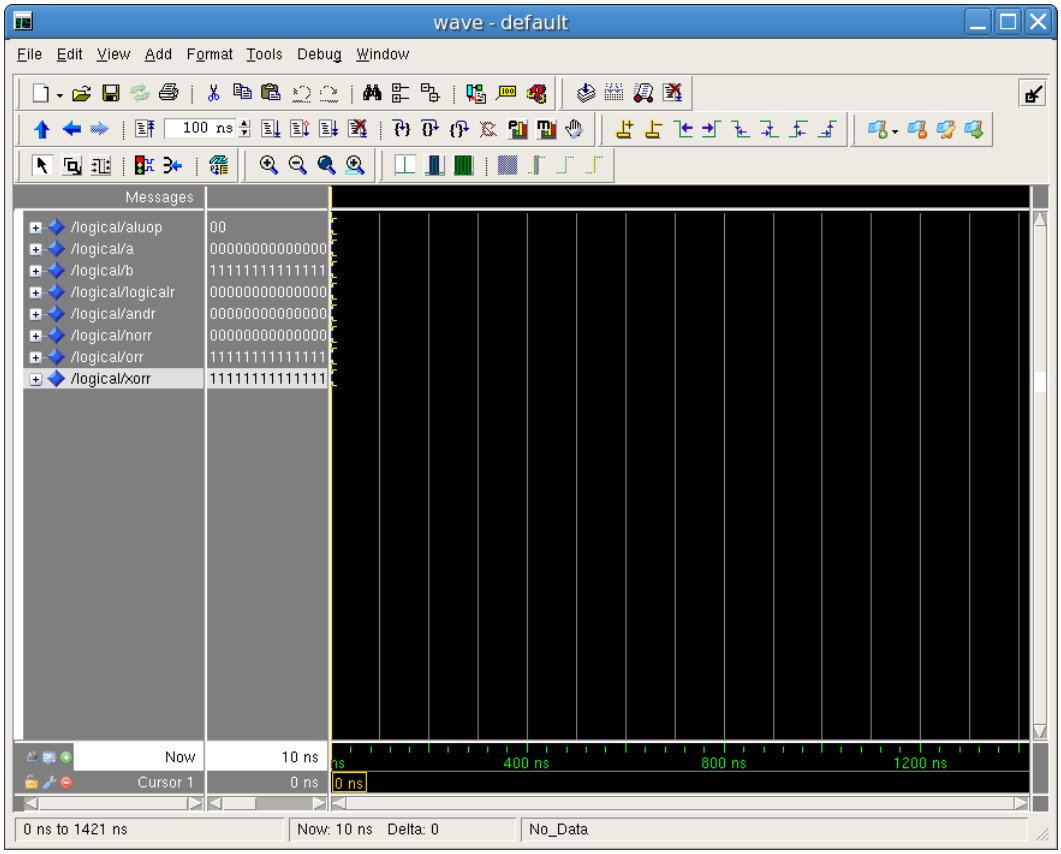
Begin by selecting the *ALUOp* (aluop) signal in the Objects window by left-clicking over it. Once you have selected it, left-click again and this time hold the mouse button down. While the mouse button remains depressed, drag the pointer over to the Wave window and release. Once the *ALUOp* signal has been added to the Wave window, you will need to resize the dividers in the Wave window to allow for enough room to display the name of the signal and the current value. The Wave window should now look like figure below:



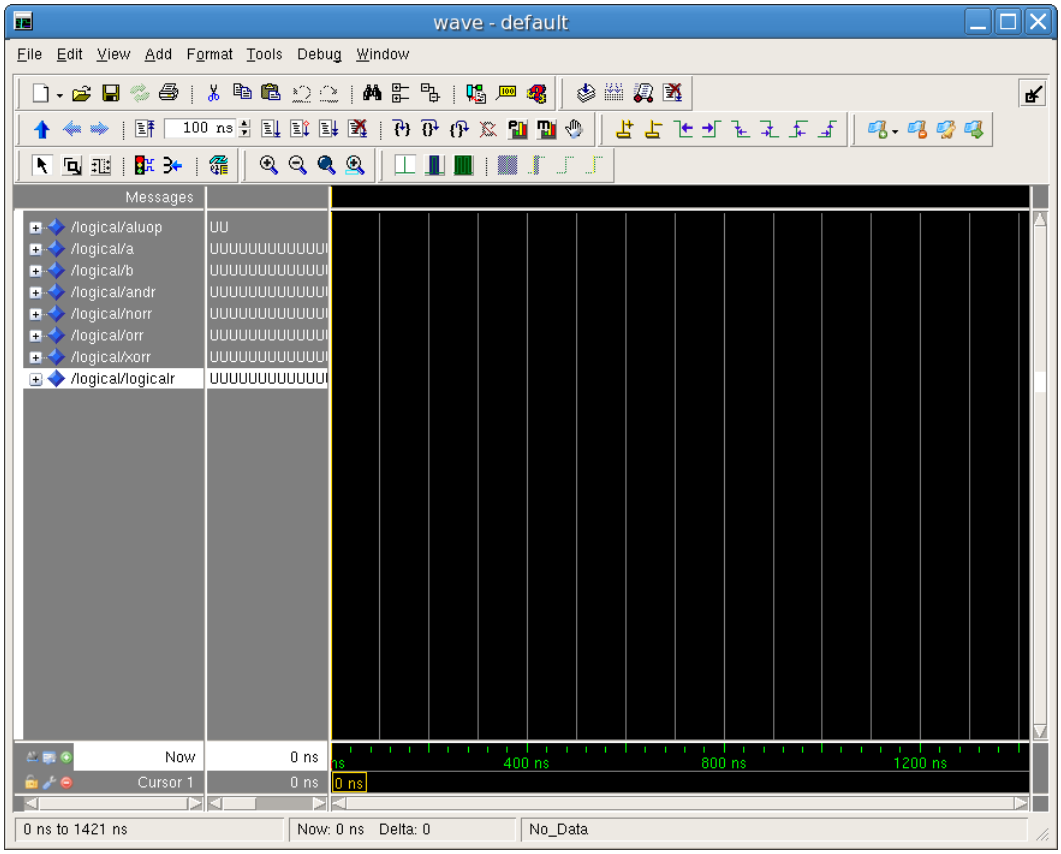
Now drag the *A* and *B* signals respectively into the Wave window so that it looks like the figure below:



We can also drag multiple signals at once. Left-click on the *LogicalR* (logicalr) signal in the Objects window to select it. Then, hold the **Shift** key down to select the bottom four signals. Now, let go of the **Shift** key and drag this group of signals onto the Wave window, which should now resemble the figure below: (the **Ctrl** key can be used to select multiple signals individually, while **Shift** selects all signals from the first signal selected to the last signal selected).

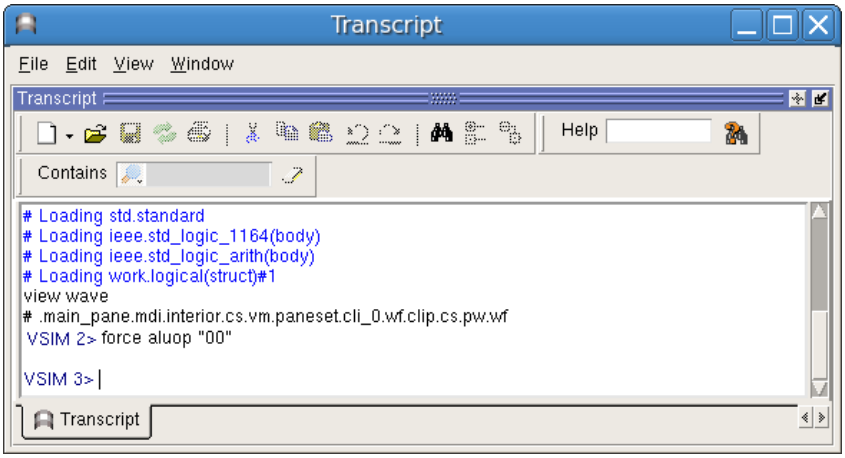


Finally, within the Wave window we can rearrange the display order to our liking. It is often convenient to place the output or outputs of the design unit at the bottom of the waveform display. Currently, however, the *LogicalR* signal is sitting in the middle of the display. Select this signal in the left-hand frame of the Wave window by left-clicking. Now left-click and hold and drag it down to the bottom of the list. Your waveform display should now look like the figure below:



Now that we have set up our display, let us look at how we can make this simulation do something. The most direct way to do this is to stimulate the input signals by forcing them to particular values. Remember, forcing a signal to a value does not actually take effect until you advance the simulator time.

The command in **ModelSim** to stimulate a signal is called *force*. For our first timestep, we wish to set the ALUOp to "00", so we type *force aluop "00"* at the **VSIM** prompt and hit return in the **Transcript** window. The Transcript window will now look like the figure below. Constant strings of bits can simply be represented by a sequence of ones and zeros.

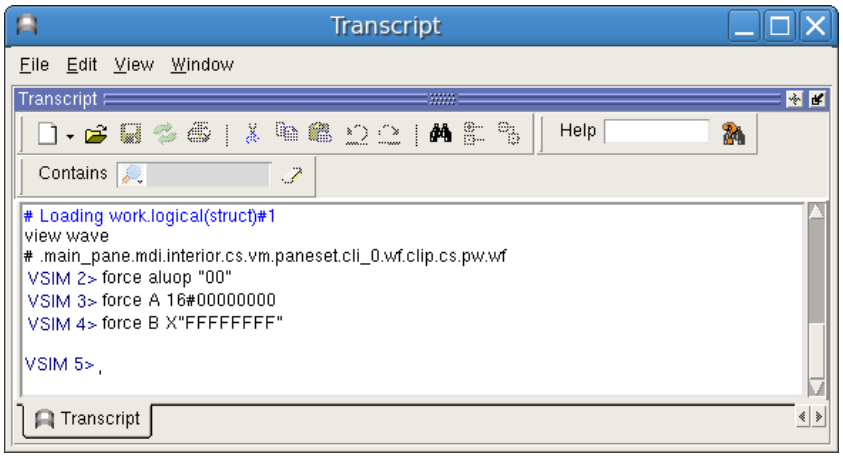


For the inputs, *A* and *B*, we also wish to assign values. However, for signals with so many bits, it is tedious to type out the constant assignment values bit by bit. Instead, it is possible to represent constant values in other bases like hexadecimal or decimal which will be converted to actual bit strings by the simulator. There are two equivalent syntaxes to do this. One is to type a decimal number representing the base, followed by the # sign, followed by the value in the appropriate base. The other is to use the VHDL standard method of the base identifier (hexadecimal is X) followed by the value in double-quotes.

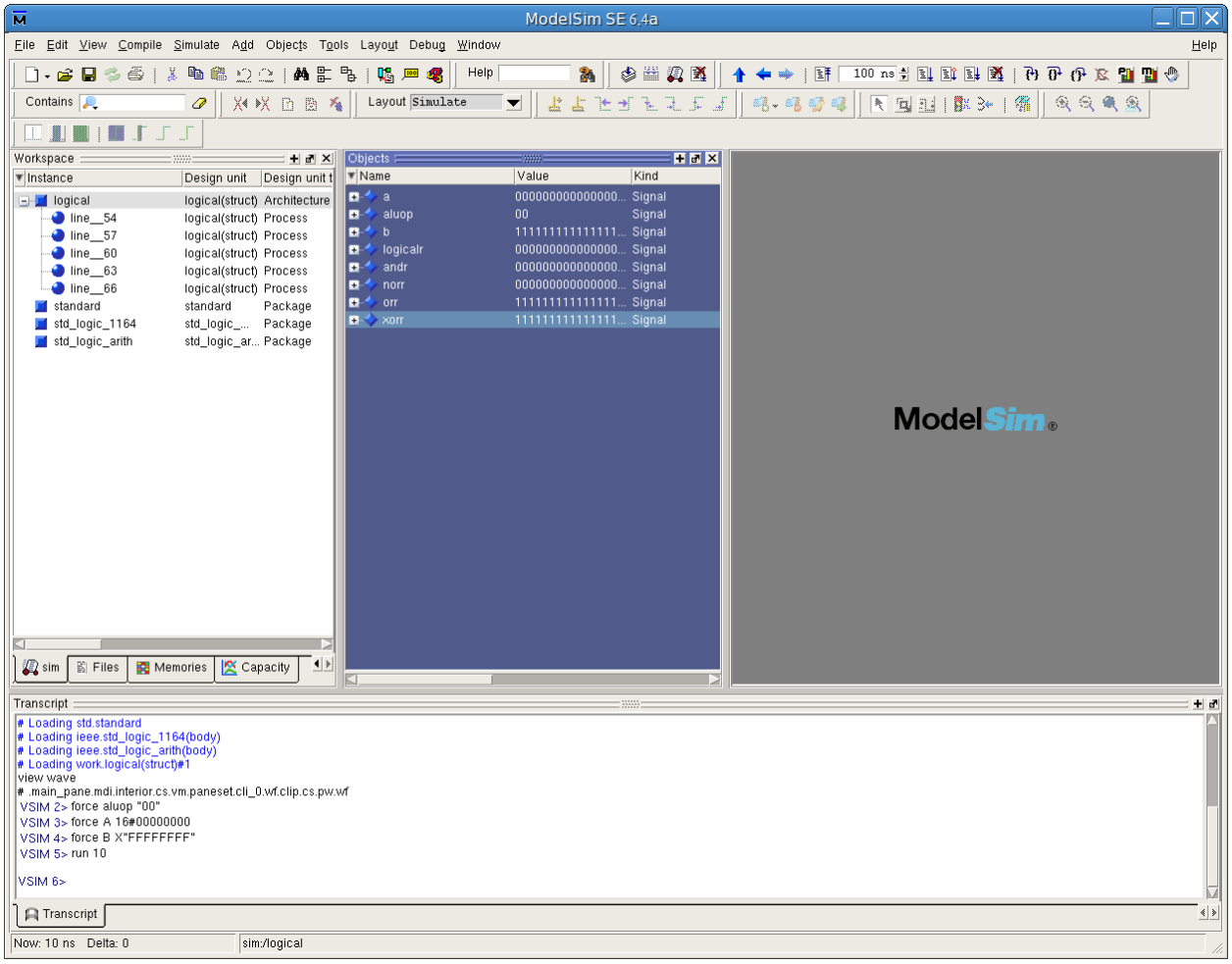
We will be assigning *A* to a value of all zeros and *B* to a value of all ones. In hexadecimal assignment, this would be done by typing:

```
force A 16#00000000
force B X"FFFFFFFF"
```

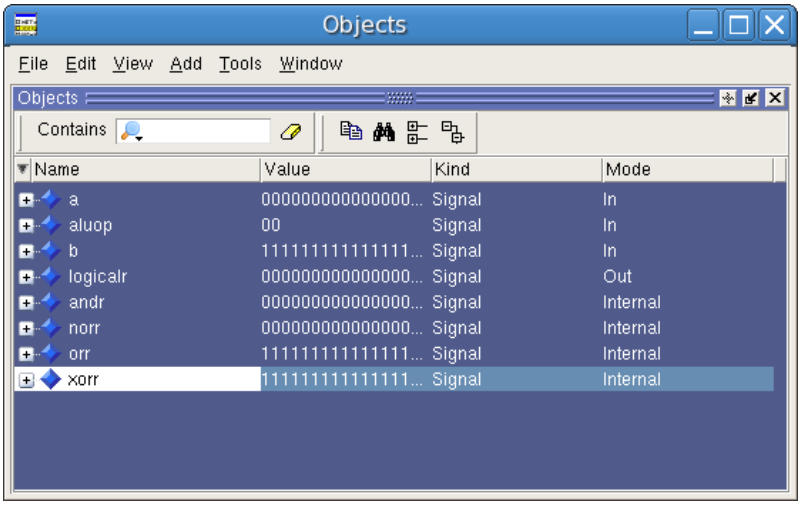
into the main window as seen in the figure below:

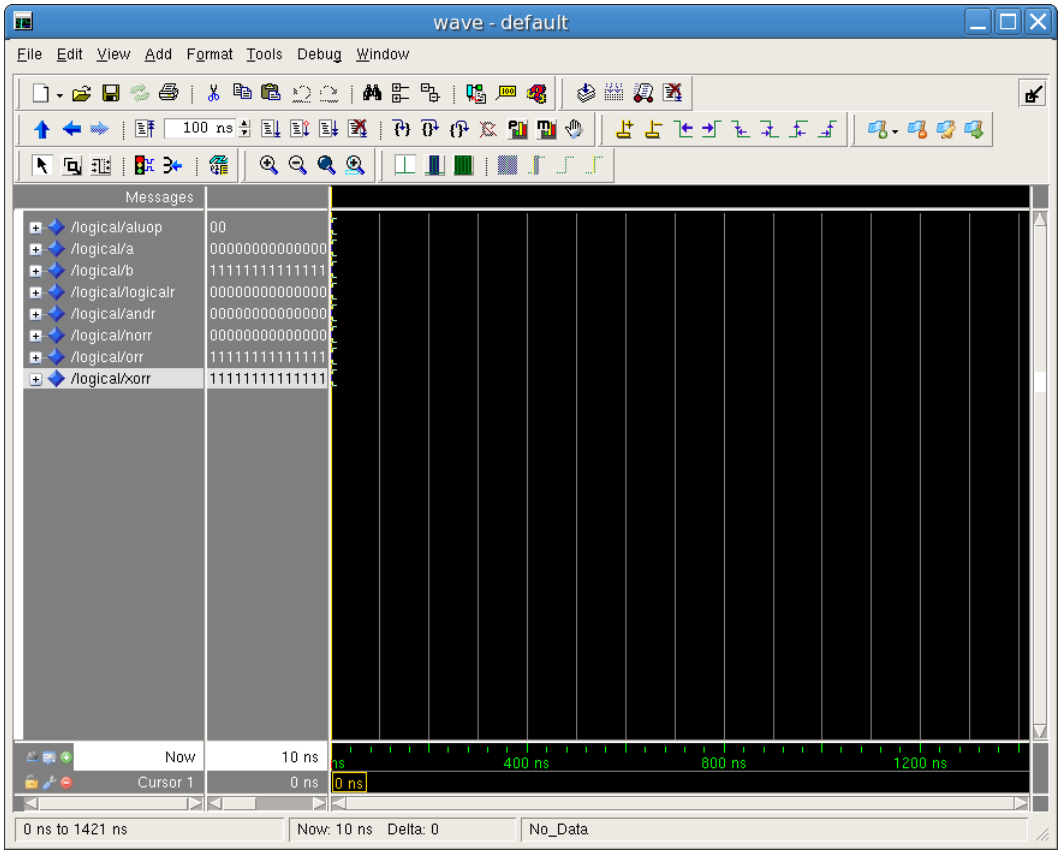


Finally, we need to step the simulator forward in time. Since we do not have any timing delay information in our design, leading to all transitions occurring instantaneously, the amount of time we step by means little, so for starters we will move forward 10ns. We run the simulator for a specified amount of time by typing *run xxx* where xxx is a time in nanoseconds. So type *run 10* at the prompt as seen in the figure below:



Now that we have moved forward with stimulus on the inputs, there should be well defined values at the outputs and the waveform window should hold history of the specified signals for 0 to 10 nanoseconds. The **Objects** and **Wave** windows should look like the following figures respectively.





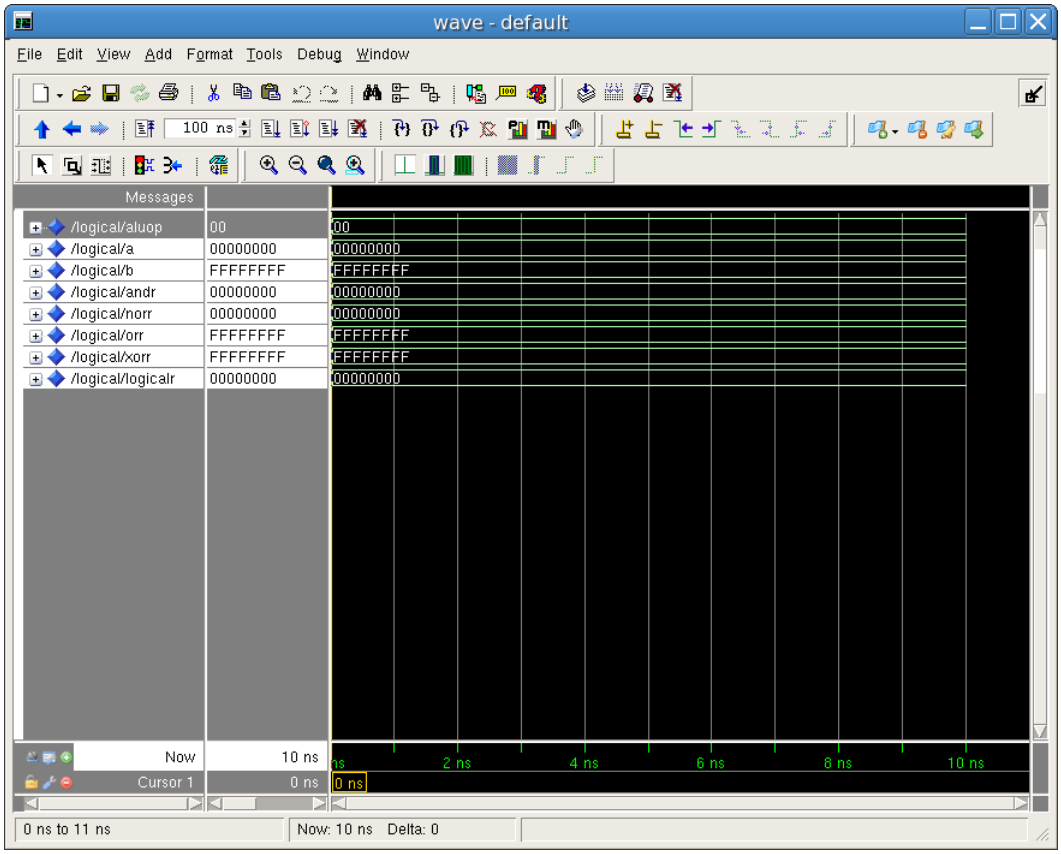
Unfortunately, in the default configuration, the waveforms in the Wave window are impossible to read. This can be fixed in several ways. First resize the window, if you haven't already done so, so that it takes up most of the width of the screen.

Next, highlight all seven of the 32 bit signals and right click on any of highlighted signals. Click **Radix** and then **Hexadecimal**.

Now position the pointer over the divider between the two viewing frames. Left-click over it and drag to the right until all of the signal values in the left frame are visible. The left frame contains the name of the signals and the value of each signal at the point in the waveform window that the cursor is located.

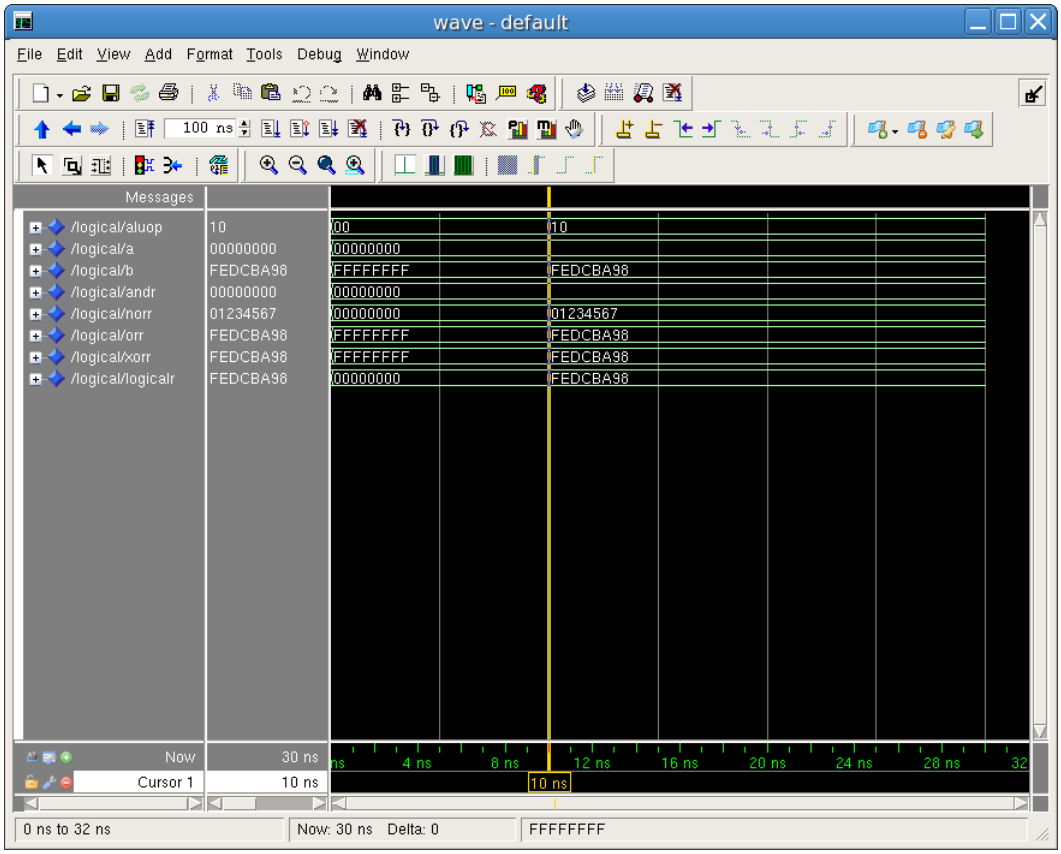
Finally, click the button on the toolbar to fully zoom in on the displayed waveform.

The Wave window should now look like the figure below:

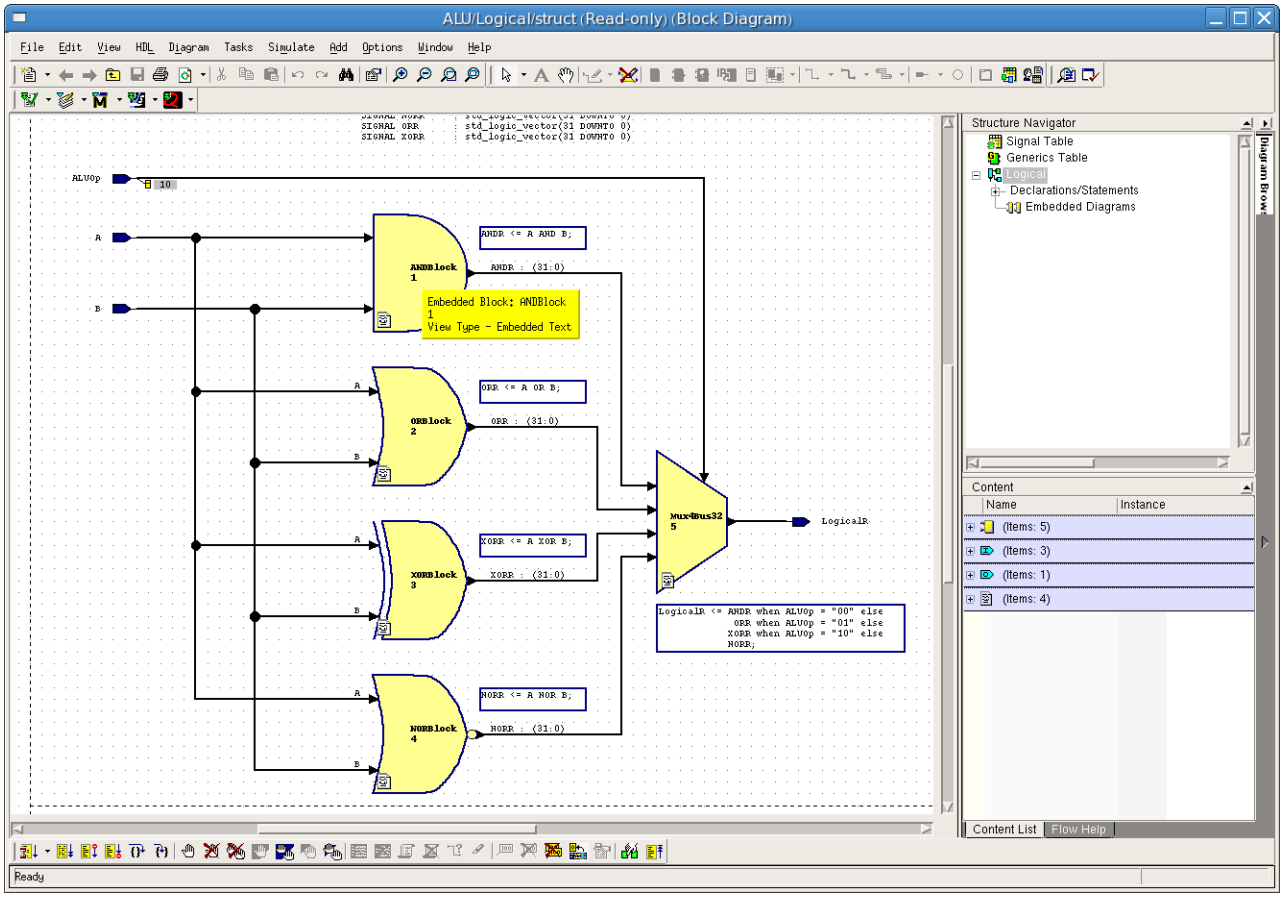


Now to show a transition on the output, let us change up the values of the inputs. Set *ALUOp* to "10", *A* to X"00000000", and *B* to X "FEDCBA98" and then run the simulator for 20ns.

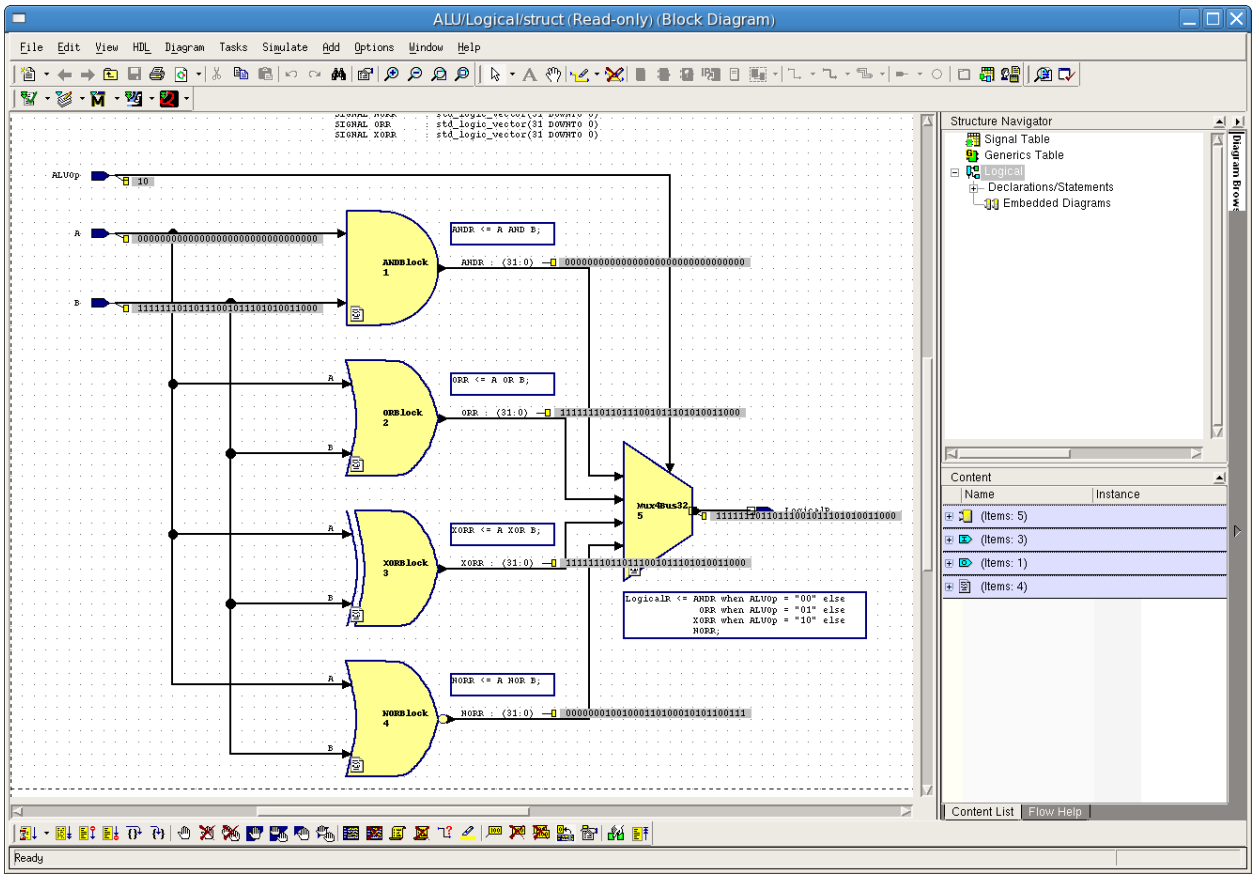
The waveform display should now show the update values for ALUOp and B transitioning at 10ns as well as new values for NORR = X"01234567", ORR= X"FEDCBA98", XORR= X"FEDCBA98", and LogicalR = X"FEDCBA98" also transitioning at 10ns. Check to make sure that your waveform window looks like the figure below. If not, then there is an error in your design.



In addition to viewing the simulator output in the various **ModelSim** windows, data is annotated to the original **FPGA Advantage** design. Go to your opened *Logical* block diagram. If you have no signals selected, at the bottom of the window on the simulation toolbar, there will be a grayed out button labeled **Add Probe**,  . By selecting a signal or bus in the design, this button will change color and allow you to activate the tool:  . Activate this tool, move the mouse pointer over the *ALUOp* bus, and left-click. This will add a probe to the net, meaning that the **current** simulator value of that signal will be displayed in a red box near the net. Since "10" was *ALUOp's* value, a "10" should appear by your probe. The design area should now look like the figure below:



Now add probes to all of the signals on the diagram so that it looks like the figure below:



Now we have seen several ways to look at the values in the simulator. However, it is still a tedious process to individually type all of the stimulus commands for a complete design test. To this end, we can use something called a *Macro File*. A macro file is simply a text file which can be created by a program such as notepad containing a sequence of **ModelSim** commands. In **ModelSim** these are usually saved with the extension *.do* because they are also known as *do* files and are executed via the *do* command.

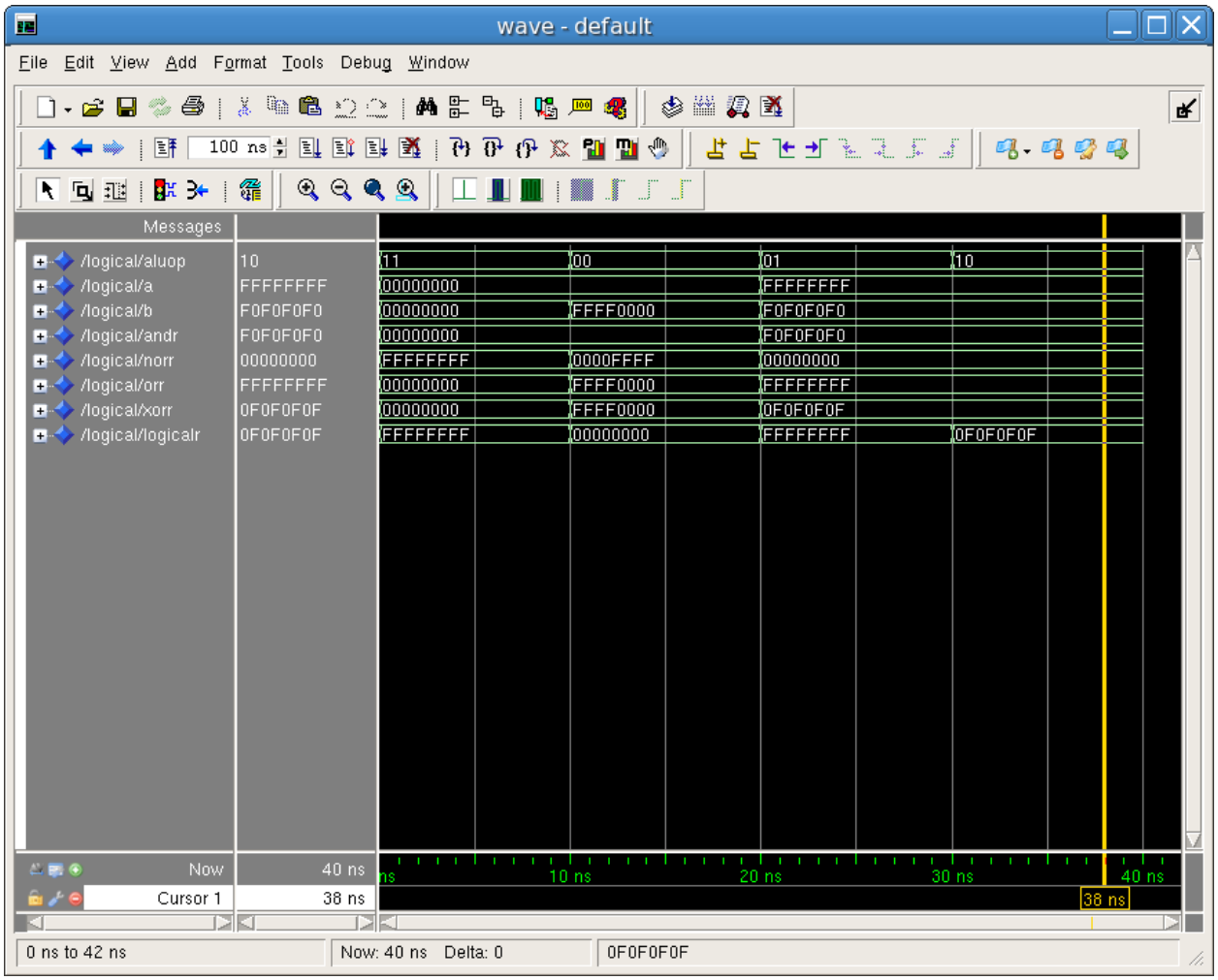
To test the *Logical* sub-block a little bit more fully, a small sequence of simulator commands with comments are listed below. Enter this sequence of commands in a text editor save it as *LogicalTest.do* in your work directory. To execute the ".do" file, you must include the full path location.

```
-- Restart the simulator
restart -f

-- First set A and B to zero and the ALUOp to NOR (11).
-- Check that NORR and LogicalR are both "FFFFFFFFFFFFFFFF" and
-- that ANDR, ORR, and XORR are all "0000000000000000".  This
-- will verify one test case for each individual logic
-- operation and will verify that the "11" select of the
-- multiplexor is working.

force ALUOp "11"
```

```
force A X"00000000"
force B X"00000000"
run 10


-- Now leave A as zero, set B to "FFFF0000" and set the
-- ALUOp to "00".  Check that ANDR and LogicalR are
-- both "00000000".  Since ANDR is the only intermediate
-- result that should be zero, if LogicalR is also
-- zero, then the multiplexor should be working for "00"
-- on the select.  ORR and XORR should be "FFFF0000" and
-- NORR should be "0000FFFF".

force ALUOp "00"
force A X"00000000"
force B X"FFFF0000"
run 10


-- Now we will set A to "FFFFFFFF" and B to "F0F0F0F0".
-- This will give a unique answer for ORR of "FFFFFFFF"
-- so we will set ALUOp to "01".  Check that ORR and
-- LogicalR are "FFFFFFFF".  ANDR should be "F0F0F0F0",
-- NORR should be "00000000" and XORR should be "0F0F0F0F".

force ALUOp "01"
force A X"FFFFFFFF"
force B X"F0F0F0F0"
run 10


-- Since there is a unique value for the only ALUOp which
-- we haven't tested on the multiplexor with the
-- current values of A and B, we can leave them alone
-- and set the ALUOp to XOR or "10".  This time all that
-- we need to check is that LogicalR is now equal to
-- the value of XORR, or "0F0F0F0F".

force ALUOp "10"
run 10
```

Once you have entered and saved this text we are ready to run the macro file. Type *do LogicalTest.do* at the prompt in the **Transcript** window.

Confirm that your design is functioning properly by checking the actual values in the Wave window with the expected results. The final output should look something like the figure below:



With **Modelsim** you have verified the functionality of your first design unit. In the next tutorial, we will create the Shifter design unit.