

CSEE W4840 Embedded System Design Lab 1

Stephen A. Edwards

Due January 25, 2007

Abstract

Learn to use the Altera Quartus development environment and the DE2 boards by implementing a small hardware design that displays and modifies the contents of a small memory.

1 Introduction

Use the Altera DE2 board to implement a simple hardware design. Describe its behavior using the VHDL language and use Altera's Quartus tools to synthesize and program the FPGA device. Use a VHDL simulator to verify and debug the design.

The circuit you program into the FPGA will display and modify the contents of a 16×8 bit RAM. Although there are dedicated RAM chips on the DE2 board, for simplicity use a RAM inside the FPGA. Use four pushbuttons as inputs and three seven-segment LED displays as outputs. Two push buttons should step up and down through the sixteen RAM locations; the other two should increment and decrement the contents of the currently-displayed memory location. One seven-segment display should show the current address (0–F), two others should display the contents of that location in hexadecimal (00–FF).

You will learn to set up a project in the Altera Quartus tool, run a VHDL simulation, and compile and download your design to the FPGA. VHDL is a hardware description language, and the process of using it is very different than developing programs in C++ or Java. You will need these skills in later labs and while you are developing your project.

Below, we introduce the DE2 board, show how to start a new project from a template, add VHDL code to a project, simulate it, and compile and download a design to the FPGA.

2 The DE2 Board

Figure 1 shows the Altera DE2 board. It consists of an Altera Cyclone II FPGA connected to a variety of peripherals including 512K of SRAM, 4 MB of Flash, 8 MB of SDRAM, VGA output, Ethernet, audio input and output, and USB ports. For this lab, we will use four of the eight seven-segment LEDs and the four blue pushbuttons. There are three USB connectors on the top of the board. The leftmost one—the one nearest the 9V DC connector—is for connecting the Altera “Blaster” cable to the workstation. It is through this connection that the FPGA will be programmed, that debugging information flows, etc. The other two USB ports can be used in projects.

The DE2 board holds two quartz crystal oscillators (clock sources: little silver boxes labeled with their frequencies). We will use the 50 MHz clock for this lab; there is also a 27 MHz clock designed for VGA timing.

The DE2 board has built-in configuration for testing and demonstration purpose. You can verify the board is working

properly by observing this default behavior. Use the following procedure to power up the DE2 board.

First, connect the USB blaster cable from the USB port on the workstation to the USB Blaster connector on the DE2 board. Next, connect the 9 V adapter to the DE2's power connector at the top left corner. Third, verify the RUN/PROG switch on the left edge of the DE2 board (just to the left of the LCD display) is in the RUN position.

Power on the DE2 board by pressing the red ON/OFF switch in the upper left corner. The LEDs should flash and the LCD should display “Welcome to the Altera DE2 Board.”

To download our design and override the default configuration of the FPGA, we use a JTAG port (JTAG is a ubiquitous standard that stands for the IEEE Joint Test Action Group). The Altera Quartus tool running on the workstation sends the configuration bit stream through the USB cable to the Cyclone II FPGA. Once programmed, the FPGA retains its configuration as long as power is applied to the board; it is lost when the power is turned off. We cover the details of this process below.

3 Getting Started with Quartus

Quartus is Altera's development environment for FPGAs. It consists of an IDE and a “compiler” that can translate circuits described in VHDL into configuration data for the FPGA. Start the Quartus IDE by running the quartus command. This requires the PATH and LM_LICENSE_FILE environment variables to be set.

Altera provides a variety of reference designs for the DE2. For lab 1, we modified the *DE2_Top* design slightly, which contains information about what each pin on the FPGA is connected to and a top-level VHDL module with a port for each pin.

Download the lab1.tar.gz file from the class website and extract it with “tar xzf lab1.tar.gz” This will place the project files, listed in Table 1, in the current directory.

DE2_TOP.qpf is the top Quartus project file. To open the project file, use File→Open Project and select *DE2_TOP*. Once the project is opened, you can see and change I/O pin assignments with Assignment→Pins. Figure 2 shows this dialog.

Table 1: Files in the DE2_TOP project

Name	Role
DE2_TOP.qpf	Quartus Project File
DE2_TOP.qsf	Pin assignments, etc.
DE2_TOP.vhd	Top-level VHDL file

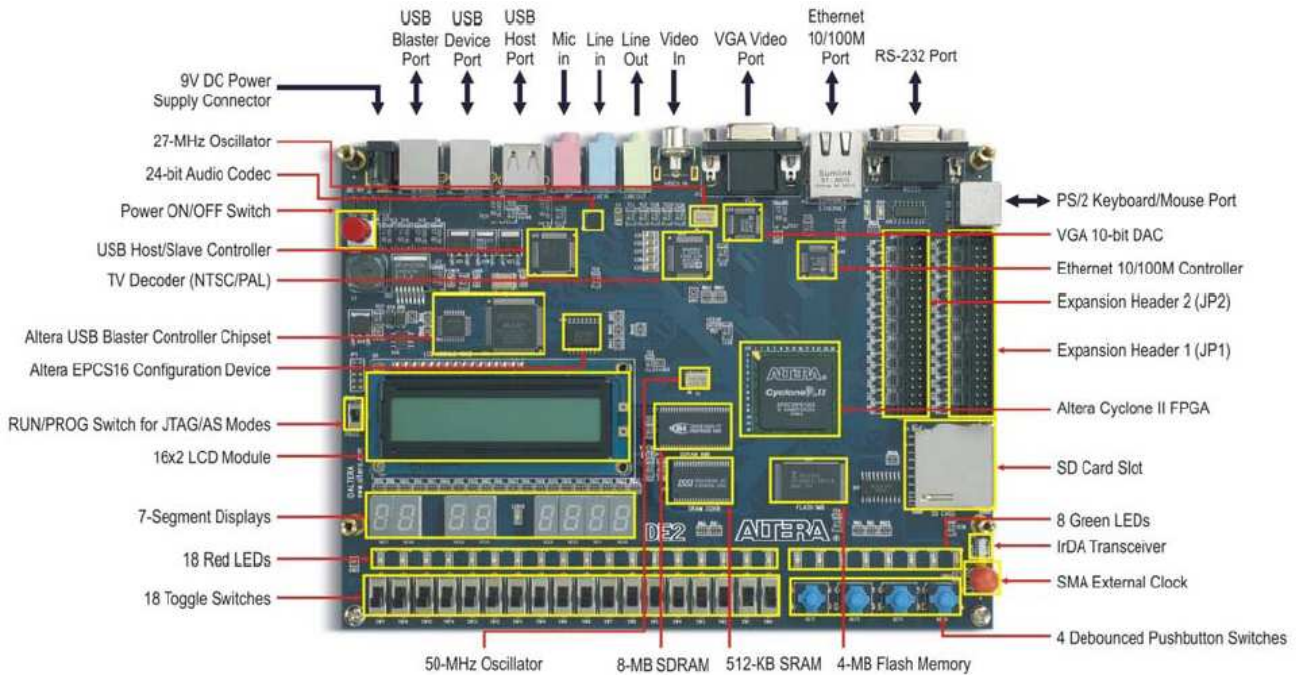


Figure 1: The Altera DE2 board

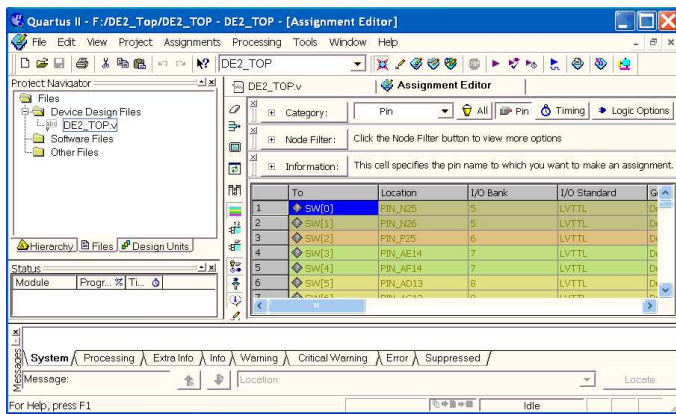


Figure 2: Assigning Pins Textually

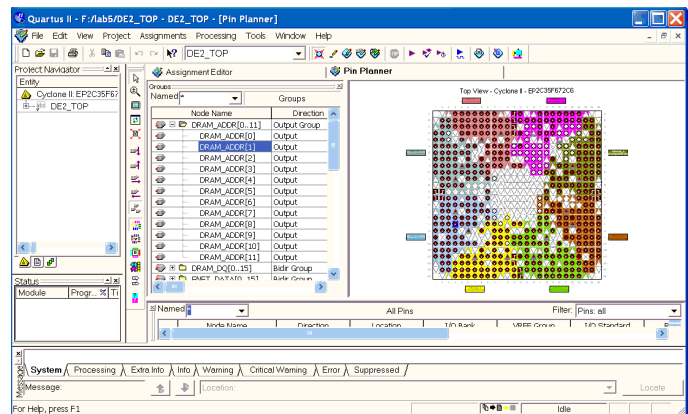


Figure 3: Assigning Pins Graphically

For Quartus to configure an FPGA, it must know which pins on the FPGA perform what roles (i.e., what each is named). This information is board-specific since the pins on the FPGA can be wired to arbitrary peripherals. The *DE2_TOP.qsf* file contains this information for the DE2 board.

DE2_TOP.vhd is the top-level VHDL module for the project, which mostly lists the top-level ports, i.e., the VHDL names for the pins. It also sets the state of the LEDs.

Although you do not need to modify I/O pin settings for this lab, you may need to do so in the future. Assignment→Pin Planner, shown in Figure 3, opens a display that shows the physical location the pins on the FPGA and their assignments.

4 Compiling for the FPGA

The supplied project can be compiled and downloaded to the board, although it does not do much. First, make sure all the

source files are included in the project. From the Project navigator window, click on the *Files* tab and then click to expand *Device Design Files*. The tool will display the VHDL files that will be compiled into the FPGA. To add a file, select Project→Add/Remove Files in Project. This opens the window in Figure 4.

Select all the VHDL files from the pop up window. If you have written multiple VHDL files, add each of them. Do not add any test benches (used for simulation) to the list of device design files since they cannot be compiled into hardware.

Now we are ready to compile. Select Processing→Start Compilation to start the compilation process (Figure 5). The window on the left reports progress.

A pop-up appears when compilation completes. If there are errors, use the Messages window to locate them (Figure 6). As usual, the first error listed is most trustworthy; any others may

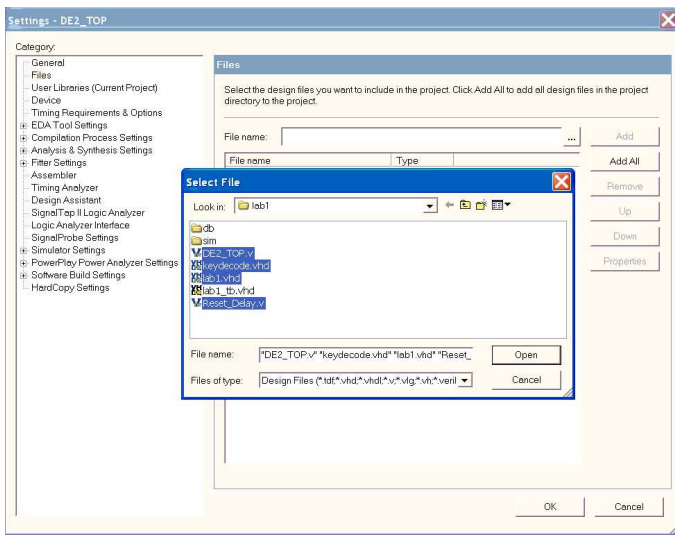


Figure 4: Adding files to a project

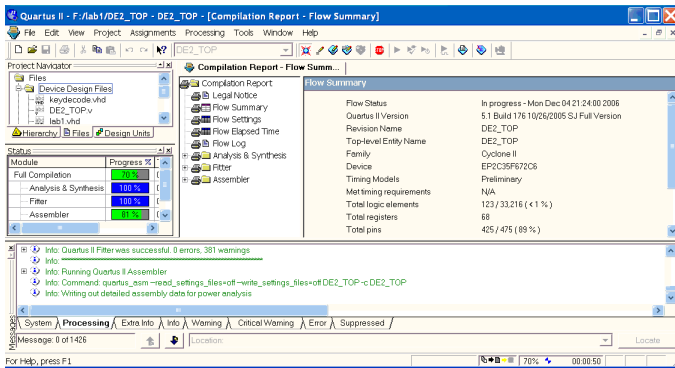


Figure 5: Compiling a design

be due to earlier errors. A compilation process usually generates many, many warnings. These can generally be ignored.

Double clicking on an error message will highlight the suspect VHDL in the editor window. The compiler may also display warning messages, which can be explored in the same way. You can obtain more information about a specific error or warning by selecting it and pressing the F1 key.

4.1 Programming the FPGA

Once your design has been compiled, it can be downloaded to configure the FPGA. Select Tools→Programmer, which will display the window in Figure 7. It should list the DE2_TOP.sof file to be programmed into the EP2K35F672 device (Altera's charming name for the FPGA on the DE2).

You may have to click on the “Hardware Setup...” button and select the USB-Blaster cable.

Click the check box under *Program/Configure* for the DE2_TOP.sof file destined for the FPGA and then click *Start* to download your design to the FPGA. If all goes well, the design should spring to life.

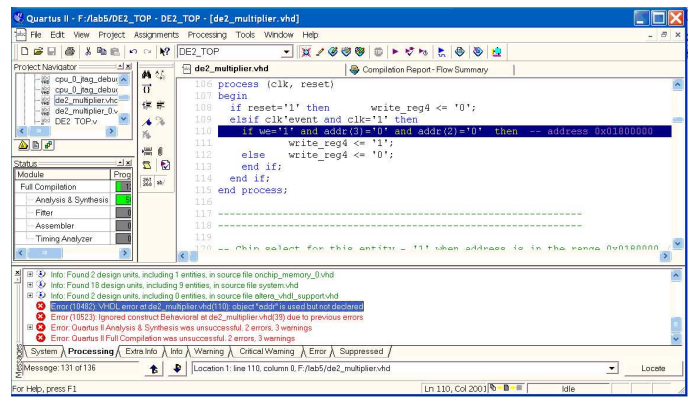


Figure 6: Diagnosing errors

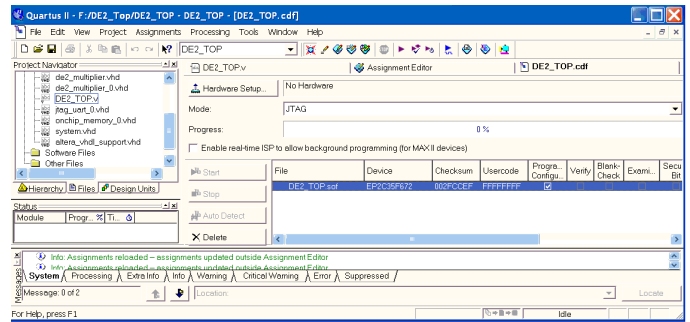


Figure 7: Programming the FPGA

5 Editing VHDL

The next step is to code your circuit in VHDL. Quartus provides a good VHDL text editor, which provides syntax highlighting, language templates, and other aspects of a good IDE. To create a new VHDL file in your project, select File→New. This will bring up the dialog in Figure 8.

Select the VHDL file option and click OK. This brings up a window where you can enter VHDL code (Figure 9).

The verbose syntax of VHDL is probably unfamiliar to you. To help, the Quartus tool provides a collection of VHDL templates, which provide examples of various types of VHDL constructs, such as an entity declaration, a process statement, and an assignment statement.

To use a VHDL template, select Edit→Insert Template. This will open a window such as Figure 10.

Select “VHDL” and the type of template you want. The OK button inserts the template in the active source file. Then fill in the details in the template, such as the name of an entity.

5.1 The RTL Viewer

We are designing a circuit but have been writing textual VHDL. Quartus includes an RTL viewer that displays your design as a schematic. Bring this up by selecting Tools→RTL Viewer (Figure 11). Note that this is merely informative and not a necessary part of the compilation process.

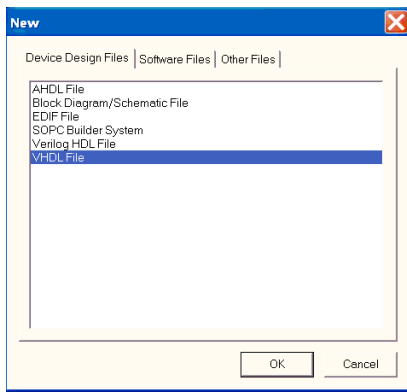


Figure 8: Creating a new VHDL file

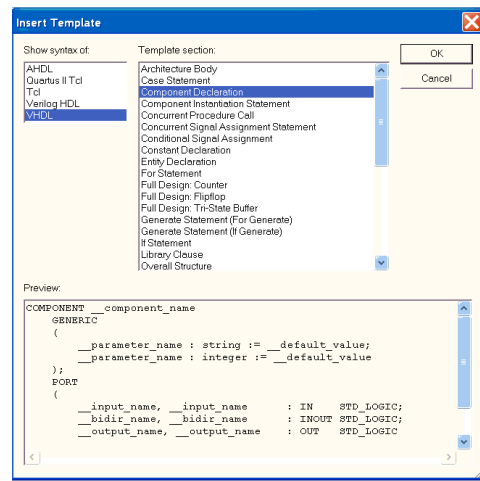


Figure 10: Inserting a VHDL template

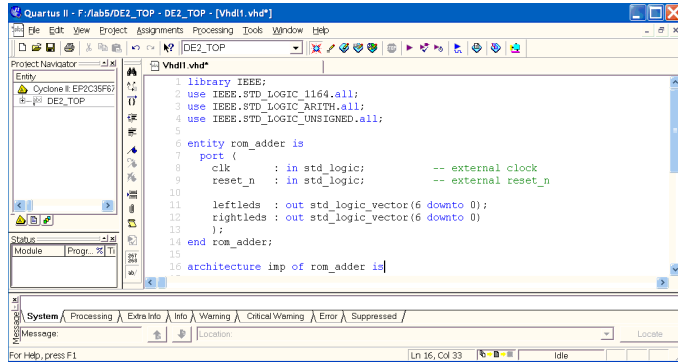


Figure 9: Editing a VHDL file

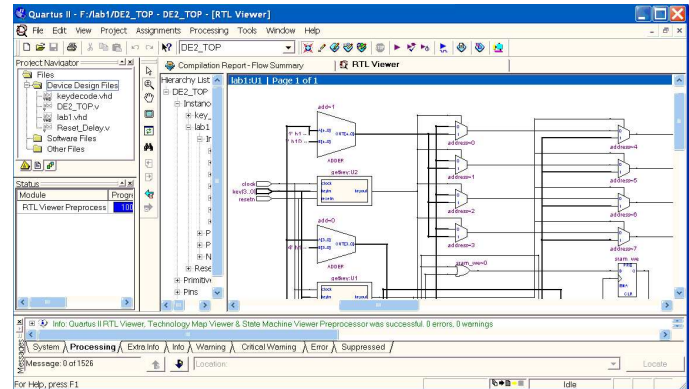


Figure 11: Seeing your design as a schematic

6 The Lab 1 Design

Your goal for this lab is to implement a memory display/modification circuit whose block diagram is shown in Figure 12. Input ports are on the left of this diagram; output ports are on the right.

To put the lab1 component into the project, you need to instantiate it in the top-level architecture in DE2_TOP.vhd. You need to do two things: declare its interface as a component and instantiate it.

The component declaration and port map must appear in the declarations of the architecture section of DE2_TOP.vhd:

```
architecture datapath of DE2_TOP is
component lab1 is
port (
clock : in std_logic;
resetn : in std_logic;
key : in std_logic_vector(3 downto 0);
hex6 : out std_logic_vector (6 downto 0);
hex5 : out std_logic_vector (6 downto 0);
hex4 : out std_logic_vector (6 downto 0)
);
end component;

begin

U1: lab1 port map
(
clock => clock_50,
resetn => sw(0),
key => key,
hex6 => hex6,
hex5 => hex5,
hex4 => hex4
```

```
);
end datapath;
```

Here, the module ports from lab1 are mapped to top level ports. The naming of these top level ports, such as CLOCK_50, SW, KEY and HEX6 4, are all defined in the Quartus .qsf file. The ports named in the DE2_TOP.vhd and QSF file must match.

Remember to disable the constant assignments to HEX4, HEX5, and HEX6 in the DE2_TOP.vhd file when you add your lab1 component.

6.1 RAM

Your design should include a 16 × 8 bit RAM, but what kind of RAM? The DE2 board contains an SRAM chip, an SDRAM chip, and RAM internal to the FPGA itself. The SDRAM chip provides the highest capacity but requires a very complicated controller. The SRAM chip is smaller, much simpler to use, and provides more storage than RAM on the FPGA. However, RAM internal to the FPGA, so-called “block RAM,” is the easiest to use (and the fastest). We will use it for this lab.

The FPGA block RAM can be configured many different ways, e.g., as one big memory, as many small regions, and as bits, bytes, or words. The easiest way to ask for a particular

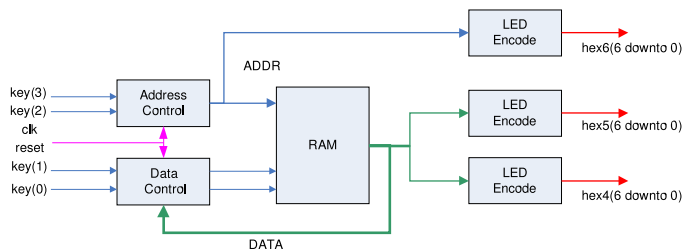


Figure 12: The block diagram of lab 1

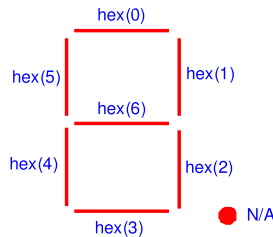


Figure 13: A seven-segment LED display

type of RAM is to allow the Quartus tool to infer it from the use of an array in VHDL. Below is code from which Quartus will infer a small RAM block.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity raminfr is
  port (
    clk : in std_logic;
    we : in std_logic;
    a : in std_logic_vector(3 downto 0);
    di : in std_logic_vector(7 downto 0);
    do : out std_logic_vector(7 downto 0)
  );
end raminfr;

architecture rtl of raminfr is
  type ram_type is array (16 downto 0) of
    std_logic_vector (7 downto 0);
  signal RAM : ram_type;
  signal read_a : std_logic_vector(3 downto 0);

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(a)) <= di;
      end if;
      read_a <= a;
    end if;
  end process;

  do <= RAM(conv_integer(read_a));

end rtl;
```

Here, the *conv_integer* function converts an input in the form *std_logic_vector* to an integer index for accessing the array.

6.2 Seven-Segment LEDs

The block diagram in Figure 12 includes three seven-segment LED output decoders. Each segment of each LED is connected to a pin on the FPGA. Driving a pin low (to 0) lights the cor-

responding segment. Figure 13 shows how the segments are arranged. Thus, to display a “1,” drive the port to “1111001.”

7 VHDL Simulation

Hardware is usually much harder to design than software for a variety of reasons. One is that that usual edit-compile-debug cycle is longer because it takes longer to compile hardware. Another reason is that the behavior of hardware is harder to observe. It is difficult to put a print statement in hardware.¹ It is even harder to probe a wire inside a chip.

One way out of this conundrum is to simulate VHDL before compiling it onto the FPGA. This is faster than compilation and makes it easy to observe everything going on inside your design, but it is much, much slower the running the actual hardware.

7.1 Testbenches and the Synthesizable Subset

There is actually two versions of VHDL: the complete language, which the simulator accepts; and the synthesizable subset—the part of the language that can be translated into hardware. The non-synthesizable part of the language is mostly useful for writing testbenches. Note that traditional programming languages such as C or Java do not have the notion of a subset.

You need two things to run an interesting simulation of a system: a description of the system itself and some input for it. This latter component is known as a testbench and you need to write additional VHDL for it when you simulate a design. A testbench instantiates the desing you are testing, stimulates the design, e.g., by applying clocks and inputs, and monitors its response. Think of a test bench as a virtual signal generator and oscilloscope.

A testbench can use non-synthesizable VHDL statements. The *wait* statement, which can delay a precise amount of time, is typical. It is not possible to build hardware that does this (although you can build something that delays a precise number of clock cycles), but the simulator can easily handle this. For example, *wait* can be used to provide a reset signal that goes low for 200 ns:

```
process
begin
  resetn <= '0';
  wait for 200 ns;
  resetn <= '1';
  wait;
end process;
```

The final *wait* stops the process so it does not automatically repeat and generate multiple resets.

Wait is also useful for modeling clocks. Here is a way to generate a clock with a 40 ns period.

```
process
begin
  clock <= '0';
  wait for 20 ns;

  loop
    clock <= '1';
    wait for 20 ns;
    clock <= '0';
    wait for 20 ns;
  end loop;
end process;
```

¹But not impossible: designers often use LEDs as one-bit debugging outputs.

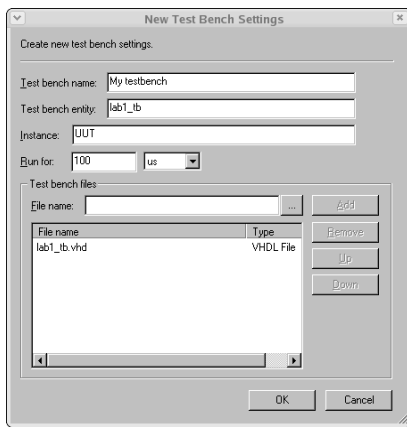


Figure 14: Telling Quartus about a new test bench

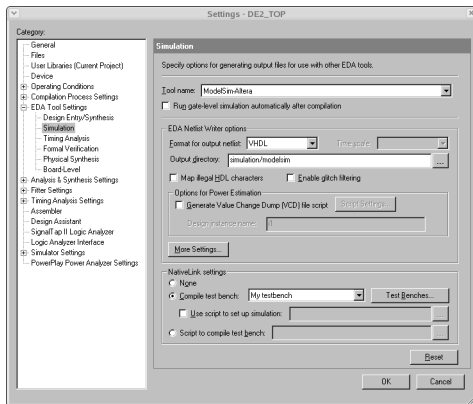


Figure 15: Selecting ModelSim-Altera as the simulator and telling it about a testbench

The *loop* statement tells the simulation to generate clock pulses forever.

Wait can also be used to separate assignment statements to generate specific input stimulus.

```

process
begin
wait for 100 ns;
a <= '0';
b <= '0';
cin <= '0';
wait for 20 ns;
a <= '1';
b <= '0';
cin <= '0';
wait for 20 ns;
a <= '1';
b <= '0';
cin <= '1';
wait;
end process;

```

You can test this lab by using this style of code to emulate buttons being pressed.

7.2 Simulating your design

Quartus can run an external VHDL simulator. We will use a version of Mentor Graphics’s *ModelSim*. It is a hassle to run the simulator the first time, but it is much easier the second.

First, you probably need to tell Quartus where the simulator is. Go to Tools→Options, select “EDA

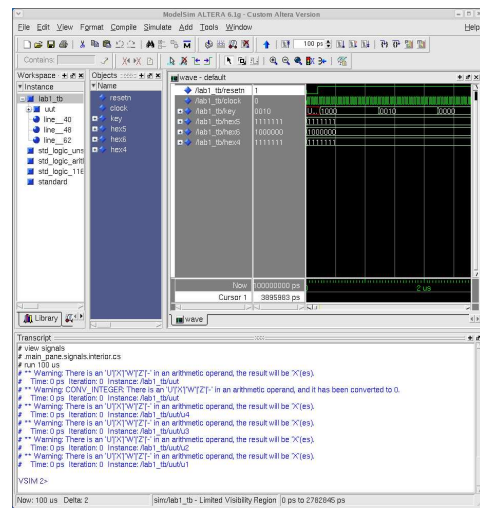


Figure 16: Running ModelSim and observing simulation results

Tool Options,” double-click on the ModelSim-Altera line and enter the name of the directory in which the “vsim” executable resides. On our machines, this is /usr/cad/altera/6.1/modelsim_ae/modeltech/linuxaloem.

Now, tell Quartus that you want to use ModelSim-Altera as the “EDA simulator.” With the project open, select Assignments→EDA tool settings and click on “Simulation.” Set “Tool name” to “ModelSim-Altera.”

The Assignments→EDA tool settings dialog is also where you must tell the simulator which testbench to use. Again under EDA Tool Settings→Simulation, specify a testbench in the “NativeLink settings” area by selecting “Compile test bench” and clicking on Test Benches.

In the Test Benches dialog, click New to create a new test bench. The name is arbitrary, but the entity name must match that in your VHDL test bench file and the instance should be the name of the instance of the design you are testing. You must also specify an execution time for your testbench. This may be a number of *μs*. Finally, add the VHDL file for your testbench by selecting it and clicking “Add.” Figure 14 illustrates this.

Once you have created a new test bench, you can select it in the pulldown menu to the right of “Compile test bench.” Figure 15 illustrates all of these settings.

Finally, you should be able to select Tools→EDA Simulation Tool→Run EDA RTL Simulation to start ModelSim.

If all goes well, you should see the ModelSim window appear and a waveform viewer display the results of the simulation: Figure 16. Use the zoom tools to zoom in and out on this display and the scrollbars to move. By default, the display will show all the signals external to the unit under test (i.e., on the entity in your VHDL test bench file you specified earlier).